Lic. Manuel A. Torres Remon

DESARROLLO DE APLICACIONES CON JAVA

JCreator - JDeveloper - NetBeans

- Fundamentos de programación
- Estructura de secuencia
- Estructuras de selección
- Programación modular
- POO
- Arrays

Incluye CD Multimedia

EMPRESA EDITORA MACRO®

EDITORIAL MACRO

DESARROLLO DE APLICACIONES CON JAVA

JCreator - JDeveloper - NetBeans

Desarrollo de aplicaciones con Java
Autor: Manuel Torres Remon

Edición a cargo de:
Empresa Editora Macro EIRL
Av. Paseo de la República 5613, Miraflores
Lima, Perú
(511) 748-0560
ventas@editorialmacro.com
http://www.editorialmacro.com

Primera edición: Abril 2013
Primera reimpresión: Mayo 2014

Impreso en los Talleres Gráficos de
Empresa Editora Macro EIRL
Lima, Perú

ISBN N.º 978-612-304-101-4
Hecho el Depósito Legal en la Biblioteca Nacional del Perú N.º 2014-06961

Manuel Torres Remon

Manuel Torres es consultor, dedicado a la docencia de cursos de tecnología como Visual NET y Microsoft SQL Server. Asimismo, ha brindado capacitación en las instituciones más importantes de Lima, Perú.

Su formación tecnológica la realizó en el Instituto Superior Tecnológico Público Manuel Arévalo Cáceres (IESTP MAC) y en la Universidad Alas Peruanas (UAP). En ambas instituciones, recibió una adecuada formación profesional; la cual se demuestra en las diferentes instituciones que labora.

Actualmente, se desempeña como consultor tecnológico de grandes empresas como Topitop, Nestlé Perú. Además, es docente en instituciones educativas como los institutos Manuel Arévalo Cáceres, Cibertec y Unimaster de la Universidad Nacional de Ingeniería (UNI). En todas ellas, imparte cursos de tecnología, especialmente en las áreas de Análisis, Programación y Base de datos.

Por otro lado, ha publicado libros como Programación VBA con Excel, Programación orientada a objetos con Visual Basic 2012 y Programación Transact con SQL Server 2012 los cuales aportan como material de apoyo a programadores, donde se exponen casos prácticos y claros sobre temas que no han sido desarrollados aún en otras publicaciones.

Si tuvieran alguna consulta sobre el material expuesto, escriba al e-mail manuel.torresr@hotmail.com o a los teléfonos (511) 982 360 540 / (511) 996 396 023.

Dedicatoria

Para mis ojos, mis brazos y piernas: Ángela-Victoria y Fernanda-Ximena, mis pequeñas y adoradas hijas.

A la gran familia Macro por confiar nuevamente en mí. Por último, quiero agradecer a mi esposa Luz que siempre me apoya y comprende en lo que me propongo y cuando nota que me desmotivo, me empuja a seguir trabajando.

Introducción

El presente libro titulado Desarrollo de aplicaciones con Java tiene como objetivo dar a conocer las funciones dadas por este lenguaje de programación, tanto a principiantes como a expertos en la materia; ya que los temas se desarrollan progresivamente para iniciarse en la programación y se incorporan herramientas importantes para el desarrollo de aplicaciones. Está dirigido, principalmente, a desarrolladores, estudiantes de programación Junior, etc.

Se pensaba que Java era un lenguaje complejo, pesado y aburrido; pero actualmente Java es considerado como uno de los lenguajes de programación más usados a nivel mundial y es que ante los abruptos cambios de hoy, las nuevas tecnologías se hacen cada vez más populares, como es el caso de Java. Esto genera que las personas se adapten más rápido a dichas tecnologías, dando paso así a la necesidad de que estas evolucionen constantemente. Ante dichos cambios, uno debe estar preparado para entender y programar el código Java, no es complicada la tarea, tómelo como un reto ya que en programación todo está basado en la lógica y cuan ordenado es usted al programar. Como podrá notar no son muchos los requisitos que se exigen.

Java está relacionado en cierta forma con otros lenguajes de programación como C++. Sin embargo Java cada vez se posiciona más como el elegido preferido para el desarrollo de todo tipo de aplicaciones tanto Web como Empresariales, lo cual hace que sea un lenguaje potente y confiable.
¿Por qué optar por Java? Java como lenguaje presenta (des)ventajas como: es totalmente libre, no necesita licencia y lo puede desarrollar en cualquier aplicación que lo puedo interpretar como NetBeans, Eclipse, JDeveloper o JCreator; por otro lado, es utilizado ampliamente en un sinnúmero de plataformas, como pueden ser Windows, Mac o Linux; asimismo, Java ofrece seguridad contra infiltraciones y virus; cuenta con una gran biblioteca de funcionalidades estándar; y por último, Java es compatible con las tendencias actuales de las aplicaciones para el Internet.

En cuanto a su estructura, Desarrollo de aplicaciones con Java está preparado para usuarios iniciales sin experiencia en programación hasta desarrolladores en Java. El libro se distribuye en 12 capítulos cada uno con casos desarrollados y explicados. La forma secuencial de los capítulos ha sido diseñada para un mejor entendimiento de una aplicación Java. En otras palabras, se emplea una metodología práctica, didáctica y visual que facilita el aprendizaje del lector de una manera clara, sencilla y directa; además contiene Casos desarrollados y problemas prácticos que el usuario debe seguir paso a paso para desarrollar sus propios ejercicios.

Por último, se adjunta un CD como material de apoyo donde se muestran algunas aplicaciones probadas y registradas, con el fin de facilitar la comprensión de estos temas.

Índice

CAP.

1

Introducción al lenguaje Java

CAPACIDAD:

- Reconocer los conceptos básicos del lenguaje Java.
- Configurar e instalar las aplicaciones Java en una computadora.
- Reconocer, diferenciar e instalar los diferentes IDE para las aplicaciones Java.

CONTENIDO:

1.1. INTRODUCCIÓN

Muchos de nosotros hemos escuchado acerca de Java sin saber que probablemente nuestras vidas están girando alrededor de este término, veamos un ejemplo simple; si usted tiene un celular en sus manos las aplicaciones con las que cuenta pueden ser Java, pero a qué se debe que Java esté en aparatos de uso diario existiendo otros lenguajes como Visual Basic, lo que diferencia a Java de otras aplicaciones es la portabilidad con la que cuentan sus aplicaciones. Otra característica favorable de Java es que cuenta con un texto plano lo que hace menos pesadas sus aplicaciones y que estas se puedan ejecutar en aparatos de poca memoria.
Java, en la actualidad, ha revolucionado la programación y ha promovido un código derivado de C y C++ que en corto tiempo ha tenido mucho éxito en las tecnologías de información debido a su filosofía y la forma de operar de sus **aplicaciones cliente-servidor,** tanto en plataforma como en la web. Por tanto, son muchas las empresas que están apostando por Java como un potente lenguaje de programación duradero en el tiempo y 100% confiable.
En este primer capítulo se verá a Java desde una perspectiva global, conociendo su historia, sus principales características y cómo se trabaja la programación orientada a objetos en Java.

1.2. ORÍGENES DEL LENGUAJE JAVA

Java fue diseñado por James Gosling licenciado en Ciencias de la Computación de la Universidad de Galgary, cuando era empleado de *Sun Microsystem* este es reconocido como el diseñador e implementador del compilador y la máquina virtual de Java (JVM), por lo que fue elegido miembro de la Academia Nacional de Ingeniería de Estados Unidos (NAE).

Inicialmente Java fue diseñado para dispositivos electrónicos relativamente pequeños como la calculadora, el microondas, el refrigerador y la televisión interactiva. Debido a la existencia de múltiples tipos de electrodomésticos y a los constantes cambios en los mismos era necesario una herramienta que no dependiera del tipo aparato, es decir, se necesitaba un código neutro; la idea principal era ejecutar los códigos en una máquina virtual que lo hiciera portable convirtiendo el código en uso particular por el electrodoméstico.

Inicialmente, Java no fue acogido por las empresas de electrodomésticos, siendo así que en el año 1995 Java se introduce como lenguaje de programación para computadoras, la incorporación de un intérprete Java en la versión 2.0 del navegador Netscape hizo de Java un lenguaje potente revolucionando la internet.

Java se creó como una herramienta de programación para ser usada en un proyecto de *set-top-box* en una pequeña operación denominada “The Green Project” en Sun Microsystems en el año 1991. El

equipo (*Green Team*), compuesto por trece personas y dirigido por James Gosling, trabajó durante 18 meses en Sand Hill Road en Menlo Park en su desarrollo.

Java como lenguaje se le denominó inicialmente "Oak" del castellano roble, luego pasó a denominarse Green tras descubrirse que "Oak" era ya una marca comercial registrada en Estados Unidos para adaptadores de tarjetas gráficas y finalmente, se renombró a Java.

Para el nombre que quedó registrado al final se han escrito muchas hipótesis, sobre el término algunos señalan que podría tratarse de las iniciales de sus creadores: James Gosling, Arthur Van Hoff, y Andy Bechtolsheim. Aunque la hipótesis que más fuerza tiene es la que Java debe su nombre a un tipo de café disponible en la cafetería cercana a la empresa Sun, de ahí que el ícono de Java sea una taza de café caliente. Un pequeño indicio que da fuerza a esta teoría es que los 4 primeros bytes (el número mágico) de los archivos .class que genera el compilador son en hexadecimal, 0xCAFEBABE. Otros afirman que el nombre fue sacado al parecer de una lista aleatoria de palabras. A pesar de todas estas teorías, ninguna ha podido ser comprobada.

La promesa inicial de Gosling era: "*Write Once, Run Anywhere*" ("escríbelo una vez, ejecútalo en cualquier lugar"), se cumplió al poder Java proporcionar un lenguaje independiente de la plataforma y un entorno de ejecución (la JVM) ligero y gratuito para las plataformas más populares. Es decir, que los binarios (bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma.

1.2.1. Comparando Java con C++

Partiendo del paradigma orientado a objetos, el lenguaje C++ es capaz de interpretar dos estilos de programación como son la programación estructurada y la orientada a objetos, a esta característica se le llama programación hacia atrás; este término no es un estereotipo de Java, puesto que este programa trabaja totalmente orientado a objetos sin perder la esencia del paradigma. Hay que reconocer que Java hereda de C++ muchas características como el punto y coma al finalizar una instrucción o su forma de declarar una variable; si usted programó en C++ se sentirá familiarizado con Java.

Ahora, comparar dos lenguajes de programación resulta un tanto difícil si comparamos que ambos lenguajes tienen un objeto parecido, el cual es el desarrollo de aplicaciones.
Veamos algunas similitudes entre los dos lenguajes:

- Presentan un lenguaje enriquecido de instrucciones que hacen a los algoritmos más complejos sencillos de expresarlos.
- Presentan consistencia y falta de ambigüedad en sus instrucciones.
- Presentan un correcto control sobre los valores multidimensional; así tenemos los arreglos, estructuras y las clases.
- Presentan un desarrollo de aplicaciones basado en la programación modular, permitiendo que la programación se desarrolle en componentes independientes que trabajan bajo un mismo fin.
- Presentan un soporte de interacción con el entorno, es decir, proporcionan entrada y salida a sus aplicaciones.
- Presentan portabilidad de sus aplicaciones haciendo que estas puedan ejecutarse en cualquier plataforma de trabajo.

Ahora veamos algunas características que diferencian a Java de C++:

- En C++ la potencia de sus aplicaciones es el uso adecuado de los punteros que permiten acceder a la memoria volátil de la computadora (RAM) que en muchas ocasiones presentaba errores de colisión en la memoria. Java no presenta esta característica, por tanto evita todo acceso a la memoria volátil. El JDK de Java proporciona métodos de acceso a la memoria de manera efectiva y controlable.
- Para mantener un valor entre procesos C++ usa variables globales, mientras que en Java lo único global es el nombre de sus clases y las variables que necesite hacerlas globales se las declarara dentro de la clase, pero al exterior de sus métodos.
- Java no usa la sentencia GoTo, en vez de ello usa *break* o *continue* para cortar o continuar alguna instrucción respectivamente. Goto hace que el código sea como un "código spaghetti" lleno de saltos, vueltas y revueltas. En Java se espera un orden sin saltos, en cambio; GoTo es como un canguro pegando saltos.

Fig. 1.1

- Las conversiones inseguras de C++ se realizan por medio de los moldeados de tipo (*type Casting*), el cual permite cambiar el tipo de un puntero. Java hace una comparación en tiempo de ejecución de la compatibilidad dejando al programador controlar dichas excepciones.

Finalmente, veamos un cuadro comparativo entre Java y C++.

JAVA	C++
Es totalmente portable.	Parcialmente portable
Controla la memoria dinámica, el *garbage collection* y no tiene punteros.	Memoria automática gestionada por le RTS, memoria dinámica gestionada por el programador.
Administración de métodos virtuales.	Administración parcial de los métodos virtuales y no virtuales.
No cuenta con herencia múltiple.	Aplica la herencia múltiple.
Control de clases en tiempo de ejecución.	Es muy limitada.
Tratamiento genérico de tipos (Wrappers).	Manejo de punteros.

1.3. Los programas en Java

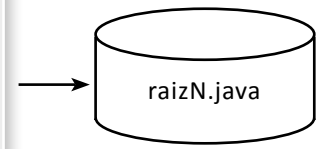
El proceso de programación en Java consta de un conjunto de actividades necesarias para la solución de un problema particular. Programar no es sencillo, hay que tener un control especial de lo que entendió del problema con lo que resultará en un programa Java.

Normalmente son 5 las etapas por la que un programa en Java debe pasar, estas son: Edición, Compilación, Carga, Verificación y Ejecución; de estos 5 mínimamente podríamos nombrar a Edición, Compilación y Ejecución.

1. ***Edición***: en esta etapa el programador digita las instrucciones Java en un editor en el cual podrá corregir alguna parte del código si fuera necesario o grabar el archivo cuando determine que el código es el correcto, cuando esto suceda se creará el archivo con extension .java.

Editor de Codigo

```
public class raizN
{
     public static void main(String[] args){
       int n=8;
       int base=3;
       double raiz=Math.pow(n,1.0/base);
       System.out.println("El numero es:"+n);
       System.out.println("La base es:"+base);
       System.out.println("La raiz es:"+raiz);
     }
}
```

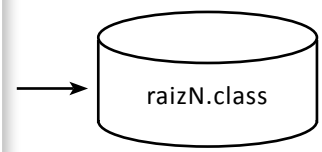


Para el siguiente caso hemos editado un código Java de consola que permite calcular la raíz N de un número entero positivo. La clase principal se llama raizN y tiene un método *void main* que permitirá operar las instrucciones. Al final se guardó el archivo en una unidad de la computadora con el mismo nombre de la clase y con la extensión .java.

2. ***Compilación:*** en esta etapa de transición se crean códigos de bytes y al ser guardados se crea el archivo con el mismo nombre del archivo Java pero con extensión .class.

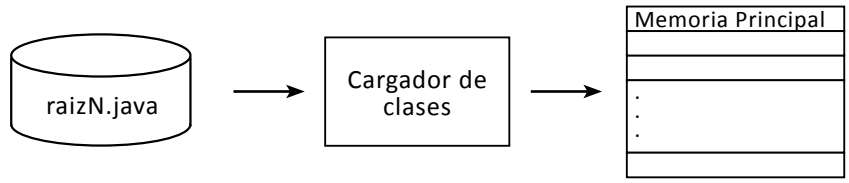
CodeByte

```
03 3b 04 00 0l 1a
05 1b a4 01 1h 5a
b1 32 l4 0f 02 3a
3b 3b 14 1f 0l 56
02 78 l4 00 02 2a
l3 h2 l4 f1 1l 84
01 11 a4 d0 1l 4m
ff 1b 04 a0 02 27
```



A partir de la compilación del archivo Java y creado el archivo .class puede ser incrustado en una página web como un *applet.*

3. ***Carga:*** en esta etapa los códigos de bytes generados son enviados a la memoria de la computadora por medio del cargador clases obtenidos desde la unidad donde se guardaron los archivos java y class.



4. ***Verificación:*** en esta etapa se verifica que el código de bytes sea válido y no contenga violaciones de seguridad Java, enviando el código a un intérprete que tendrá por misión hacer que este código sea entendible por la computadora.

5. ***Ejecución:*** en este etapa el código de bytes depurado es enviado a la Máquina Virtual de Java (JVM) para su ejecución y visualización en el entorno que el usuario crea conveniente de aquí se desprende el termino de portabilidad ya que dependiendo de la JVM podrá ejecutar la aplicación en una u otra plataforma.

1.4. Evolución de la plataforma Java

La evolución de Java es controlada por la Comunidad de Procesos Java (JCP) que usa los JSR's, la cual propone y especifica los cambios dentro de la plataforma Java.

- JDK 1.0 (Enero 1996)

  Contiene 8 paquetes y 212 clases

- JDK 1.1 (Febrero 1997): es la versión más estable y robusta.

  Contiene 2 paquetes y 504 clases
  Mejor rendimiento en la JVM
  Paquete de evento AWT
  Clases anidadas
  Serialización de objetos
  API JavaBeans
  Archivo JAR
  JDBC Java Data Base Connectivity
  RMI Remote Method Invocation

- J2SE 1.2 (Diciembre 1998): también llamada Java 2

  Contiene 59 paquetes y 1520 clases
  JFC Clases Swing
  Java 2D
  API Collections

- J2SE 1.3 (Mayo 2000)

  Contiene 77 paquetes y 1595 clases
  JNDI Java Naming and Directory Interface

API Java Sound
JIT Just in Time
Orientada a la resolución de errores

- J2SE 1.4 (Febrero 2002)

Contiene 103 paquetes y 2175 clases
API de entrada y salida de bajo nivel
Clases collection
Procesador XML
Soporte a criptografía con Java Cryptography Extension (JCE)
Inclusión de Java Secure Socket Extensio JSSE
Inclusión de Java Authentication and Authorization Service (JAAS)

- J2SE 5.0 (Java 5)

Contiene 131 paquetes y 2656 clases
Tipos genéricos
Autoboxing – Unboxing
Enumeraciones
Metadatos en clases y métodos

- Java SE 6 (Diciembre 2006)

Collection Framework
Drag and Drop
Internationalization Support
I/O Support
JAR

- Java Web Start

Java DB 10.2 JDBC 4 Early Access
Java Management Extensions (JMX)
Java Platform Debugger Architecture (JPDA)
Java Virtual Machine Tool Interface (JVM TI)
Reflection
Remote Method Invocation (RMI)
Swing

- Java SE7 (Julio 2011)

Soporte XML dentro del mismo lenguaje
Soporte para closures
Superpaquete

1.5. TIPOS DE APLICACIONES JAVA

El código Java no tiene distinción cuando se trata de las aplicaciones que desarrolla, veamos algunos tipos:

- Aplicaciones Java de Consola
- Aplicaciones GUI
- Aplicaciones Java Applets
- Aplicaciones Servlets
- Aplicaciones JavaBeans

Aplicaciones Java de Consola
Son aplicaciones que permiten concentrarse netamente en la implementación del problema y no en su entorno de desarrollo. Se caracterizan por:

- No permitir el uso del mouse
- Trabajar la ejecución en modo D.O.S.
- No contar con elementos visuales

En la Fig. 1.2. se muestra la ejecución de una aplicación de consola.

Fig. 1.2

Aplicaciones GUI
Son aplicaciones que no necesitan de un navegador ya que se ejecutan como una aplicación de escritorio. La mayoría de las aplicaciones Java están implementadas por una Interfaz Gráfica de Usuario (GUI), por su no exposición de los datos en la web y ser una aplicación interna dentro de una organización.

Fig. 1.3

Aplicaciones Java Applets

Estas aplicaciones se caracterizan por ser pequeñas aplicaciones que se incorporan en una página web y que necesitan de un navegador compatible con Java, en realidad todos los navegadores son compatibles, pero se tiene que descargar un Plugin según el tipo de navegador. Los applets se ejecutan siempre en el lado cliente ya que son descargados desde un servidor web conjuntamente con la página HTML.

Los applets hacen referencia al archivo compilado .class, si tenemos una aplicación compilada llamada raizN.class entonces el código HTML sería de la siguiente manera:

```
<html>
<head>
    <title>Aplicacion de Prueba</title>
</head>
<body>
    Raiz N
    <APPLET>
    CODE="raizN.class"
    WIDTH=200
    HEIGHT=70
    </APPLET>
</body>
</html>
```

Aplicaciones Servlets

Son objetos de Java que se ejecutan dentro de un servidor web, como por ejemplo TomCat que los reutiliza, recibe peticiones mediante una URL y genera respuestas al Cliente. Genera páginas web dinámicas a partir de los valores que sean enviados al servidor como se muestra en la Fig.1.4.

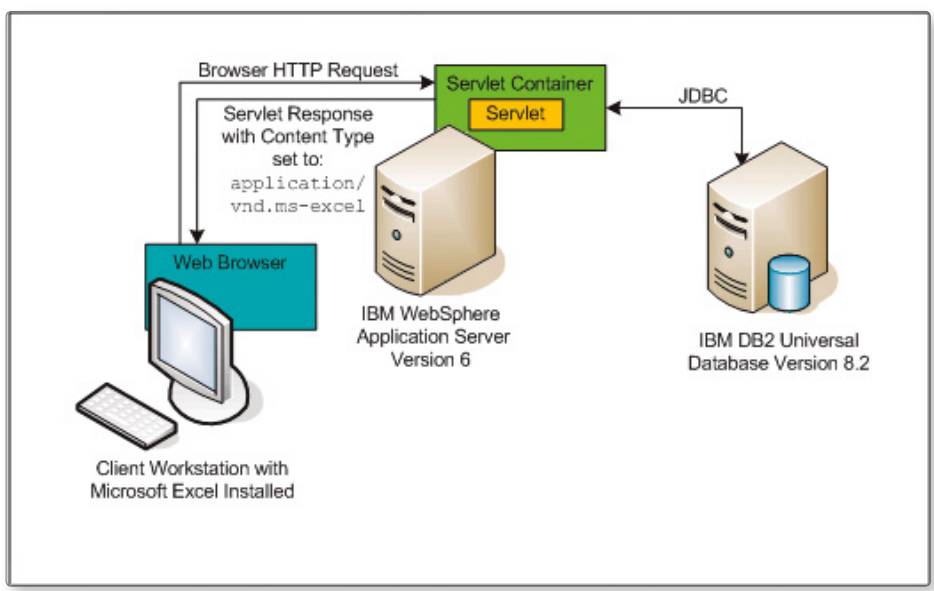


Fig. 1.4

Aplicaciones JavaBeans

Es un modelo de componentes que se usa para encapsular varios objetos en otro único. Se define como un componente de software reutilizable que se puede administrar desde una herramienta de construcción, un ejemplo de código JavaBean es el que sigue:

```
public class VendedorBean
  implements java.io.Serializable {

   private String vendedor;
   private double monto;

   public VendedorBean() {}

   public void setVendedor(String ven) {
     vendedor = ven;
   }

   public void setMonto(double mon) {
     monto = mon;
   }

   public String getVendedor() {
    return vendedor;
   }

   public int getEdad() {
    return monto;
   }
 }
```

1.6. Requisitos para la instalación de Java

Los requisitos mínimos para la instalación de Java son los siguientes:

Requerimiento	Especificación
Sistema Operativo	Windows 8, 7, Vista SP2, XP S3 Windows Server 2008, 2012
Memoria RAM	128MB 64MB
Espacio de Disco	124MB
Navegador Web	IE 7.0 o superior Firefox 12 o superior Chrome

1.6.1. Instalando Java

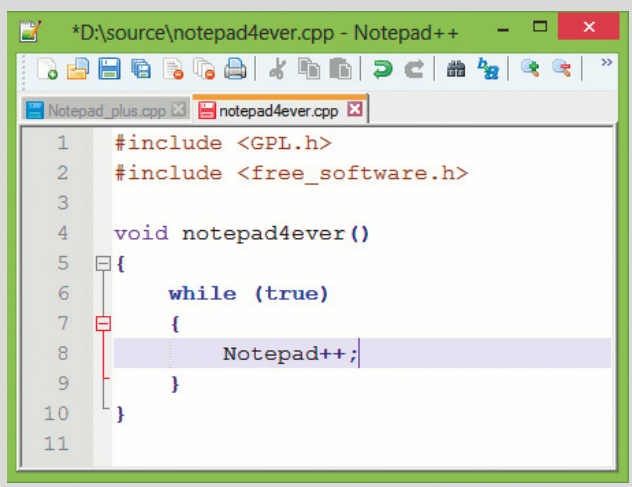
Java SE7 es la versión más actualizada de Java, la cual contiene muchas características nuevas dentro del lenguaje. Veamos algunas:

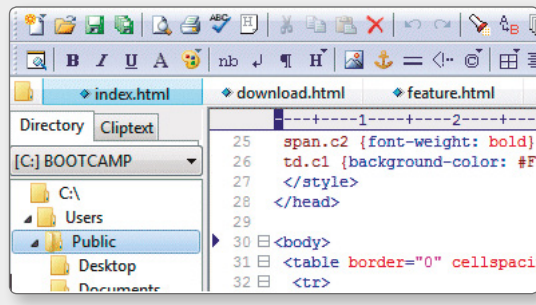
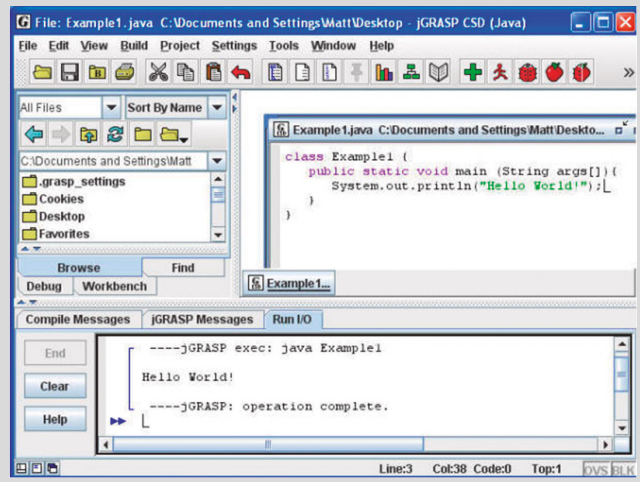
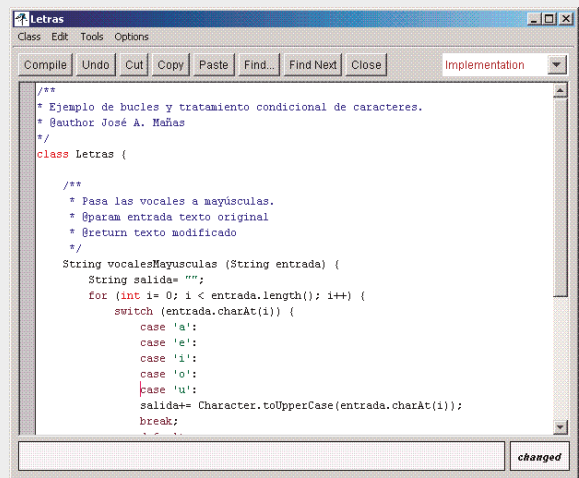
- Mejoras en el rendimiento, estabilidad y especialmente la seguridad.
- Mejoras en los Plugins Java para el desarrollo de aplicaciones para la web.
- Mejoras en la edición y optimización del código Java, que permiten un trabajo efectivo de los desarrolladores.
- Mejora en la JVM de Java ofreciendo soporte para diferentes lenguajes no necesariamente Java.

Debemos tener en cuenta que para ejecutar aplicaciones Java en una computadora personal se necesitan de las siguientes aplicaciones:

- Editor del Lenguaje
- Compilador Java
- Máquina Virtual de Java
- Librerías

Editor del Lenguaje: son aplicaciones que permiten editar un código Java, es decir, solo podrá digitar el código pero no podrá compilarlo ni ejecutarlo. Podríamos nombrar las siguientes aplicaciones:

Aplicación	Ubicación o descarga
Bloc de Notas NotePad	En todas las versiones de Windows
NotePad++	http://notepad-plus-plus.org/ 

APLICACIÓN	UBICACIÓN O DESCARGA
EditPlus	http://www.editplus.com/ 
JGrasp	http://www.eng.auburn.edu/grasp/ 
BlueJ	http://www.bluej.org/download/download.html 

Compilador Java

Como ya habíamos comentado líneas arriba, los compiladores generan un archivo .Class que es enviado a la memoria de la computadora en código de Bytes llamado ByteCode.

Máquina Virtual de Java (JVM)

La ejecución de las aplicaciones implementadas para Java se llevan a cabo gracias a la Máquina Virtual de Java garantizando así la portabilidad de las mismas, esta define una computadora lógica que ejecutará las instrucciones Bytecode y su intérprete ejecutará las instrucciones generadas dentro del archivo .class. Veamos algunas características de la JVM:

- Administra de manera correcta la memoria no usada gracias a su *Garbage Collector* o recolector de desechos.
- Reserva espacios en la memoria principal de la computadora para aquellos objetos que se crean en la aplicación, estos usan el operador new.
- Es la encargada de asignar variables a los registros y pilas.
- Administración de funciones del sistema huésped para el acceso a los dispositivos de la computadora.
- Administra la seguridad en las aplicaciones Java.
- No gestiona punteros, en su lugar crea objetos de clases.
- Administra de manera correcta los Casting entre los diferentes tipos de datos.

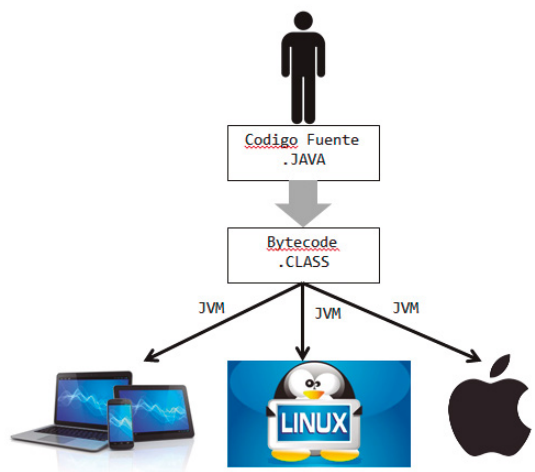


Fig. 1.5

Como se muestra en la Fig. 1.5 el código es compilado y enviado a los diferentes entornos por medio de la JVM.

Librerías

Es un apoyo al desarrollo de las aplicaciones en Java ofreciendo métodos agrupados que permiten agilizar el desarrollo y relacionarse con sistemas externos como las API's de acceso a datos o las bibliotecas de Interfaz de Usuario como la AWT y la Swing y API's para la captura, procesamiento y reproducción de audio en Java.

Ahora se estará preguntando, ¿cuánto software descargaré para poder tener el Java en mi computadora? Pues la respuesta se puede resumir a solo dos aplicaciones:

APLICACIÓN	CONTENIDO
JDK 7 (Java Development Kit)	• Compilador Java • Máquina Virtual de Java • Librerías
IDE (Entorno de Desarrollo Integrado)	• Netbeans • JCreator • JDeveloper • Eclipse

1.7. EL JDK 7 Y SU INSTALACIÓN

El JDK es un Kit que contiene el compilador de Java, la JVM y las librerías necesarias para generar una aplicación Java. Para descargar el JDK 7 ingrese a la siguiente URL:

http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

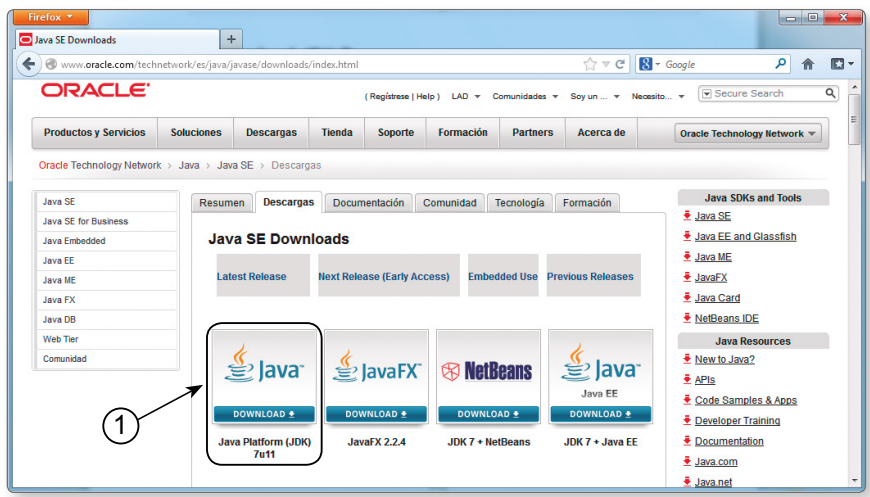


Fig. 1.6

En la Fig. 1.6 se muestra la sección de descargas de Java desde la página oficial de Oracle; como notará Java ya no pertenece a la compañia Sun. Para descargar el JDK seleccionaremos Download del Java Platform (JDK) como lo muestra la Fig.1.7

Fig. 1.7

Seguidamente debe seleccionar:

- Accept License Agreement
- Windows x86 – 88.77 MB – jdk-7u11-windows-i586.exe esto se debe a que la versión de Windows que se utilizó para este material es Windows 7 de 32 Bits. Como lo muestra la Fig. 1.8.

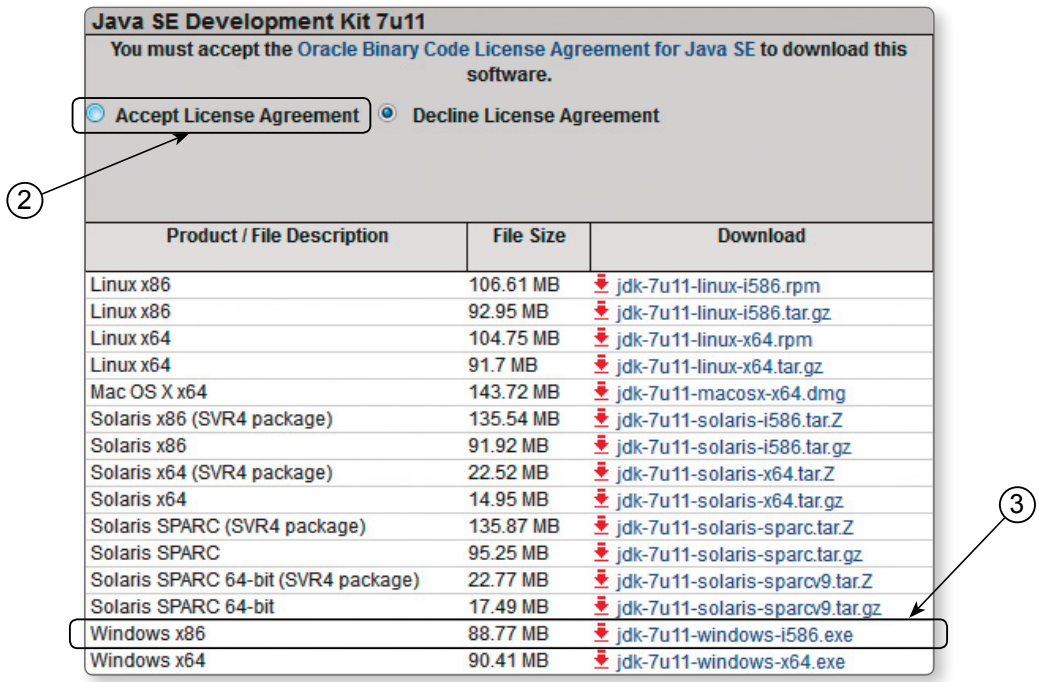


Fig. 1.8

Luego debe guardar el archivo en su Unidad de Disco estable para poder instalarlo desde allí como lo muestra la Fig. 1.9:

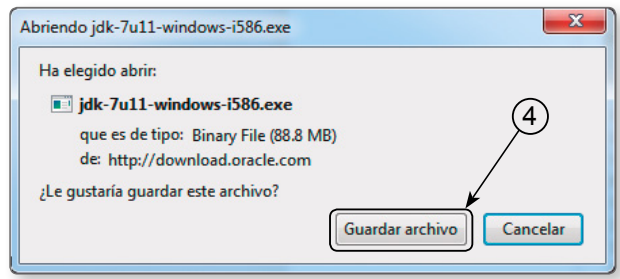


Fig. 1.9

Una vez finalizada la descarga del archivo JDK se procederá con la instalación, en la Fig. 1.10 presione Next>.

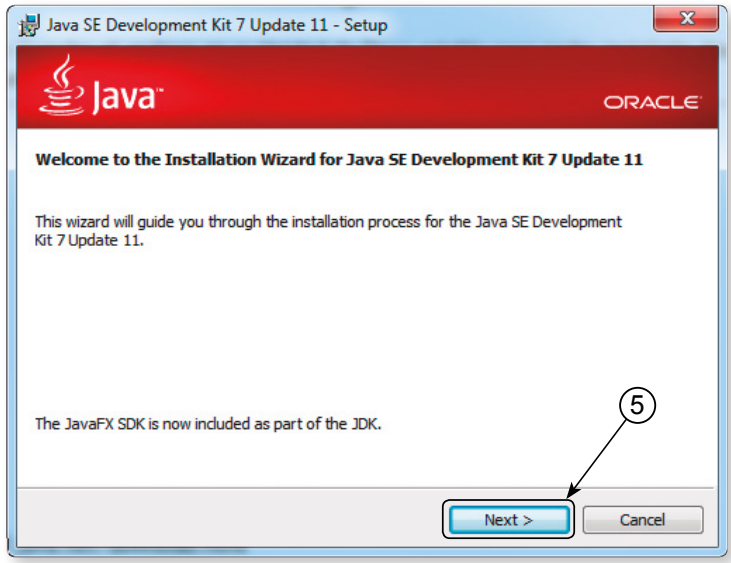


Fig. 1.10

En la Fig. 1.11 se muestra los objetos que se instalarán, si lo desea aquí puede cambiar la ubicación de la instalación presionando el botón Change..., luego presione Next>.

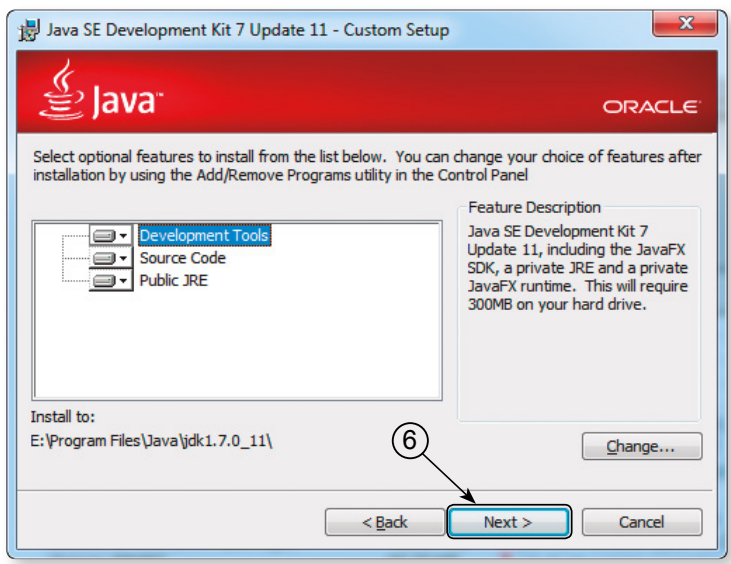


Fig. 1.11

En la Fig. 1.12 se muestra el avance de la instalación, en esta ventana no se selecciona ningún elemento.

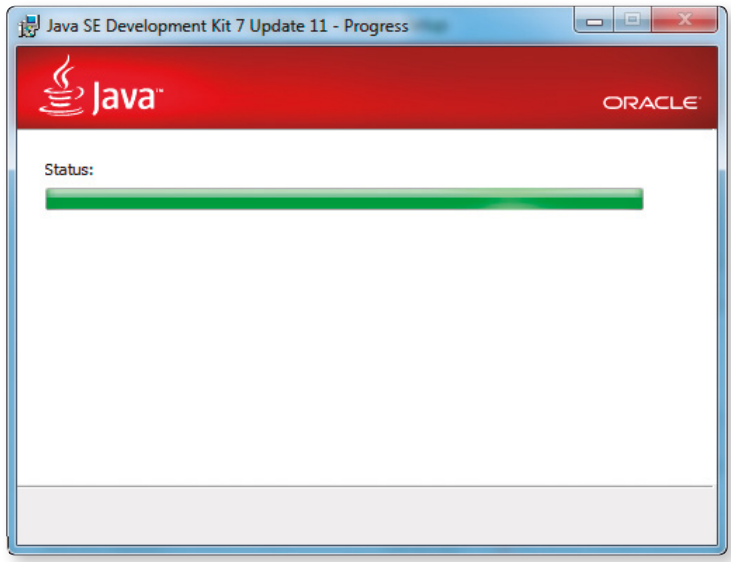


Fig. 1.12

En la Fig. 1.13 se muestra la ventana de selección de destino de los archivos JRE7 para lo cual no es necesario cambiarlos, pero si lo desea puede seleccionar Change, pero recuerde que no es necesario cambiarlos.

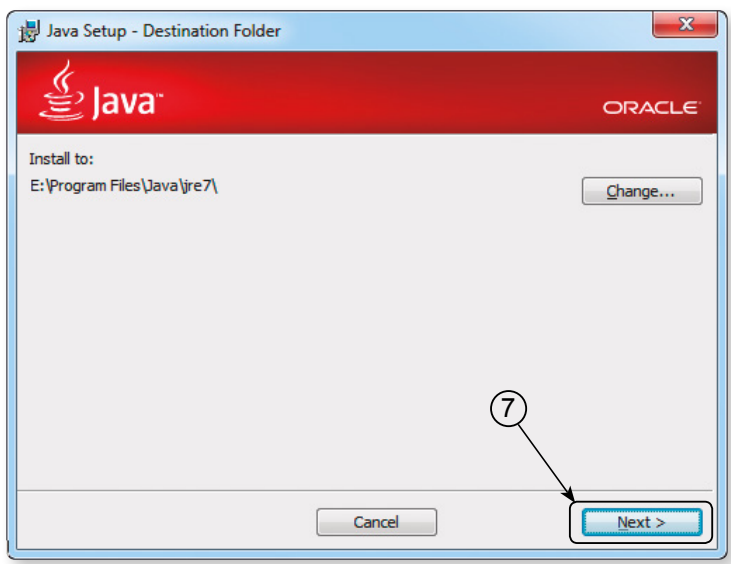


Fig. 1.13

En la Fig. 1.14 se muestra el proceso de instalación:

Fig. 1.14

Finalmente, si todo es correcto se debe mostrar la ventana que muestra la Fig. 1.15. Para finalizar la instalación presione Close.

Fig. 1.15

También se puede descargar el JDK de la siguiente URL:
http://jdk7.java.net/download.html

Tal como lo muestra la Fig. 1.16 primero debe seleccionar Accept License Agreement, desde la plataforma Windows seleccione exe 88.94MB y proceda con la instalación.

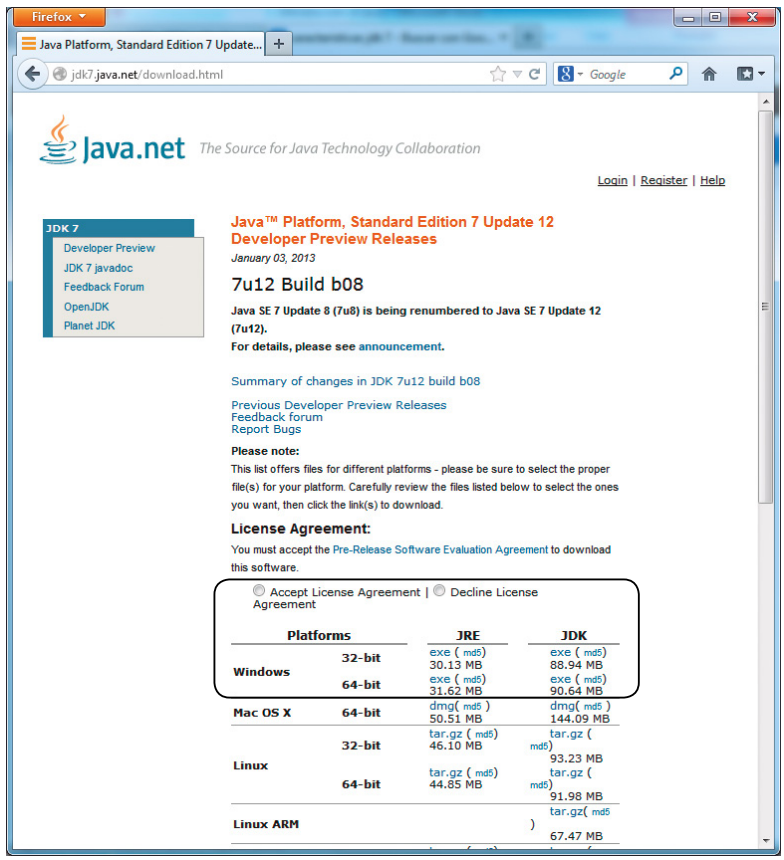


Fig. 1.16

1.8. El IDE JCreator y su instalación

El JCreator es una aplicación que permite crear aplicaciones con Java bajo el entorno Windows. La compañía que lo produce es Xinox Software, la cual ofrece dos versiones del producto: la gratuita llamada JCreator LE y la licenciada llamada JCreator Pro; esta última incorpora comandos, plantillas y otros complementos que la versión libre no los tiene, esto no implica que no pueda desarrollar aplicaciones Java.

Para descargar las versiones de JCreator visite la siguiente URL:
http://www.jcreator.org/download.htm

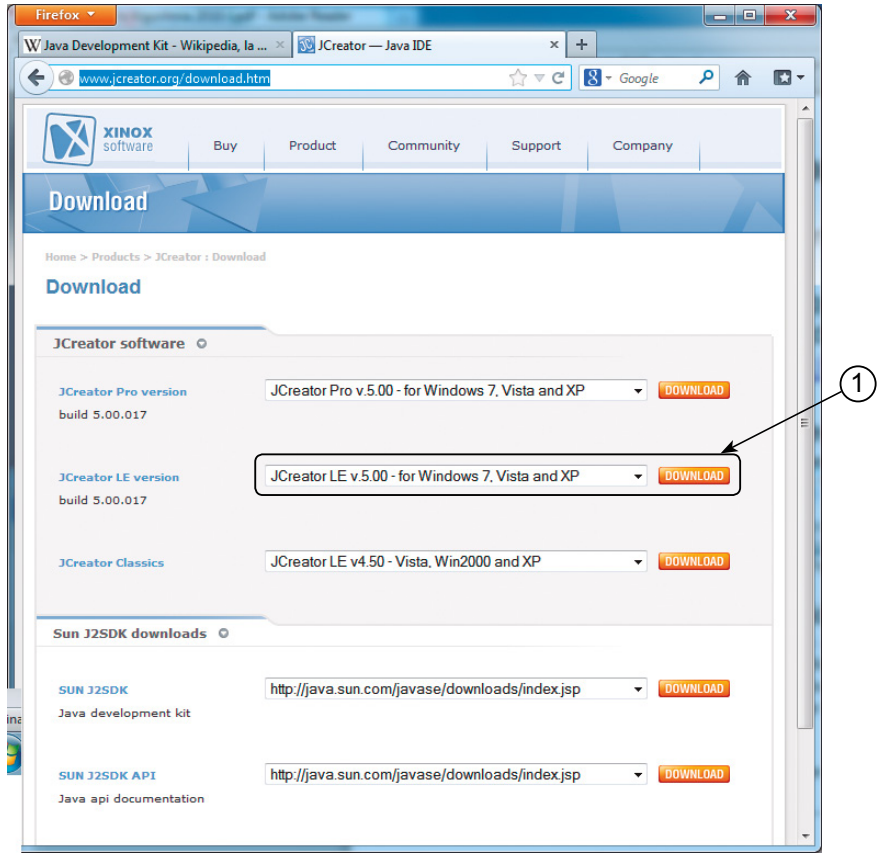


Fig. 1.17

Una vez descargado procederemos a instalarlo, presionando Next> desde la Fig. 1.18.

Fig. 1.18

Luego aceptamos en contrato como lo muestra la Fig. 1.19

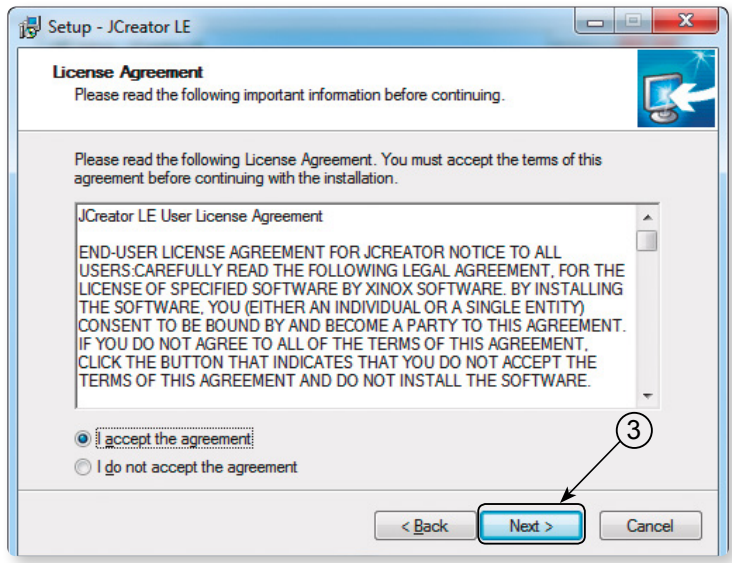


Fig. 1.19

Seguidamente presionamos Next, a menos que desee cambiar la ubicación de los archivos de instalación para esto presione Browse…, finalmente presione Next>.

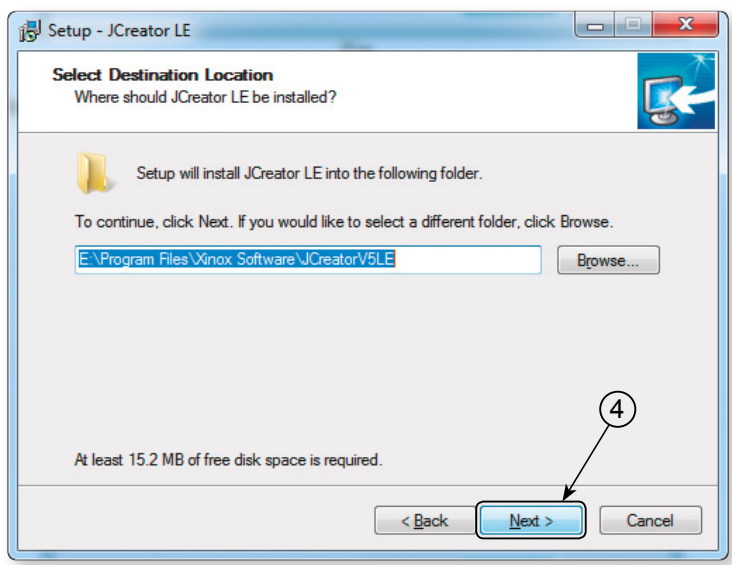


Fig. 1.20

Siga los pasos del asistente de instalación hasta llegar a la ventana mostrada en la Fig. 1.21, presione Finish para empezar a usar JCreator.

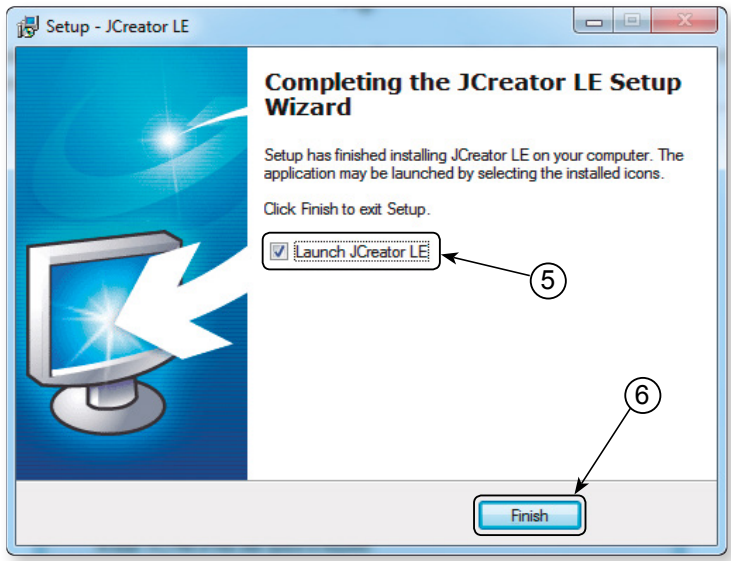


Fig. 1.21

Por ser la primera vez que se instala este software se muestra la siguiente ventana:

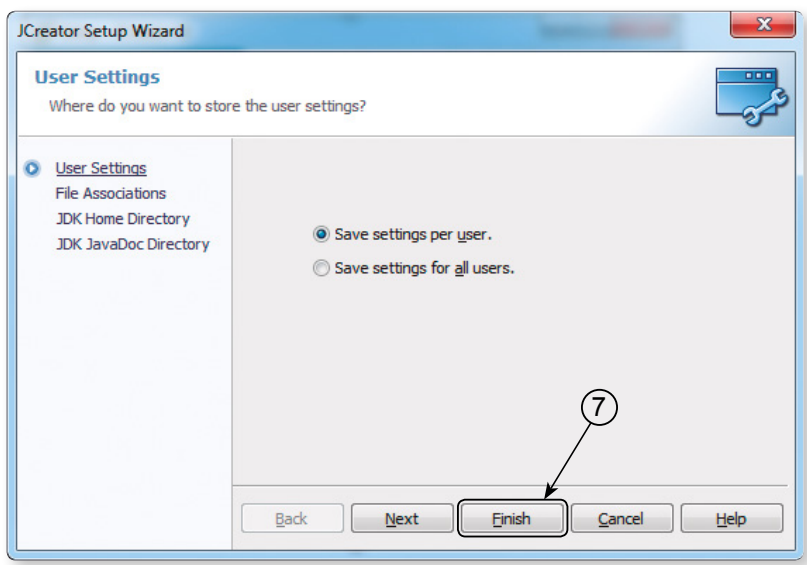


Fig. 1.22

Para lo cual solo presione Finish y se mostrará la pantalla inicial del JCreator como lo muestra la Fig. 1.23.

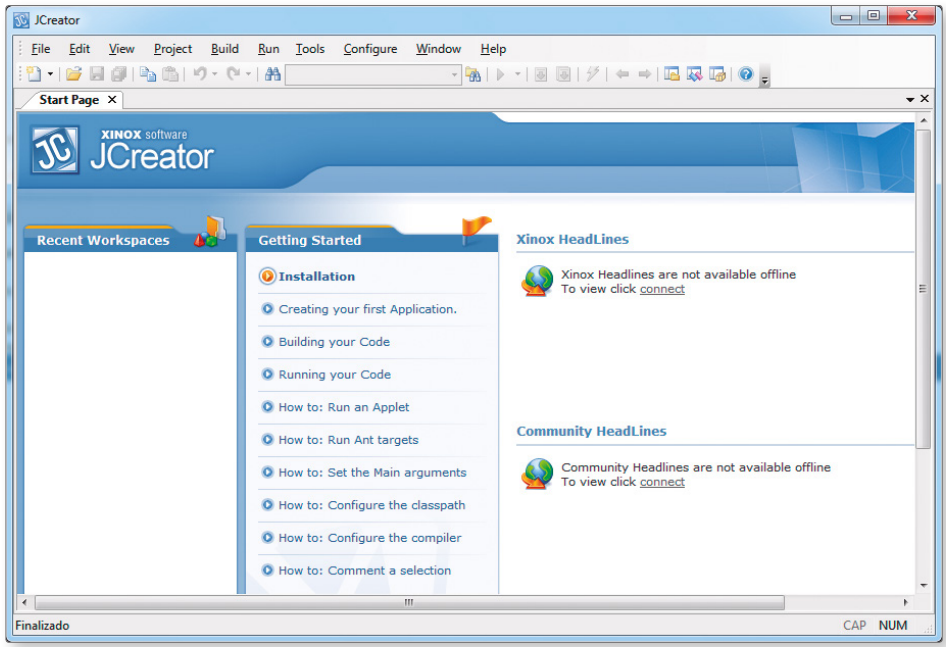


Fig. 1.23

1.9. El IDE NetBeans y su instalación

NetBeans IDE es una aplicación de código abierto diseñada para el desarrollo de aplicaciones Java entre las distintas plataformas.
Con NetBeans se puede:

- Crear interfaces gráficas de forma visual.
- Desarrollar aplicaciones web.
- Crear aplicaciones compatibles con teléfonos móviles.

Para descargar la version 7.2 de NetBeans visite la siguiente URL:
http://netbeans.org/

Fig. 1.23

Se descargará la versión más actualizada hasta el momento de la edición de este material, seleccione el idioma del IDE (en este caso es English), la plataforma de trabajo Windows y finalmente presione Donwload Free 240MB para tener toda la tecnología de NetBeans completa (ver fig. 1.24).

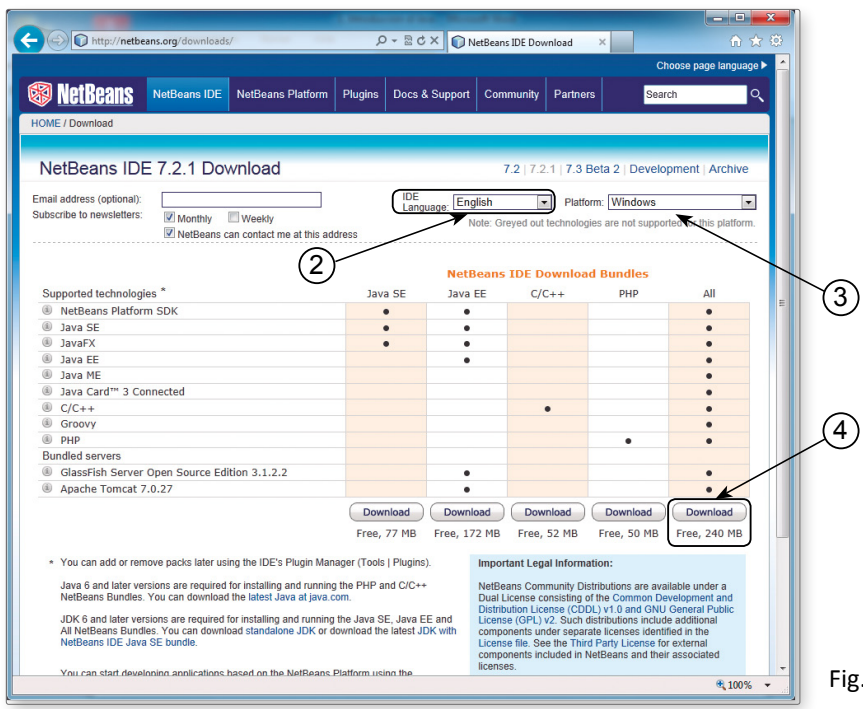


Fig. 1.24

Una vez descargado el IDE NetBeans se procederá a instalarlo de la siguiente manera:

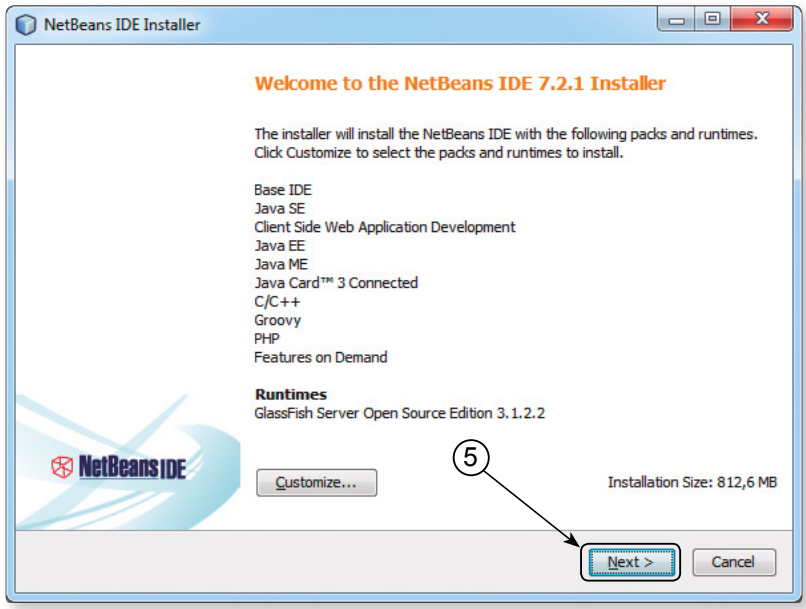


Fig. 1.25

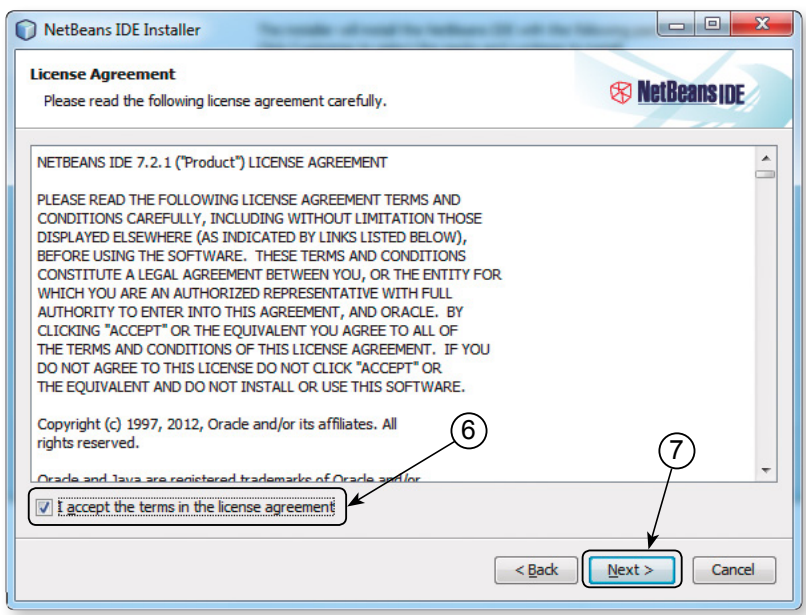


Fig. 1.26

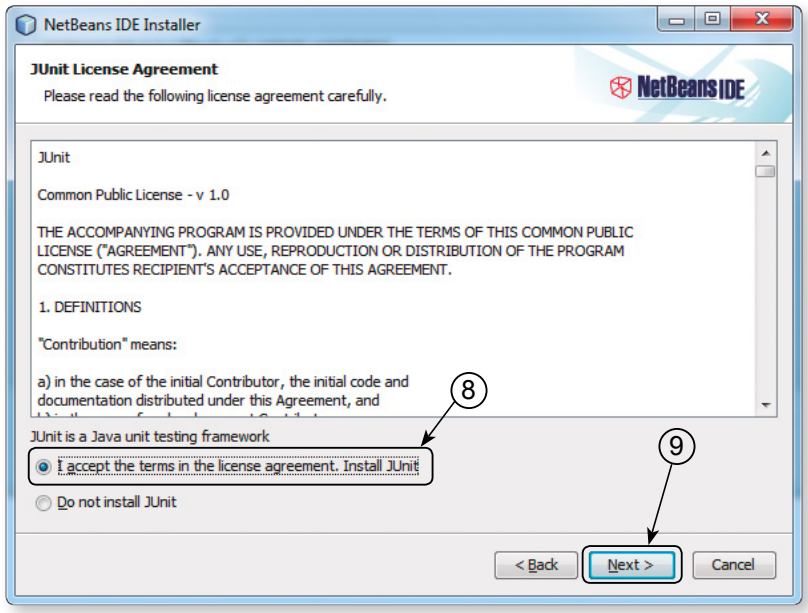


Fig. 1.27

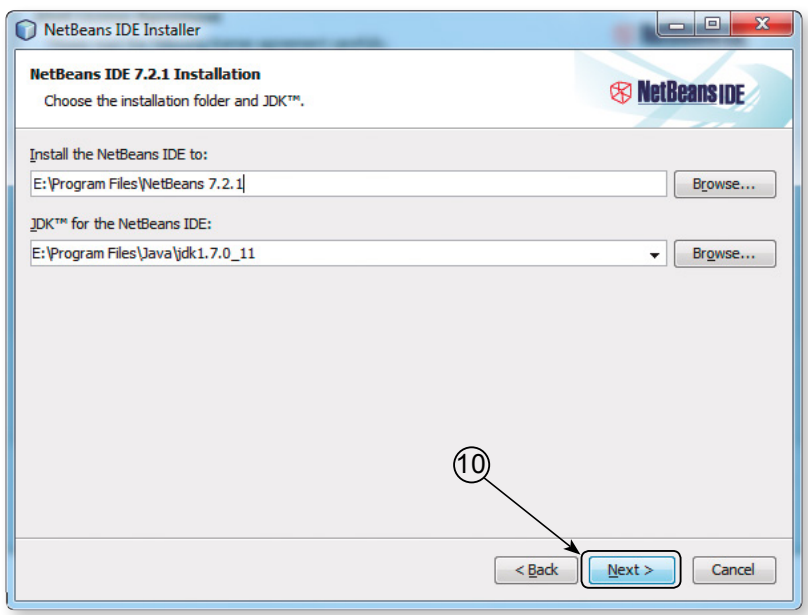


Fig. 1.28

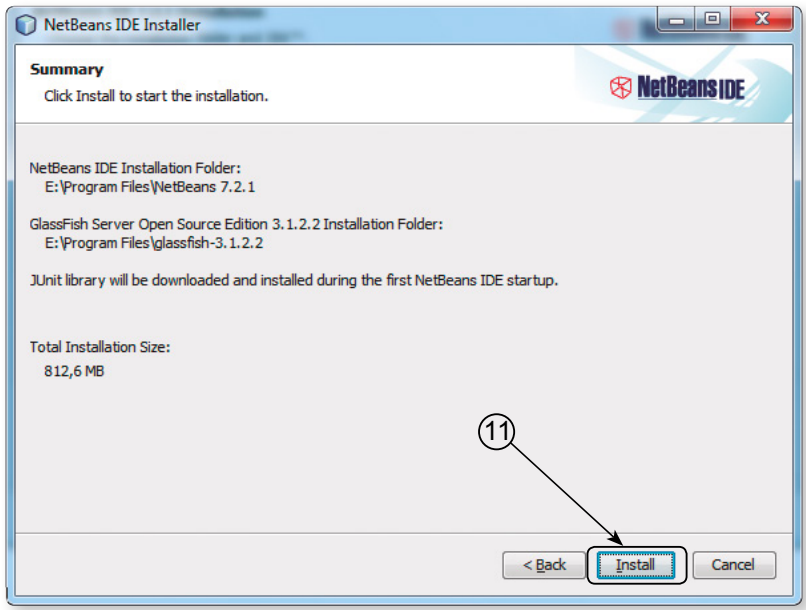


Fig. 1.29

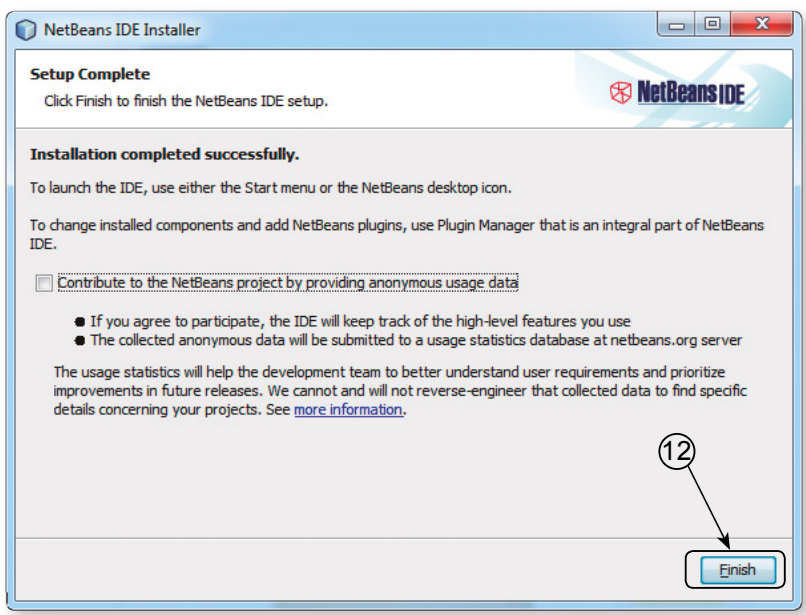


Fig. 1.30

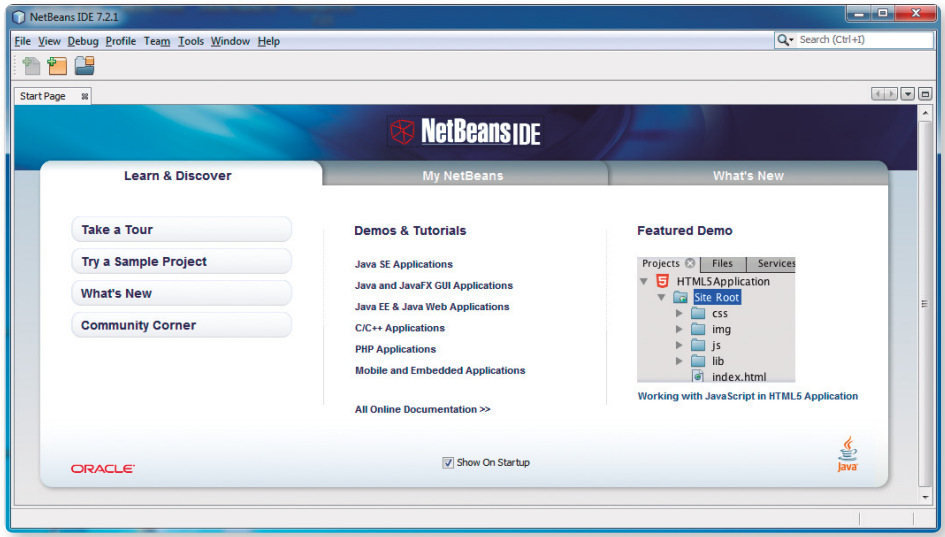


Fig. 1.31

1.10. EL IDE JDEVELOPER 11G Y SU INSTALACIÓN

JDeveloper es un entorno de desarrollo integrado, desarrollado por la compañía Oracle Corporation para los lenguajes Java, HTML, XML, SQL, PL/SQL, Javascript, PHP, Oracle ADF, UML y otros. Es un software propietario pero gratuito desde el 2005.
Con JDeveloper se puede realizar lo siguiente:

- Crear interfaces gráficas de forma visual.
- Desarrollar aplicaciones web.
- Crear aplicaciones compatibles con teléfonos móviles.

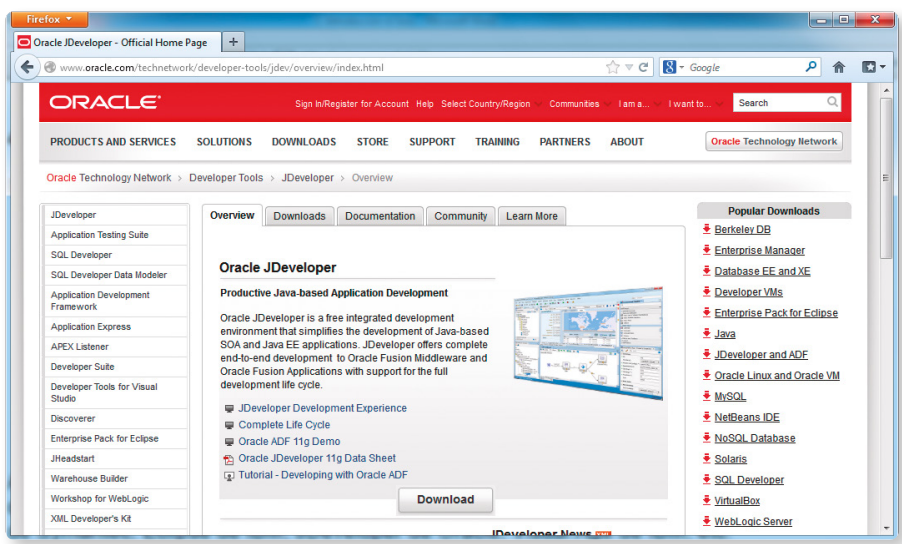


Fig. 1.32

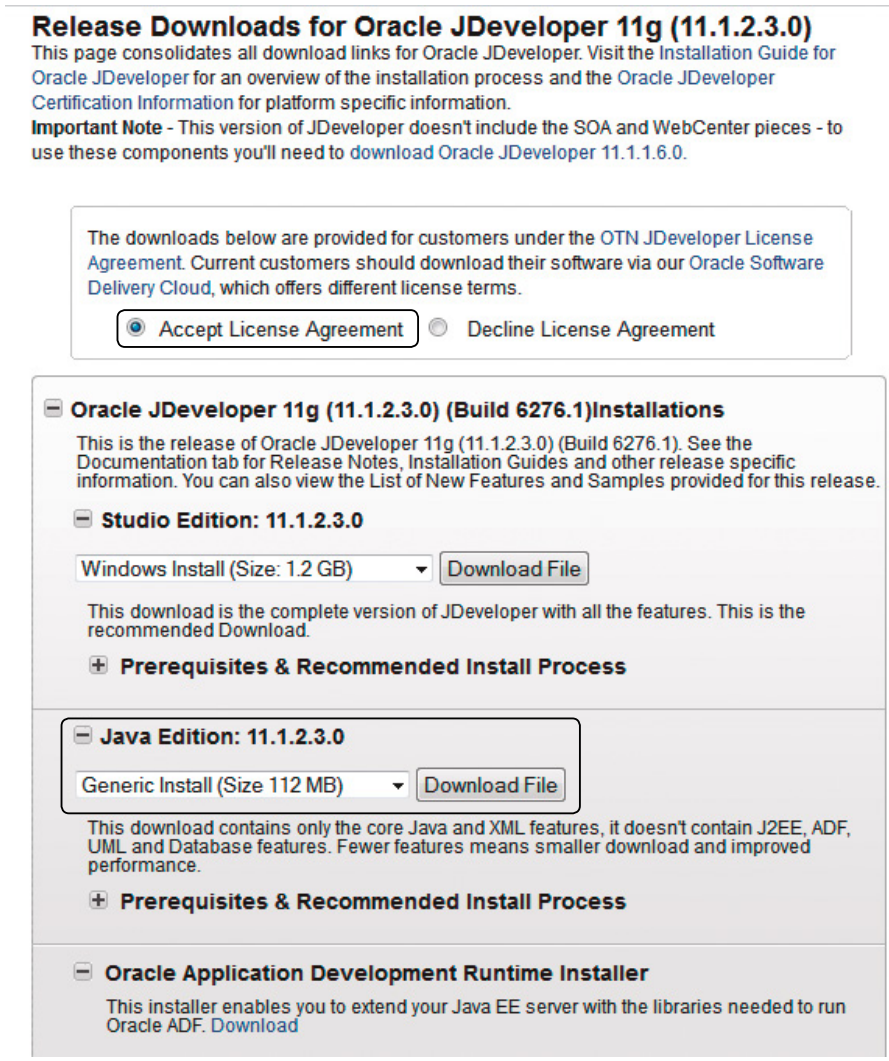


Fig. 1.33

Fig. 1.34

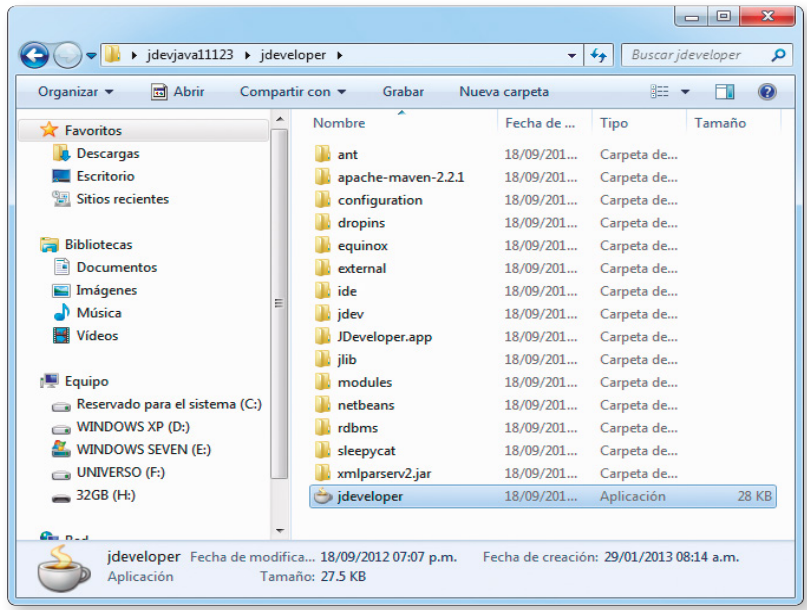


Fig. 1.35

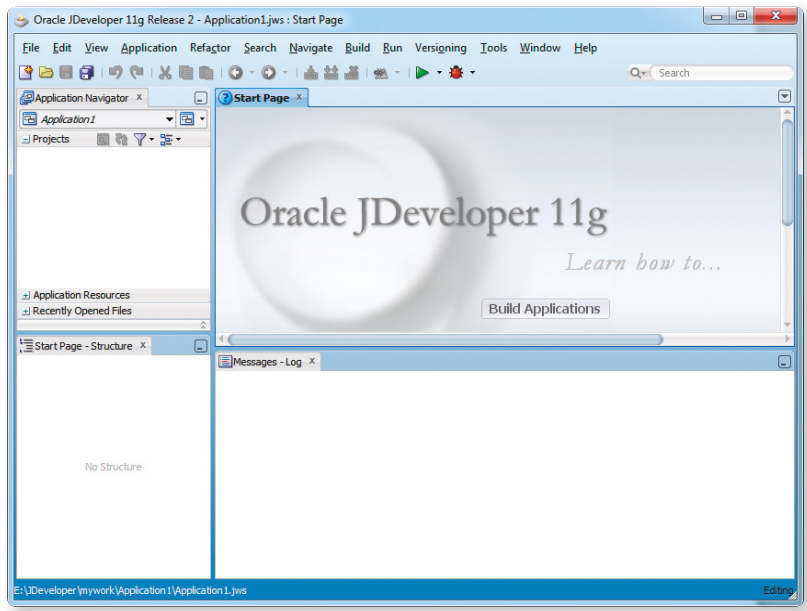


Fig. 1.36

CAP.

2

Fundamentos de programación

CAPACIDAD:

- Reconocer conceptos básicos de programación estructurada.
- Reconocer los elementos que componen una aplicación Java.
- Implementar proyectos en diferentes IDE como JCreator, NetBeans y JDeveloper.

CONTENIDO:

2.1. INTRODUCCIÓN

Si sabemos que los programas de computadora realizan tareas con un fin en común, también se debe tener en cuenta que dichos programas no deben tener errores y ser lo más entendible posible para el usuario final. Si tomamos en cuenta que un problema es un asunto que requiere de una solución adecuada, a nivel informático, se trata de alguna situación en concreto que, en el momento en que se logra solucionar, aporte beneficios a una determina organización haciendo más productiva las labores y por consecuencia estar a la vanguardia de la sociedad informática.
Para la resolución de un problema debemos planear una estrategia adecuada, mínimamente se puede proponer:

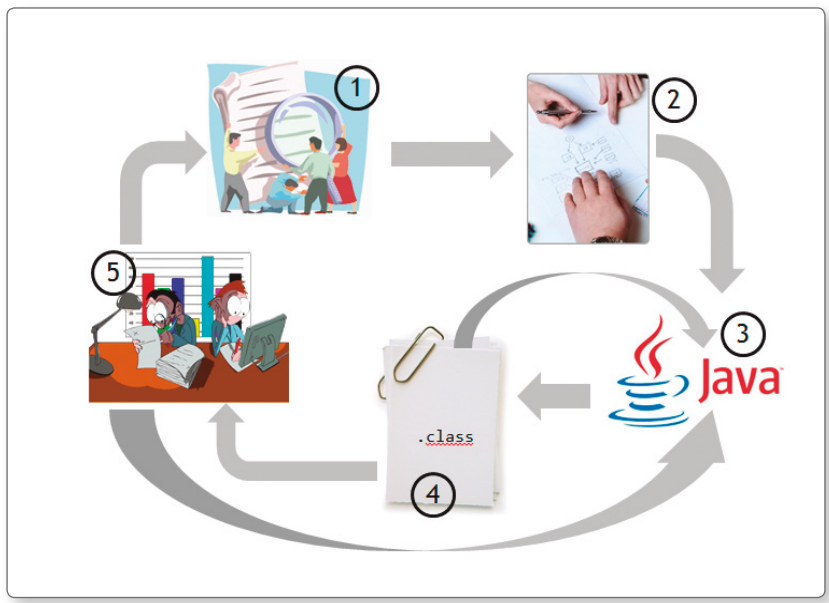
1.- Análisis del problema, es decir, entender el problema desde ¿Cuál es su objetivo? y ¿Qué necesita para lograrlo?

2.- Diseño de un algoritmo

3.- Convertir el algoritmo en una aplicación Java

4.- Compilar la aplicación

5.- Validar los datos



En el punto uno se analiza el problema, luego se diseña el algoritmo de solución como se muestra en el punto dos. Luego se convierte el algoritmo en código Java, es decir, una aplicación como se muestra en el punto tres. Seguidamente se compila y se crean los archivos .class de Java como se ve en el punto cuatro, pero si se encuentran errores en el tiempo de compilación este lo regresará al punto tres, si todo es correcto es enviado a la validación de los datos resultantes, es decir, si estos datos son los esperados por el usuario como lo muestra el punto cinco; en caso se encontrase errores desde aquí se enviará nuevamente al punto tres para que sea revisado y modificado si fuera necesario y en caso extremo se encontrase errores de tratamiento de datos será enviado nuevamente al punto uno y empezar el ciclo nuevamente.

También hay que recalcar que lo mencionado anteriormente pertenece solo a la parte de la implementación de una aplicación, ya que si estuviéramos hablando de un software se deberían seguir las siguientes fases:

1.- Análisis de requisitos

2.- Especificación de requisitos (casos de uso)

3.- Diseño de la Arquitectura

4.- Programación

5.- Prueba

6.- Documentación

7.- Mantenimiento

Con estas fases expuestas debemos tener claro que la programación de una aplicación es una fase en la construcción de un software, este libro apunta justamente en ese sentido: solo nos dedicaremos a la implementación de código en el desarrollo de aplicaciones Java.

2.2. Los algoritmos

Del griego y latín .dixit algorithmus" y este a su vez del matemático persa Al-Juarismi, es un conjunto de instrucciones o reglas bien definidas, ordenadas y finitas que permiten realizar un proceso mediante pasos sencillos de entender, es decir, expresado en un lenguaje simple.

Las características de un algoritmo son:

- Tiene que ser preciso
- Debe ser secuencialmente ordenado
- Debe estar bien definido
- Debe tener un fin

Las partes de un algoritmo son:

- Entrada: responde a la pregunta ¿Qué datos necesito para la solución del problema?
- Proceso: responde a la pregunta ¿Cómo lo hago o cuál es la fórmula para dar solución al problema?
- Salida: responde a la pregunta ¿Cuál es el resultado o cuál es el objetivo del problema?

2.3. ¿CÓMO SOLUCIONAR UN PROBLEMA?

Para esto proponemos un problema y comenzaremos las fases de la solución del mismo, para este caso usaremos JCreator.

Una empresa desea calcular el pago mensual realizado a sus empleados, el cual cuenta con las siguientes características:

Sueldo básico: se calcula en base al número total de horas trabajadas basado en una tarifa horaria.
Bonificación: se calcula obteniéndose el 20% del sueldo básico.
Sueldo neto: se obtiene de descontar el 10% de la diferencia entre el sueldo básico y su bonificación.

Entonces, se requiere implementar una aplicación Java que permita calcular el sueldo básico, la bonificación y el sueldo neto de un empleado.

FASE 1: ANALIZAR EL PROBLEMA

- ¿Qué datos necesito para la solución del problema?
  - Conocer el nombre del empleado, las horas de trabajo y la tarifa asignada por hora de trabajo.
- ¿Cuál es el resultado final del problema?
  - Imprimir el sueldo básico.
  - Imprimir el monto de bonificación.
  - Imprimir el sueldo neto.

FASE 2: DISEÑO DEL ALGORITMO

Para diseñar un algoritmo de solución adecuada se puede usar herramientas:

- Pseudocódigo: el cual presenta un lenguaje informal del problema que permite razonar al desarrollar ya que se concentra solo en el programa y no en la parte gráfica del mismo.
- Diagrama de flujo de datos: representa en forma grafica lo que se desarrolla en un pseudocódigo.

Entonces, usaremos para este caso el pseudocódigo como herramienta principal para el desarrollo de nuestras aplicaciones a lo largo de estos capítulos. Veamos la solución:

```
Inicio
     //Entrada
     Leer Empleado, horas, tarifa

//Proceso
     Basico ← horas * tarifa
     Bonificacion ← basico * 0.2
     Neto ← ( basico + bonificacion ) * 0.9

//Salida
     Imprimir Empleado, Basico, Bonificacion, Neto
Fin
```

Fase 3: Convertir el algoritmo en una aplicación Java

Al realizar esta transición se debe tener en cuenta que Java es un lenguaje de programación y, por lo tanto; propone reglas y sintaxis que debemos respetar y que a lo largo de este libro las vamos a exponer.

```
//Entrada
    int horas=Integer.parseInt(txtHoras.getText());
    double tarifa=Double.parseDouble(txtTarifa.getText());

    //Proceso
    double basico = horas * tarifa;
    double bonificacion = basico * 0.2;
    double neto = (basico + bonificacion) * 0.9;

    //Salida
    txtSalida.setText(" ** Resumen de Pago a Empleado ** ");
    txtSalida.append("El básico es: "+basico);
    txtSalida.append("La bonificación es: "+bonificacion);
    txtSalida.append("El neto es: "+neto);
```

Fase 4 y 5: Compilar y validar los datos

Comprobemos que la aplicación resultante es la esperada por el usuario final. Como lo muestra la Fig. 2.1.

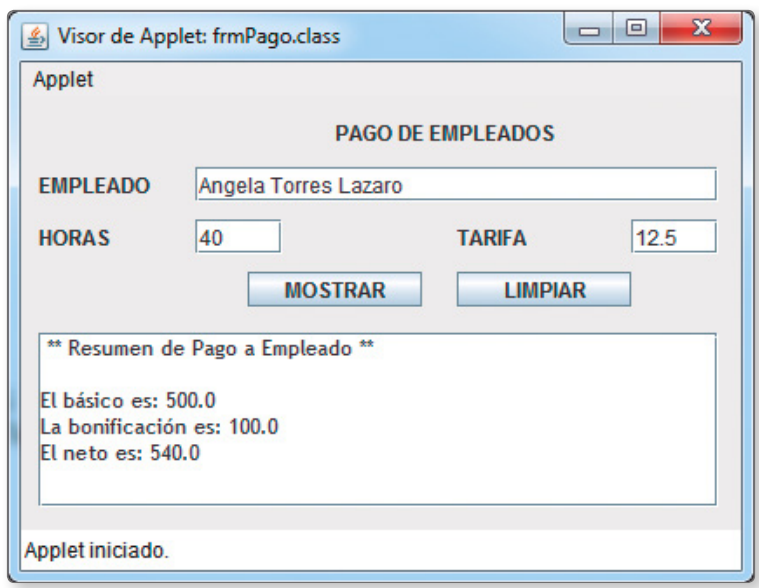


Fig. 2.1

2.4. ELEMENTOS QUE COMPONEN UNA APLICACIÓN JAVA

Una aplicación Java se compone de muchos elementos que el compilador tiene que reconocer y principalmente diferenciarlos; por lo tanto, usted como desarrollador debe conocer cuáles son dichos elementos ya que el compilador traducirá a código de bytes sin especificar espacios en blanco ni comentarios realizados dentro de la aplicación.

Veamos el código de una aplicación básica:

```
public class raizN{
   public static void main(String[] args){
     int n = 5;
     int raiz = 2;
     double respuesta = Math.pow(n,(1.0/raiz));
     System.out.println("La raiz es: "+respuesta);
   }
}
```

Como notará el código mostrado tiene varios elementos que al final componen una aplicación Java, en los siguientes temas explicaremos dichos elementos.

2.5. LOS IDENTIFICADORES

Es el nombre que se le asigna a cualquier elemento de una aplicación Java, estos identifican un elemento para el compilador y dan un sentido lógico a lo desarrollado, puede ser una clase, método, variables, etc. Hay ciertas reglas que debemos tener en cuenta al asignar los identificadores:

- El identificador es único dentro de una aplicación.
- El identificador tiene que darle sentido al valor que reservará.
- Hay diferencias entre mayúsculas y minúsculas de un identificador.
- Siempre debe comenzar por una letra, un subrayado bajo o el símbolo de dólar ($), los demás caracteres del identificador pueden ser la combinación de letras y números.
- Un identificador no puede ser una palabra reservada por el lenguaje Java.

En el caso siguiente listaremos los identificadores válidos y no válidos:

Una empresa desea calcular el pago mensual realizado a sus empleados, el cual cuenta con las siguientes características:

Sueldo básico: se calcula en base al número total de horas trabajadas basado en una tarifa horaria.
Bonificación: se calcula obteniéndose el 20% del sueldo básico.
Sueldo neto: se obtiene de descontar el 10% de la diferencia entre el sueldo básico y su bonificación.

Entonces, se requiere implementar una aplicación Java que permita calcular el sueldo básico, la bonificación y el sueldo neto de un empleado.

Válidos	No Válidos
empleado nombreEmpleado nombre_Empleado strEmpleado	1Empleado Nombre Empleado
horas horasTrabajadas horas_Trabajadas intHoras	@Horas@ Horas Trabajadas 40Horas
tarifa tarifaHora tarifa_Hora douTarifa	10Tarifa Tarifa Hora Double
$sueldoBasico sBasico sueldo_Basico _sueldoBasico douSueldoBasico	$ sueldoBasico Sueldo Basico 1000.20
_bonificacion __bonificacion $bonificacion $_bonificacion	_ bonificacion $ bonificacion 10%Bonificacion $ 90.00
neto	monto neto

2.6. Palabras reservadas por el Lenguaje Java

Es una palabra que tiene un significado gramatical especial para el compilador del lenguaje Java y no puede ser utilizada como un identificador por ningún motivo ya que generaría un error.

Veamos la lista de palabras reservas con que cuenta Java:

Palabra Reservada	Definición
abstract	Permite implementar clases abstractas que no podrán ser asociadas a ningun objeto.
boolean	Tipo de datos primitivo de tipo lógico (True/False)
break	Permite un salto fuera de la instruccion que lo implementa y continua con la siguiente posterior a la instruccion.
byte	Tipo de datos primitivo de tipo entero con capacidad 1 Byte.
case	Es la clausula de la instrucción switch que permite especificar los posibles valores que ingresan a la instrucción.
catch	Es el codigo que se ejecuta cuando ocurre una excepcion controlada con Try.
char	Tipo de datos primitivo de tipo carácter.
class	Permite la implementación de clases dentro de la aplicación Java.
const	Sin uso en Java.

Palabra Reservada	Definición
continue	Permite un salto dentro de la instruccion que lo implementa y continua con la misma instrucción hasta finalizar la instrucción.
default	Es una clausula de la estructura switch que determina que instrucción se deben realizar en una condicion por defecto, es decir cuando se cumpla con los valores especificadas.
do	Es una estructura que permite generar ciclos repetidos mientras una condicion dada sea verdadera.
double	Tipo de datos primitivo de tipo real con capacidad de 8 bytes.
else	Es una clausula del lado contrario a una condicion de la instrucción If.
extends	Asigna a un clase ser derivada o heredada de otra.
final	Es un atributo que se le asigna a una variable o a un metodo, para indicar que ya no puede ser modificado una vez declarado.
finally	Es el bloque de codigo que se ejecutara siempre así se genere una excepcion o no; controlada con try.
float	Tipo de datos primitivo de tipo real con capacidad de 4 bytes.
for	Es una estructura que permite generar ciclos repetidos de manera limitada.
goto	Permite direccionar a una línea especifica del codigo Java.
if	Implementa una estructura condicional en Java.
implements	Permite implementar clases abstractas que pueden ser sobrescritas.
import	Sirve para referenciar a un paquete de Java.
instanceof	Sirve para consultar si un objeto es instancia de una clase determinada o de su clase padre.
int	Tipo de datos primitivo de tipo entero con capacidad 4 Bytes.
interface	Implementa una clase abstracta en la cual sus metodos no tienen implementación.
long	Tipo de datos primitivo de tipo entero con capacidad 8 Bytes.
native	Indica que el método está escrito en un lenguaje dependiente de la plataforma, habitualmente en C. Cuando usemos un método nativo, en nuestro fichero .java sólo incluiremos la cabecera del método con el modificador native y terminado en punto y coma (;) (como si fuera un método abstracto).
new	Es el operador que permite crear un objeto nuevo de una clase determinada.
package	Permite agrupar clases su trabajo es similar a las carpetas de windows simplemente organización.
private	Determina que un elemento Java sea accesible a su valor solo dentro de la clase donde se implementó.
protected	Determina que un elemento Java sea accesible a su valor dentro de la clase que lo implementó y las clases derivadas de el.
public	Determina que un elemento Java sea accesible a su valor tanto interna como externamente.
return	Permite devolver un valor desde un método con valor de retorno.
short	Tipo de datos primitivo de tipo enteros con capacidad 2 Bytes.
static	Permite compartir el valor de una variable miembro entre objetos de una misma clase así como lo hace una variable global.

Palabra Reservada	Definición
strictfp	Cuando colocamos el modificador strictfp delante de una clase o delante de un método, estamos indicando que nuestra clase o método se rige por el estandar IEEE 754 para manejar números en coma flotante y para realizar operaciones de coma flotante. De esta forma se puede suponer cómo van a comportarse las operaciones en punto flotante sin importar la arquitectura de la JVM que se encuentre por debajo.
super	Permite llamar a un método desde la clase padre o superclase.
switch	Es una estructura condicional multiple o también llamado multivalor.
synchronized	Se usa para indicar que ciertas partes del código Java están sincronizadas, es decir, que solamente un subproceso puede acceder a dicho método a la vez.
this	Permite acceder a las variables de instancia de una clase, este hace referencia a los miembros de la propia clase.
throw	Se usa para disparar una excepcion manualmente.
throws	Se utiliza para indicarle al compilador que una función puede disparar una o más excepciones.
transient	Utilizado para indicar que los atributos de un objeto no son parte persistente del objeto o bien que estos no deben guardarse y restaurarse utilizando el mecanismo de serialización estándar.
try	Es el bloque de codigo que el desarrollador prevee que se genera una excepcion y este sea controlado.
void	Permite implementar metodos que no tienen un valor de retorno.
volatile	Se utiliza este modificador sobre los atributos de los objetos para indicar al compilador que es posible que dicho atributo vaya a ser modificado por varios threads de forma simultanea y asíncrona, y que no queremos guardar una copia local del valor para cada thread a modo de caché, sino que queremos que los valores de todos los threads estén sincronizados en todo momento, asegurando así la visibilidad del valor actualizado a costa de un pequeño impacto en el rendimiento.
while	Es una estructura que permite generar ciclos repetidos mientras una condicion dada sea verdadera.

2.7. LOS OPERADORES

También son conocidos como .operandos" dentro de una expresión, a través de ellos podemos implementar expresiones combinando identificadores, métodos, etc. Tenemos:

- Operadores aritméticos binarios
- Operadores aritméticos unarios
- Operadores de incremento y decremento
- Operadores de asignación
- Operadores de comparación
- Operadores lógicos
- Operador de concatenación

2.7.1. Operadores Aritméticos Binarios

OPERADOR	ACCIÓN	EJEMPLO

+	Suma	`int n1 = 10;`
		`int n2 = 15;`
		`int suma = n1 + n2;`
		
		`Resp: 25`

-	Resta	`int n1 = 20;`
		`int n2 = 10;`
		`int resta = n1 - n2;`
		
		`Resp: 10`

*	Mutiplicación	`int n1 = 5;`
		`int n2 = 9;`
		`int multiplica = n1 * n2;`
		
		`Resp: 45`

/	División	`int n1 = 1;`
		`int n2 = 2;`
		`int divide = n1 / n2;`
		
		`Resp: 0`
		
		`int n1 = 1.0;`
		`int n2 = 2;`
		`int divide = n1 / n2;`
		
		`Resp: 0.5`

%	Módulo	`int n1 = 4;`
		`int n2 = 3;`
		`int resto = n1 % n2;`
		
		`Resp: 1`

2.7.2. Operadores Aritméticos Unarios

Operador	Acción	Ejemplo

+	Positivo	`char a='0';`
		`int n=+a;`
		`JOptionPane.showMessageDialog(null,""+n);`
		
		`Resp: 48`

-	Negativo	`int a=10;`
		`int n=-a;`
		`JOptionPane.showMessageDialog(null,""+n);`
		
		`Resp: -10`

2.7.3. Operadores de Incremento y Decremento

Operador	Acción	Ejemplo

++	Prefija	`int i=0;`
		`i++;`
		`JOptionPane.showMessageDialog(null,""+i);`
		
		`Resp: 1`
		`Incrementa i en 1 pero se evalúa al valor anterior al incremento.`

++	Posfija	`int i=0;`
		`++i;`
		`JOptionPane.showMessageDialog(null,""+i);`
		
		`Resp: 1`
		`Incrementa i en 1 pero se evalúa al valor posterior al incremento.`

--	Prefija	`int i=10;`
		`i--;`
		`JOptionPane.showMessageDialog(null,""+i);`
		
		`Resp: 9`
		`Decrementa i en 1 pero se evalúa al valor anterior al incremento.`

--	Posfija	`int i=10;`
		`--i;`
		`JOptionPane.showMessageDialog(null,""+i);`
		
		`Resp: 9`
		`Decrementa i en 1 pero se evalúa al valor posterior al incremento.`

2.7.4. Operadores de Asignacion

OPERADOR	ACCIÓN	EJEMPLO

=	Asignación	int n;
		n = 10;
		
		Se declara la variable n y se le asigna el valor 10, también se podría implementar de la siguiente forma:
		
		int n = 10;

2.7.5. Operadores de Comparación

OPERADOR	ACCIÓN	EJEMPLO

==	Igualdad Numérica o Caracter	Evaluar un valor numérico
		if (n == 10)
		
		Evaluar un valor decimal
		if (sueldo == 1000.00)
		
		Evaluar un valor booleano
		if (estado == true)
		
		Evaluar si el valor n ya es 10 dentro del ciclo while
		while (n == 10)

>	Mayor que	Evaluar si el valor de N supera a 10
		If (n > 10)
		
		Evaluar si el valor de N es superior a 10 dentro de un ciclo de repeticiones
		while (n > 10)

>=	Mayor o Igual que	Evaluar si el valor sueldo es mayor o igual a 1000.00 If (sueldo >= 1000.00)

<	Menor que	Evaluar si el valor de N es inferior a 10
		If (n < 10)
		
		Evaluar si el valor de N es inferior a 10 dentro de un ciclo de repeticiones
		while (n < 10)

<=	Menor o Igual que	Evaluar si el valor sueldo es menor o igual a 1000.00 If (sueldo <= 1000.00)

!=	Diferente	Evaluar si el valor de N no es igual a 10
		If (n != 10)
		
		Evaluar si el valor N no llega al valor 10 en un ciclo de repeticiones
		while (n !=10)

2.7.6. Operadores Lógicos

Operador	Acción	Ejemplo
&	Y Logica a nivel de bit	`if (n>=1 & n<=10)` Devuelve verdadero si ambas condiciones son verdad, hay que tener en cuenta que el compilador evalua a ambas expresiones.
&&	Y Logica	`if (n>=1 & n<=10)` Devuelve verdadero si ambas condiciones son verdad, hay que tener en cuenta que el compilador condicionalmente evalua solo a n<=10.
|	O Logica a nivel de bit	`if (n==1 | n==10)` Devuelve verdadero si una de las expresiones es verdad, hay que tener en cuenta que el compilador evalua a ambas expresiones.
||	O Logica	`if (n>=1 || n<=10)` Devuelve el verdadero si una de las expresiones es verdad. Debe tener en cuenta que el compilador condicionalmente evalúa solo a n<=10.
!	Negacion	`if !(n==1)` Niega el valor obtenido es una expresion logica, es decir a pesar de que N puede ser igual a uno la condicion resultaria falsa.

2.7.7. Operador de Concatenación

Operador	Acción	Ejemplo

+	Concatenar	Permite unir dos o mas expresiones.
		
		Ejm:
		
		`String dia="02";`
		`String mes="Febrero";`
		`String año="2013";`
		
		`String fecha = dia+mes+año;`
		
		Resp: 02Febrero2013
		
		Para agregarle espacios entre los valores se tiene que expresar de la siguiente forma:
		
		`String fecha = dia+" "+mes+" "+año;`
		
		En nuestras aplicaciones lo usaremos de la siguiente forma:
		
		`lblDescuento.setText("Descuento es: $ "+descuento)`
		
		donde la variable descuento se concatena con el texto "Descuento es: $" el resultado seria:
		
		Descuento es: $ 12.50

2.8. ORDEN DE PRIORIDAD DE LOS OPERADORES

Al momento de implementar una expresión para Java se debe tener en cuenta que los operadores tengan el orden necesario para una respuesta efectiva de parte de la aplicación.

Veamos la tabla de prioridad:

TIPO DE OPERADOR	OPERADORES
Operadores Posfijos	`expr++` `expr--`

Operadores Prefijos y Unarios	`++expr`
	`--expr`
	`+expr`
	`-expr`
	`!`

Aritmeticos binarios	`*`
	`/`
	`%`

Aritmeticos binarios	`+` `-`

Comparación	`<`
	`<=`
	`>`
	`>=`
	`instanceof`

Igualdad y desigualdad	`==` `!=`
Y a nivel de bit	`&`
O a nivel de bit	`|`
Y lógico	`&&`
O lógico	`||`

Operadores de Asignación	`=`
	`+=`
	`-=`
	`*=`
	`/=`
	`%=`
	`&=`
	`^=`

2.9. LOS SEPARADORES

Conforme se avanza en los temas, se dará cuenta que van apareciendo puntos nuevos en Java, los cuales presentarán una forma particular de separación. A continuación mostraremos una lista de estos y en dónde se implementarían, recuerde que toda aplicación Java contendrá los mismos separadores:

Operador	Descripción

()	Los parentesis se pueden implementar cuando:
	
	• Se especifican parámetros a los metodos
	Ejm:
	`void imprimir(String trabajador, double sueldo)`
	
	• Se invocan a los metodos
	Ejm:
	`imprimir("Manuel Torres R.",3500.00)`
	
	• Se define la precedencia en una expresion
	Ejm:
	`double r = (1.0/2) * 2;`
	
	el resultado no seria el mismo si lo expresaramos de la siguiente forma:
	`double r = 1.0/2 * 2;`
	
	• Se fuerza la conversión de tipo
	Ejm:
	`int n = 10;`
	`float x = (float) n;`

{}	Las llaves se pueden implementar cuando:
	
	• Se define el contenido de una matriz
	Ejm:
	`int[] edad = {45, 23, 11, 9};`
	
	• Se define un bloque de codigo dentro de una clase, método o estructura
	Ejm:
	`public class Persona {`
	`}`
	`public void calculaImporte(){`
	`}`
	`String asignaRegalo(){`
	`}`
	`If (edad>18){`
	`}`
	`while(i<=10){`
	`}`

OPERADOR	DESCRIPCIÓN

[]	Los corchetes se usan para definir y establecer valores referenciados a una matriz.
	Ejm:
	
	• Declaracion de arreglos
	
	`String alumnos[];`
	`int A[][];`
	
	• Asignar un valor a los arreglos
	
	`alumnos[1] = "Fernanda Torres L.";`
	
	`a[1][1] = 10;`

;	El punto y coma separa todo tipo de expresion en una aplicación Java, hay que tener en cuenta que las estructuras tienen un formato especifico para su terminacion, por ejemplo el siguiente codigo es errado:
	
	`if (estado == false);`
	
	pero, en el siguiente caso sí es válido escribirlo de la siguiente forma:
	
	`do{`
	`…`
	`}while(n<=10);`
	
	También se le puede observar dentro de la estructura for separando las clausulas del formato.
	Ejm: For para 10 numero enteros
	
	`for (int i=1;i<=10;i++)`

,	La coma separa identificadores consecutivos que se pueden realizar en una declaracion de variables o la separacion de parámetros en un método dentro de una aplicación Java.
	
	Ejm: Declarar 5 variables de tipo entero
	
	`int n1,n2,n3,n4,n5;`
	
	Ejm: implementar 3 variables de tipo entero a un metodo
	
	`void calculaPromedio(int n1, int n2, int n3)`
	
	También se le puede observar dentro de la estructura for.
	Ejm:
	
	`for (int i=1,int j=0;i<=10,j<=9;i++,j++)`

Operador	Descripción

.	El punto separa el nombre de los paquetes de sus clases contenedoras.
	Ejm:
	`imports javax.swing.*;`
	También se puede usar para invocar a variables o métodos procedentes de una instancia de clase.
	Ejm:
	`Persona objPersona = new Persona();`
	`objPersona.nombres = "Angela Torres L.";`

2.10. Los comentarios

Como habíamos comentado en los capítulos anteriores, Java interpreta una aplicación y la convierte en un código de bytes; solo faltó mencionar que cuando realiza dicha actividad obvia los comentarios implementados dentro de una aplicación Java. Se pueden implementar comentarios en algunos de los siguientes casos:

- Cuando se necesite especificar algún proceso o actividad del código.
- Cuando se necesite especificar la autoría de la aplicación.
- Cuando no necesite que una o más líneas de código sea interpretado por el compilador de Java.

Tenemos:

Operador	Descripción

//	Comentario de una sola línea.
	Ejm:
	- Comentario simple
	//Calcula el Promedio de Notas
	- Anulando a una expresion
	//int n = 10;
	- Aplicando un comentario dentro de una expresion
	double raiz; //Declarando raiz como doble
	raiz = Math.pow(n,(1/2.0)); //Eleva a la potencia N

/*
*/
	Comentario de varias lineas.
	Ejm:
	`- Especificar autoria de la aplicación`
	`/*`
	`Autor: Lic. Manuel Torres R.`
	`Fecha: 03 Febrero 2013`
	`IDE: JCreator LE`
	`*/`

2.11. EXPRESIONES EN JAVA

Una expresión en una aplicación es un conjunto de elementos que se forma para un objetivo especifico, en Java para indicar que una expresión ha finalizado, se debe colocar un punto y coma al final de la misma. Veamos algunos ejemplos de expresiones:

- La declaración-asignación de un valor a la variable N es una expresión simple:

```
int n = 10;
```

- La fórmula para hallar el valor X es otro modelo de expresión:

```
double x = (1.2 + 5.0) * Math.pow(2);
```

- La implementación de una condición es otro modelo de expresión:

```
double tarifa = 0;
if (categoria.equals("A")){
        tarifa = 15.50;
        descuento = 0.18;
}
```

Hay que tener en cuenta que para implementar una expresión se debe considerar el patrón que el lenguaje Java propone, para esto veremos algunas expresiones algebraicas que serán convertidas a una expresión legible por el compilador de Java.

Si tenemos la siguiente expresión:

$$r = 5a^2b^3 + \sqrt{a^2 + b^2}$$

Y analizamos la expresión tendremos:

- Los valores a y b no son conocidos; por lo tanto, son valores que deben ser ingresados por el usuario.
- Existe una raíz cuadrada que en Java se implementará con el método pow de la clase Math.
- Finalmente el resultado de la expresión no ha sido especificada, pero como desarrollador usted le pondrá una variable de respuesta.

La conversion sería:

```
double x;
x=5*Math.pow(a,2)* Math.pow(b,3)+Math.sqrt(Math.pow(a,2)+ Math.pow(b,2));
```

O también podría ser de la siguiente forma:

```
double x;
x=5*a*a*b*b*b+Math.sqrt((a*a)+(b*b));
```

Veamos algunos casos:

Expresión Aritmética	Expresión Java
$\sqrt[3]{4ab^3}+(a+b)^2$	double x; x=Math.pow(4*a*Math.pow(b,3),1.0/3)+Math.pow(a+b,2);
$\frac{a^3}{2ab^3}-\sqrt{12d^4}$	double x; x=Math.pow(a,3)/(2*a*Math.pow(b,3))- Math.sqrt(12*Math.pow(d,4));

2.12. Tipo de datos

En los lenguajes de programación antes del paradigma orientado a objetos no se usaban las instancias de clases ni mucho menos se hablaba de objetos y clases, solo estaban basadas en las variables. Estas variables dependían su valor del tipo de datos como se le declaraba, vale decir, que si una variable se declaraba como entera los valores asignados a la variable serían solo valores enteros.

En Java los tipos de datos tienen las siguientes características:

- Una variable no es un objeto.
- Se tienen 8 tipos de datos primitivos en 4 grupos que son Enteros, Reales, Caracteres y Boleanos.
- Un tipo de datos determina el rango que una variable puede tener.
- La definición de una variable en Java no distingue la plataforma de trabajo.
- Java es un lenguaje de programación estrictamente tipado, es decir, toda variable debe ser declarada con su propio tipo de datos.

Según el tipo de información que se almacene dentro de una variable se asignará un tipo de datos siguiendo dos tendencias:

- Tipos de datos Primitivos: estos representan a un solo valor. La forma de identificar un tipo de datos primitivo es que siempre se inicia con una letra minúscula.
- Tipos de datos por referencia: estos representan a un conjunto de valores como por ejemplo los arreglos, clases, interfaces, etc. La forma de identificarlos es que siempre se inician con una letra mayúscula.

Veamos los tipos de datos agrupados por sus categorías:

Categoria	Tipo de Datos
Enteros	byte (entero de 8 bits) short (entero de 16 bits) int (entero de 32 bits) long (entero largo de 64 bits)
Reales	float (coma flotante de precision simple de 32 bits) double (coma flotante de precision doble de 64 bits)
Carácter	char (caracter)
Boleanos	boolean (logico)

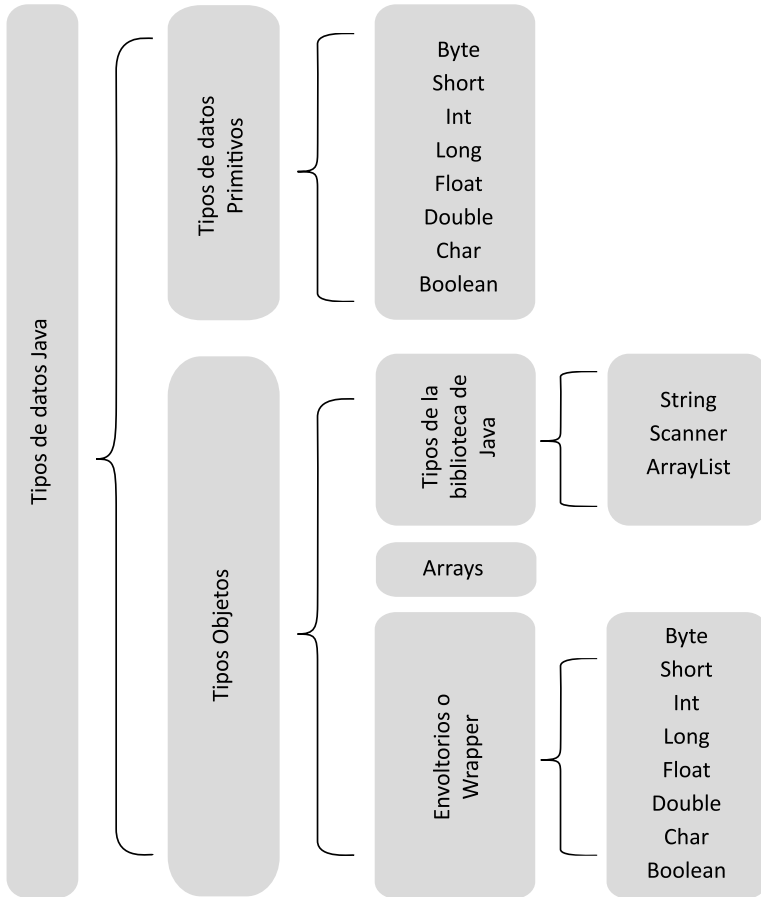
Ahora veamos las capacidades de los tipos de datos primitivos:

TIPO DE DATOS	CAPACIDAD MINIMA	CAPACIDAD MAXIMA
byte	-128	+127
short	-32768	+32767
int	-2147483648	+2147483647
long	-9223372036854775808	+9223372036854775807
float	±3.40282347E+38	±1.40239846E-45
double	±1.79769313486231570E+308	±4.94065645841246544E-324
char	\u0000	\uFFFF

También se debe considerar el valor por defecto de cada tipo de datos primitivos, veamos una lista de dichos valores:

TIPO DE DATOS	VALOR POR DEFECTO
byte	`0`
short	`0`
int	`0`
long	`0L`
float	`0.0f`
double	`0.0d`
char	`'\u000'`
boolean	`False`

Finalmente, veamos una lista de los tipos de datos en Java:



2.13. Los literales de los tipos de datos primitivos en Java

Una literal en Java representa a un valor constante formado por una secuencia de caracteres. Este literal dependerá directamente del tipo de datos y su rango.

- **Literal de tipo entero**

  Este puede expresarse en decimal, es decir, en base10, octal en base 8 y hexadecimal en base 16. Cuando el valor es positivo no será necesario especificarlo por el símbolo + caso contrario al negativo que obligatoriamente se colocará dicho símbolo.

Literal	Caso	Ejemplo
Entero base 10	La edad de una persona La nota en un curso Numero de eventos Stock de productos	35 10 1099567890L -10
Entero base Octal	Base 8	028
Entero base 16	Basde 16	0xf4

- **Literal real**

Este puede expresarse en decimal de base 10, el cual contiene un parte entera y otra fraccionaria y el signo necesario de especificación es el negativo.

LITERAL	CASO	EJEMPLO
Real base 10	Estatura de una persona Valor de PI Sueldo de un trabajador Interes perdido de un banco	1.70 3.1416 3500.00 -0.000083
Notacion Cientifica	Notacion Cientifica para $14x10^3$ Notacion Cientifica para $0.0006x10^4$	14E-3 .0006E4
Float	Peso de una persona	75.56F
Double	Peso de una persona	75.56D

- **Literal carácter**

Este puede expresar un valor de un solo carácter, el cual se referencia por medio de comillas simples.

LITERAL	CASO	EJEMPLO
Caracter	Categoría de un trabajador Cambio de línea Tabulación	'A' '\n' '\t'

- **Literal booleano**

Este puede expresar los valores True y False.

LITERAL	CASO	EJEMPLO
Booleanos	Sexo de un trabajador Condicion de un alumno aprobado	True False

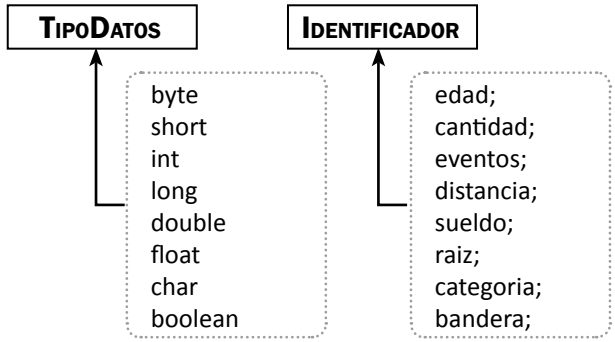
2.14. LAS VARIABLES Y SU DECLARACION EN JAVA

Una variable es derivada del término .variabilis" que representa a todo aquello que varía o que está sujeto a algún tipo de cambio durante un determinado proceso. El compilador de Java gestiona por medio de la variable un espacio en la memoria de la computadora. En dicha variable se puede almacenar algún valor permitido por el tipo de datos declarada a la variable.

Al declarar variables se debe tener en cuenta:

- Asignar un nombre dependiendo de lo que va a almacenar; por ejemplo, si necesitamos almacenar un salario bruto entonces se debe declarar la variable como salarioBruto.
- Si la variable declarada es totalmente en mayúsculas así deberá usarse dentro de la aplicación esto es debido a la sensibilidad a las mayúsculas y minúsculas de Java.
- Toda variable se compone de un nombre, tipo y valor a almacenar en él.

Formato:



Veamos un caso de declaración de variables a partir de un caso:

Una empresa desea calcular el pago mensual realizado a sus empleados, el cual cuenta con las siguientes características:

Sueldo básico: se calcula en base al número total de horas trabajadas basado en una tarifa horaria.
Bonificación: se calcula obteniéndose el 20% del sueldo básico.
Sueldo neto: se obtiene de descontar el 10% de la diferencia entre el sueldo básico y su bonificación.

Entonces. se desea implementar una aplicación Java que permita calcular el sueldo básico, la bonificación y el sueldo neto de un empleado.

Solución:

Variables	Declaracion en Java
Nombre del empleado	`String empleado;`
Horas trabajadas	`int horas;`
Tarifa Horaria	`double tarifa;`
Sueldo Basico	`double sueldoBasico;`
Bonificacion	`double bonificacion;`
Sueldo Neto	`double sueldoNeto;`

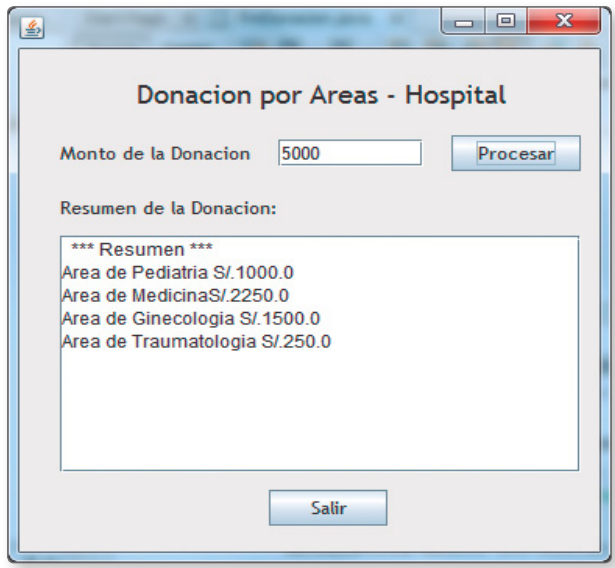
Veamos un caso de declaración de variables a partir de una expresión:

$$r = 5a^2b^3 + \sqrt{a^2 + b^2}$$

Solución:

Variables	Declaracion en Java
a	`double a;`
b	`double b;`
r	`double r;`

Veamos un caso de declaración de variables a partir de un formulario:



Solución:

VARIABLES	DECLARACION EN JAVA
Donacion	`double donacion;`
Monto Pediatria	`double mPediatria;`
Monto Medicina	`double mMedicina;`
Monto Ginecologia	`double mGinecologia;`
Monto Traumatologia	`double mtraumatologia;`

2.15. LA CLASE STRING

Java posee gran capacidad para el manejo de cadenas dentro de sus clases String y StringBuffer. Un objeto String representa una cadena alfanumérica de un valor constante que no puede ser cambiada después de haber sido creada. Un objeto StringBuffer representa una cadena cuyo tamaño puede variar.

Los Strings son objetos constantes, y por lo tanto, muy baratos para el sistema. La mayoría de las funciones relacionadas con cadenas esperan valores String como argumentos y devuelven valores String.

Paquete: java.lang.String

Métodos Constructores:

String	Permite inicializar un nuevo String vacio.
String("Cadena")	Permite inicializar un nuevo String con un valor predeterminado.

Métodos:

MÉTODO	EXPLICACIÓN

charAt	Retorna el carácter de una posición específica de la cadena, tenga en cuenta que cada carácter es almacenado en una posición determinada del arreglo unidimensional y que los espacios en blanco también ocupan un espacio.
	
	Ejm. Si tenemos la siguiente declaración:
	
	String cadena = "Programación Java";
	
	P r o g r a m a c I o n J a v a
	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
	
	Y se necesita imprimir los caracteres ubicados en la posición 1, 4 y 13. El código sería:
	
	`JOptionPane.showMessageDialog(null,"Posicion 1:"+cadena.charAt(1));`
	`JOptionPane.showMessageDialog(null,"Posicion 4:"+cadena.charAt(4));`
	`JOptionPane.showMessageDialog(null,"Pos. 13:"+cadena.charAt(13));`
	
	Ahora, si quisiéramos imprimir el reverso de la cadena sería de la siguiente forma:
	
	`for(int i=cadena.length()-1;i>=0;i--)`
	`JOptionPane.showMessageDialog(null,cadena.charAt(i));`
	
	Primero, debemos determinar la cantidad de caracteres que tiene la cadena; para eso usamos el metodo length() e implementamos la estructura for para que recorra en forma descendente. Luego, se imprimen los caracteres por medio de un mensaje.

compareTo	Compara dos cadenas en forma lexicográfica. Emitirá un valor cero (0) si las cadenas comparadas son exactamente iguales, caso contrario enviará el valor uno (1).
	
	Ejm. Veamos el siguiente script:
	
	`//1.`
	`String cadena="Java";`
	`int i=cadena.compareTo("Java");`
	
	`//2.`
	`if (i==0)`
	`JOptionPane.showMessageDialog(null,"Son Iguales");`
	`else`
	`JOptionPane.showMessageDialog(null,"No son Iguales");`
	
	En el punto uno se declara y asigna con el valor Java a la variable cadena, luego se crea la variable numérica i que obtendrá el valor numérico de la comparación (0 iguales – 1 No iguales).
	
	En el punto dos se implementa la condicional if en la cual se compara qué valor devolvió la comparación y de acuerdo a esto emitirá mensajes.

MÉTODO	EXPLICACIÓN

concat	Permite concatenar dos cadenas en una sola.
	
	Ejm. Veamos el siguiente script:
	
	`String cadena="Programacion ";`
	`String nuevaCadena=cadena.concat("Java");`
	`JOptionPane.showMessageDialog(null,nuevaCadena);`
	
	El resultado de la concatenación es: Programación Java.

equals	Compara la cadena con un objeto especificado.
	
	Ejm:
	
	`//1.`
	`String cadena1="Java";`
	`String cadena2="java";`
	
	`//2.`
	`if (cadena1.equals(cadena2))`
	`JOptionPane.showMessageDialog(null,"Son Iguales");`
	`else`
	`JOptionPane.showMessageDialog(null,"No son Iguales");`
	
	En el punto uno se declara la variable cadena1 con el valor Java donde la primera letra es mayúscula y en la cadena2 con el valor Java totalmente en minúsculas.
	
	En el punto dos se implementa la condición que compara la cadena1 con la cadena2 con el método equals cadena1.equals(cadena2) o también cadena2.equals(cadena1), al final se imprimen los mensajes dependiendo de la igualdad de las cadenas.

equalsIgnoreCase	Compara una cadena con otra sin considerar mayúsculas.
	
	Ejm. Logeo de usuario y su clave:
	
	`//1.`
	`String usuario=JOptionPane.showInputDialog(null,`
	`"Ingrese usuario: ");`
	`String clave=JOptionPane.showInputDialog(null,`
	`"Ingrese clave: ");`
	
	`//2.`
	`if (usuario.equalsIgnoreCase("mtorres") &&`
	`clave.equalsIgnoreCase("abc"))`
	`JOptionPane.showMessageDialog(null,"Bienvenido usuario");`
	`else`
	`JOptionPane.showMessageDialog(null,"Error de usuario");`
	
	En el punto uno se declara la variable usuario y clave, para poder asignar los valores que el usuario ingrese se usa la instrucción showInputDialog que permite ingresar un valor textual.
	
	En el punto dos se compara el valor ingresado con el usuario y la clave predeterminada, al usar el método equalsIgnoreCase se podrá ingresar el nombre del usuario y la clave en mayúsculas o minúsculas.

<table>
<tr><th>MÉTODO</th><th>EXPLICACIÓN</th></tr>
<tr><td>getChars</td><td>Copia caracteres de una cadena a un arreglo de caracteres.

Ejm. Copiar de una cadena a otra:

<pre>
//1.
char vocales[]={'a','e','i','o','u'};
String cadena="AEIOU";

//2.
cadena.getChars(1, 3, vocales, 1);

//3.
for(int i=0;i<5;i++)
   JOptionPane.showMessageDialog(null, vocales[i]);
</pre>
Resp: a E I o u

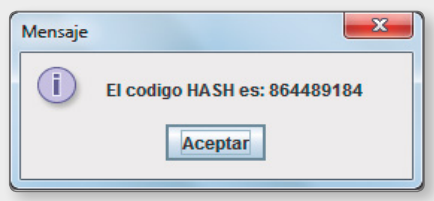
En el punto uno se declara la variable vocales inicializada con las vocales en minúsculas, luego se declara la variable cadena con el valor inicial AEIOU.

En el punto dos se usa el método getChars en donde a la variable vocales se le agrega los caracteres de la variable cadena desde la posición 1 hasta 3.

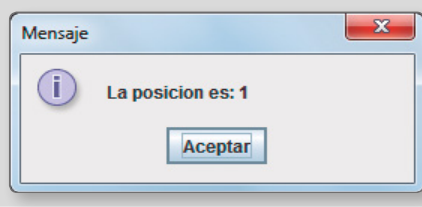
En el punto tres se imprimen los caracteres de la variable vocales para acceder a los caracteres se usa la estructura for.</td></tr>
<tr><td>hashCode</td><td>Retorna el código hash para una cadena especificada.

Ejm. El siguiente ejemplo determina el numero HASH asignado a la cadena Programación Java:

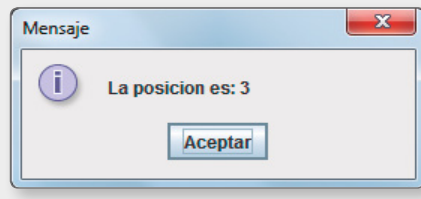
<pre>
String cadena="Programacion Java";
int hash=cadena.hashCode();

JOptionPane.showMessageDialog(null,
                              "El codigo HASH es: "+hash);
</pre>

</td></tr>
<tr><td>indexOf</td><td>Retorna la posición según el numero de carácter Ascii incluido en una determinada cadena. Retornara -1 cuando el código buscado no se encuentra en la cadena.

Ejm. El siguiente ejemplo determina la posición del carácter Ascii 97 (a minúsculas).

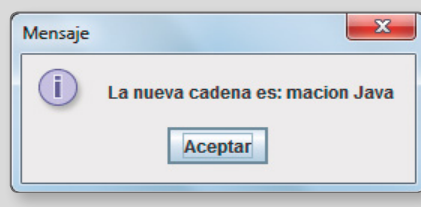
<pre>
String cadena="Java";
int valor=cadena.indexOf(97);
JOptionPane.showMessageDialog(null, "La posicion es: "+valor);
</pre>

</td></tr>
</table>

MÉTODO	EXPLICACIÓN

lastIndexOf	Retorna la posición del carácter Ascii empezando por el final de la cadena. Retornara -1 cuando el código buscado no se encuentra en la cadena.
	
	Ejm. El siguiente ejemplo determina la posición del carácter Ascii empezando por el final de la cadena:
	
	`String cadena="Java";`
	`int valor=cadena.lastIndexOf(97);`
	`JOptionPane.showMessageDialog(null, "La posicion es: "+valor);`
	
	 

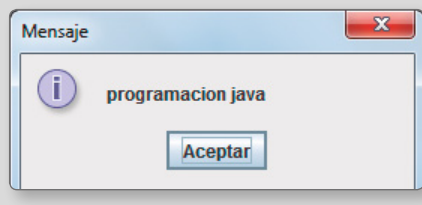
length	Retorna el número total de caracteres de una determinada cadena incluyendo espacios en blanco.
	
	Ejm. Determinar la longitud de la cadena PROGRAMACIÓN JAVA.
	
	`String cadena="PROGRAMACION JAVA";`
	`int longitud=cadena.length();`
	`JOptionPane.showMessageDialog(null,`
	`"La longitud es: "+longitud);`
	
	Resp: 17 caracteres

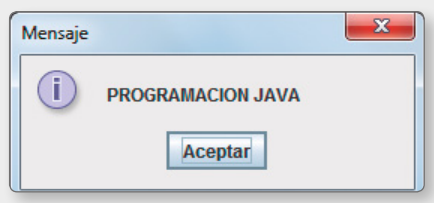
replace	Reemplaza un carácter por otro dentro de una cadena.
	
	Ejm. En el siguiente ejemplo se reemplaza todos los caracteres `'e'` por la `'a'`.
	
	`String cadena="Jeve";`
	`cadena=cadena.replace('e','a');`
	`JOptionPane.showMessageDialog(null,`
	`"La nueva cadena es: "+cadena);`
	
	Resp: Java

subString	Permite retornar una parte de una cadena especificando la posición de captura.
	
	Ejm. En el siguiente ejemplo se substrae de la cadena “Programación Java” desde la posición 6.
	
	`String cadena="Programacion Java";`
	`String nuevacadena=cadena.substring(6);`
	`JOptionPane.showMessageDialog(null,`
	`"La nueva cadena es: "+nuevacadena);`
	
	 

MÉTODO	EXPLICACIÓN

toCharArray	Convierte el string en arreglo de caracteres.
	
	Ejm:
	
	`//1.`
	`String cadena="Programacion Java";`
	`char arreglo[]=cadena.toCharArray();`
	
	`//2.`
	`for(int i=0;i<cadena.length();i++)`
	`JOptionPane.showMessageDialog(null,arreglo[i]);`
	
	En el punto uno se declara la variable cadena con el valor predeterminado "Programación Java", luego este se envía a la variable arreglo con el método toCharArray.
	
	En el punto dos se imprimen los caracteres contenidos dentro del arreglo usando la estructura repetitiva for.

toLowerCase	Permite convertir una cadena en minúsculas.
	
	Ejm. En el siguiente ejemplo se convierte la cadena "PROGRAMACIÓN JAVA"
	
	`String cadena="PROGRAMACION JAVA";`
	`JOptionPane.showMessageDialog(null,cadena.toLowerCase());`
	
	 

toUpperCase	Permite convertir una cadena a mayúsculas.
	
	Ejm. En el siguiente ejemplo se convierte la cadena "programación java"
	
	`String cadena="programacion java";`
	`JOptionPane.showMessageDialog(null,cadena.toUpperCase());`
	
	 

trim	Permite eliminar los espacios blancos al final de una cadena de caracteres.
	
	Ejm. En el siguiente ejemplo permite eliminar los espacios en blanco contenido dentro de la cadena "programación java".
	
	`String cadena="programacion java         ";`
	`JOptionPane.showMessageDialog(null,cadena.trim());`

MÉTODO	EXPLICACIÓN

valueOf	Permite retornar la representación de la cadena.
	
	Ejm. En el siguiente ejemplo se tiene un cuadro combinado (Combobox) con 4 país predeterminados, al seleccionar uno de ellos le devuelve el nombre del país seleccionado desde el Combobox.
	
	`String pais;`
	`pais=String.valueOf(cboPais.getSelectedItem());`
	`JOptionPane.showMessageDialog(null,`
	`"El Pais seleccionado es: "+pais);`

Opciones:

- Declarar un objeto de la clase String > String cadena;
- Crear un objeto de tipo String > cadena = new String(.Java");

2.16. CONVERSIONES ENTRE TIPOS DE DATOS EN JAVA

Las conversión de tipos se refiere a la transformación que se puede realizar entre tipos de datos en una misma aplicación, a este proceso se le conoce como moldeado o tipado. Así tenemos:

- **La conversion automatica:** esta se realiza cuando el compilador determina que el tipo de datos origen es inferior en capacidad al tipo de datos destino. Por ejemplo, la conversión de una variable de tipo byte a integer no necesita realizar conversión alguna ya que solo con almacenar el valor de tipo byte al integer la conversión es efectuada. Es también llamado ensanchamiento o promoción de tipo ya que el tipo de datos pequeño se promociona en el tipo de datos más grande. Veamos un código de conversión automática:

```
//1.
byte n=100;
int x=n;

//2.
JOptionPane.showMessageDialog(null,"El Valor de X es: "+x);
```

En el punto uno se declara la variable n de tipo byte (-128 a +127) y se le asigna el valor 100. Luego, se declara la variable x que se le asigna el valor de n, por tanto x almacena también 100, siendo 100 un límite permitido en los enteros la conversión se realiza en forma automática. En el punto dos se imprime la variable x.

- **Conversion explícita:** esta se realiza cuando un valor de tipo de datos de mayor capacidad es convertida a una de inferior capacidad. A esto se le llama estrechamiento debido a que se estrecha explícitamente el valor para que quepa en el tipo de datos destino. Veamos el código de conversión explícita:

```
//1.
int n=100;
byte x=(byte)n;

//2.
JOptionPane.showMessageDialog(null,"El Valor de X es: "+x);
```

En el punto uno se declara la variable n de tipo entero con el valor 100, luego se declara la variable x de tipo byte haciendo una conversión de tipo explícita de la variable n al tipo byte, el formato indica que se debe colocar en paréntesis del lado izquierdo. En el punto dos se imprime el valor de x convertido a byte.

Cuando se realizan conversiones debemos conocer exactamente la capacidad de cada uno de ellos (revise la tabla de capacidades de los tipos primitivos) ya que todos los tipos de datos se pueden convertir, pero no asegura una conversión segura entre ellas. Si tenemos una variable de tipo integer (32 bits) y deseamos convertirla al tipo byte (8 bits) se perderán 24 bits en plena conversión, esto podría ocasionar una pérdida importante al valor original entero. Para esto debemos considerar la siguiente tabla:

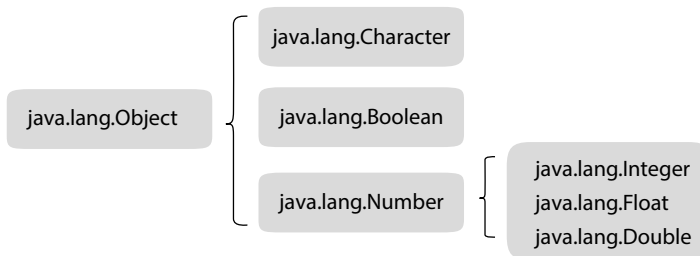
Tipo de datos origen	Tipo de datos destino
byte	double, float, long, int, char, short
short	double, float, long, int
char	double, float, long, int
int	double, float, long
long	double, float
float	double

La tabla muestra la norma que se necesita para las conversiones sin pérdida de información.

2.17. Clases envoltorio o Wrapper

Hasta este punto ya debemos reconocer los tipos de datos primitivos de Java y la forma en que interactúa en una aplicación Java. Así como vimos en un gráfico sobre los tipos de datos en Java los arrays, listas enlazadas, colecciones o conjuntos de Java necesitarán en algún momento usar los tipos de datos primitivos, eso quiere decir que estos valores deben ser tratados como objetos.

El lenguaje Java ofrece un conjunto de clases envoltorios exactamente para cada uno de los tipos de datos primitivos. La característica principal de estas clases es que empiezan con una letra mayúscula caso contrario en los tipos de datos primitivos que siempre empiezan con una letra minúscula. A la vez permiten el tratamiento de los tipos de datos primitivos como objetos, proporcionando así diferentes métodos que serán de gran utilidad para nuestras aplicaciones Java.
Veamos los componentes de la clase Object.



Veamos los tipos de datos primitivos y sus wrapper:

TIPOS DE DATOS PRIMITIVO	WRAPPER
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

2.18. LA CLASE INTEGER

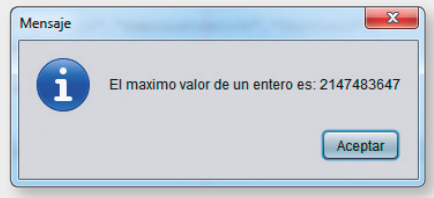
Es la clase más usada de todos los wrappers ya que posee métodos que permiten convertir un valor de tipo int en otro como long, float o double. La clase Integer envuelve un valor de tipo int en un objeto, este objeto contiene un campo simple cuyo tipo de datos es int.

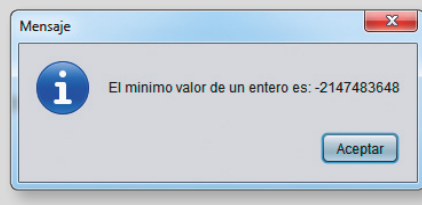
Constructor:

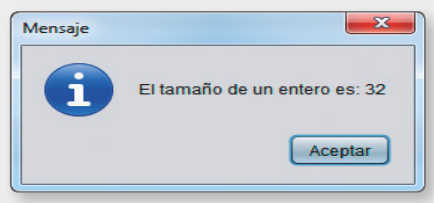
INTEGER(INT VALOR)	Permite construir un objeto de la clase Integer a partir de un parámetro de tipo int.
	
	Ejm:
	
	`Integer n = new Integer(20);`
	`JOptionPane.showMessageDialog(null,`
	`"El Valor de N es: "+n);`

INTEGER(STRING VALOR)	Permite construir un objeto de la clase Integer a partir de un parámetro de tipo String.
	
	Ejm:
	
	`String n = "1250";`
	`Integer cantidad=new Integer(n);`
	`JOptionPane.showMessageDialog(null,`
	`"El Valor de N es: "+cantidad);`

Métodos:

MAX_VALUE	Permite devolver el valor maximo de un tipo int .
	
	Ejm:
	
	`int n = Integer.MAX_VALUE;`
	`JOptionPane.showMessageDialog(null,`
	`"El maximo valor de un entero es: "+n);`
	
	 

MIN_VALUE	Permite devolver el minimo valor de un tipo int.
	
	Ejm:
	
	`int n = Integer.MIN_VALUE;`
	`JOptionPane.showMessageDialog(null,`
	`"El minimo valor de un entero es: "+n);`
	
	 

SIZE	Permite devolver el tamaño en bits usado al representar un valor de tipo int.
	
	Ejm:
	
	`int t = Integer.SIZE;`
	`JOptionPane.showMessageDialog(null,`
	`"El tamaño de un entero es: "+t);`
	
	 

MÉTODO	DESCRIPCIÓN

byteValue()	Convierte el valor del objeto Byte al tipo de datos primitivo byte.
	
	Ejm:
	
	`Byte b=new Byte(Byte.MAX_VALUE);`
	`byte x = b.byteValue();`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`

compare(int x, int y)	Compara dos valores numericos de tipo int.
	
	Ejm:
	
	`Integer a = new Integer(10);`
	`Integer b = new Integer(10);`
	`int x=Integer.compare(a,b);`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`

doubleValue()	Convierte el valor del objeto Double al tipo de datos primitivo double.
	
	Ejm:
	
	`Double d=new Double(Double.MAX_VALUE);`
	`double x = d.doubleValue();`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`

equals(Object obj)	Permite comparar dos valores de la clase Integer.
	
	Ejm:
	
	`Integer n=new Integer(100);`
	`Integer m=new Integer(100);`
	`if (n.equals(m))`
	`JOptionPane.showMessageDialog(null,`
	`"Son iguales");`
	`else`
	`JOptionPane.showMessageDialog(null,`
	`"No Son iguales");`

floatValue()	Convierte el valor del objeto Float al tipo de datos primitivo float.
	
	Ejm:
	
	`Float d=new Float(Float.MAX_VALUE);`
	`float x = d.floatValue();`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`

MÉTODO	DESCRIPCIÓN

intValue()	Convierte el valor del objeto Integer al tipo de datos primitivo int.
	
	Ejm:
	
	`Integer i=new Integer(Integer.MAX_VALUE);`
	`int x = i.intValue();`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`

longValue()	Convierte el valor del objeto Long al tipo de datos primitivo long.
	
	Ejm:
	
	`Long i=new Long(Long.MAX_VALUE);`
	`long x = i.longValue();`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`

parseInt(String s)	Devuelve el valor de tipo int desde un valor de tipo String. Hay que tener en cuenta que el valor tiene que ser numérico, caso contrario genera un error de excepcion llamada NumberFormatException.
	
	Ejm:
	
	`String n="100";`
	`int x = Integer.parseInt(n);`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`
	
	En el primer script se declara n como un valor de tipo String (cadena) el cual debe convertirse a entero para este efecto se usa el método parseInt.
	
	`String n=txtN.getText();`
	`int x = Integer.parseInt(n);`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`
	
	En este segundo script se inicializa la variable n de tipo con el valor ingresado por el usuario mediante el objeto txtN. Luego se convierte a entero gracias al método parseInt, tambien podría definirse de la siguiente forma:
	
	`int x=Integer.parseInt(txtN.getText);`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`

shortValue()	Convierte el valor del objeto Short al tipo de datos primitivo short.
	
	Ejm:
	
	`Short s=new Short(Short.MAX_VALUE);`
	`short x = s.shortValue();`
	`JOptionPane.showMessageDialog(null,`
	`"El valor de X es: "+x);`

MÉTODO	DESCRIPCIÓN

toString()	Permite convertir un numero entero en un objeto de tipo String.
	
	Ejm:
	
	`Integer n = new Integer(100);`
	`String x=n.toString();`
	`JOptionPane.showMessageDialog(null,`
	`"El Valor de X es: "+x);`
	
	Tambien se puede representar:
	
	`Integer n = new Integer(100);`
	`JOptionPane.showMessageDialog(null,`
	`"El Valor de N es: "+n.toString());`

toString(int i)	Permite convertir un numero entero en un objeto de tipo String usando al numero entero como parametro.
	
	Ejm:
	
	`int n=Integer.MAX_VALUE;`
	`JOptionPane.showMessageDialog(null,`
	`"El Valor de N es: "+Integer.toString(n));`

valueOf(int i)	Permite convertir un valor de tipo int en un objeto de la clase Integer.
	
	Ejm:
	
	`int n=Integer.MAX_VALUE;`
	`JOptionPane.showMessageDialog(null,`
	`"El Valor de N es: "+Integer.valueOf(n));`

valueOf(String s)	Permite devolver un objeto de la clase Integer a partir de un parametro de tipo String.
	
	Ejm:
	
	`String n="123";`
	`JOptionPane.showMessageDialog(null,`
	`"El Valor de N es: "+Integer.valueOf(n));`

2.19. CREANDO UN PROYECTO CON JCREATOR

JCreator es un entorno de desarrollo integrado para programación en lenguaje Java bajo el entorno Windows, la aplicación se puede ejecutar en todas las versiones de Windows (este material está usando la versión Windows Seven).

Hay que tener en cuenta que JCreator es un IDE bastante ligero puesto que no cuenta con herramientas visuales, eso no quiere decir que no lo implemente. Existen dos ediciones del IDE, una gratuita, llamada JCreator LE y otra llamada JCreator Pro. Para el libro se usó Jcreator LE que también está en el contenido del CD que acompaña a este material

URL de descarga: http://www.jcreator.org/download.htm

Acceso a la aplicación:

CREANDO UNA APLICACIÓN CON JCREATOR LE

PASO 1:

File > New > Blank Workspace… como se muestra en la Fig. 2.2

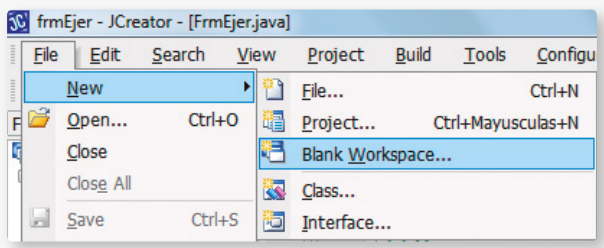


Fig. 2.2

PASO 2:

En la Fig. 2.3 se debe asignar un nombre al Espacio de Trabajo y la ubicación del mismo. Con respecto al nombre no hay restricciones de espacios ni cantidad de caracteres; por tanto, podría ser "Sistema de Ventas" o "Control de Notas" En el caso de la ubicación se tiene que crear previamente una carpeta y luego ubicarla a partir de la opción Location presionando en el botón …, finalmente se debe presionar el botón Next>.

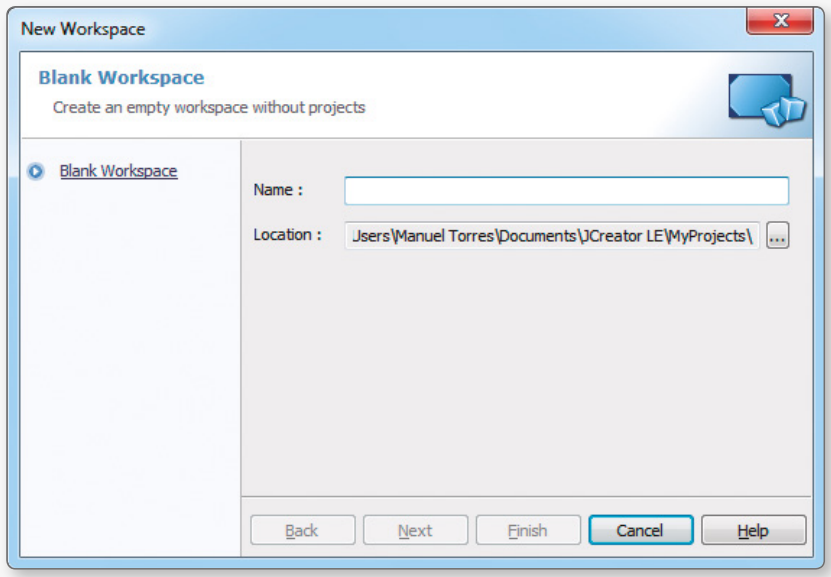


Fig. 2.3

Una vez creado el espacio de trabajo se le agrega un nuevo proyecto presionando clic derecho sobre el proyecto > Add New Project o también se puede agregar desde el menú Project > New Project…

Desde la fig. 2.4 se debe seleccionar la plantilla de acuerdo con el tipo de aplicación que se desea realizar en JCreator. Para este caso, se seleccionará **Basic Java Applet**. *Luego presione Next >.*

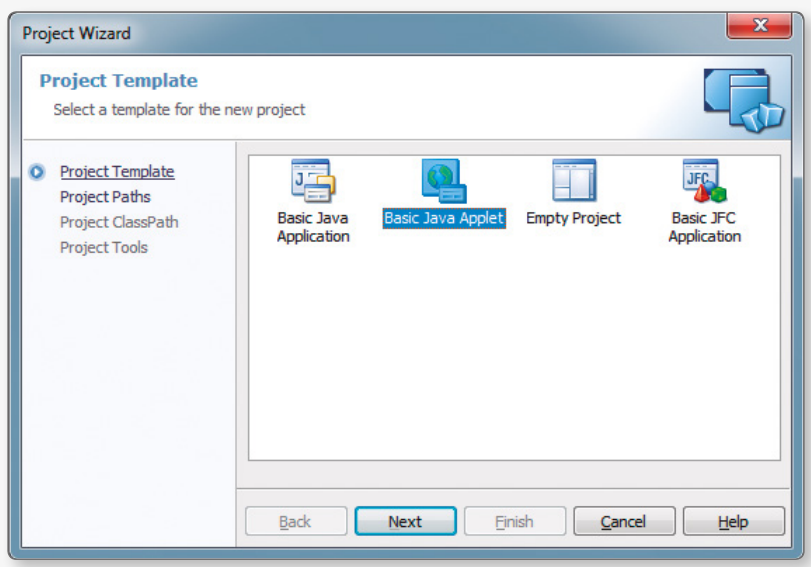


Fig. 2.4

PASO 3:

En la fig. 2.5, se muestran los valores ingresados al proyecto; en este caso, se coloca el nombre "frmPago" y ya no tiene que configurar Location, Source Path u OutPut Path, ya que todo eso se configuró al crear el espacio de trabajo.

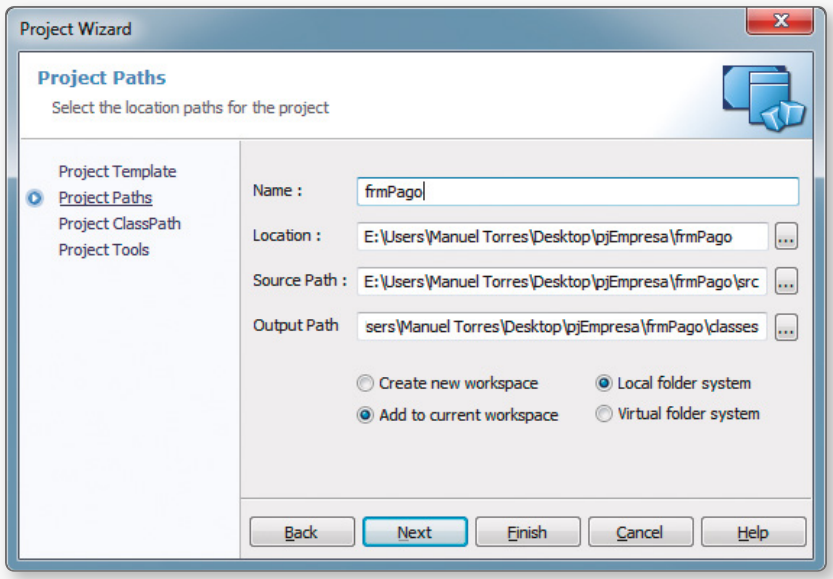


Fig. 2.5

PASO 4:

En la figura, se muestra la estructura del proyecto Applet frmPago.

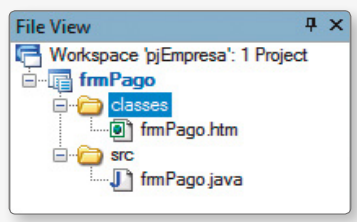


Fig. 2.6

Donde:

- *clasess: almacena los archivos de compilación y el htm de la aplicación Applet.*
- *src: almacena los archivos .java*

PASO 6:

En la Fig. se muestra una implementacion GUI del caso Pago de Empleados.

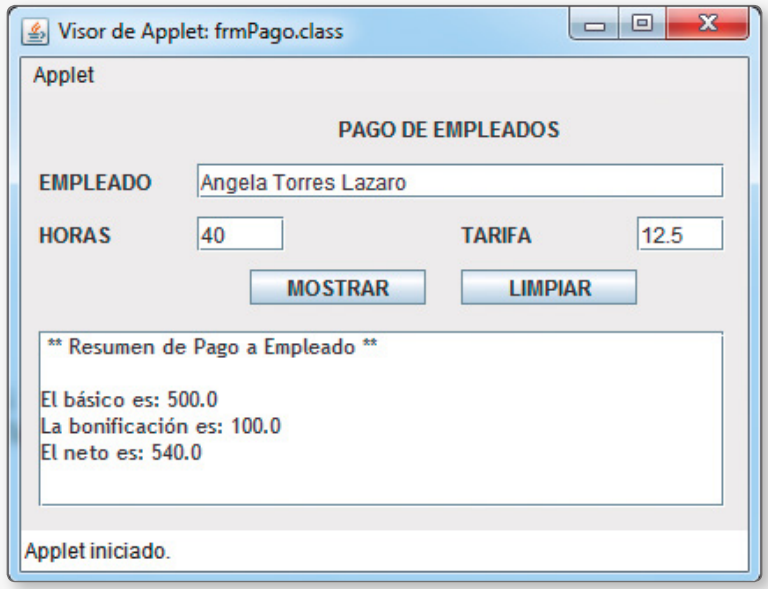


Fig. 2.7

2.20. CREANDO UN PROYECTO CON NETBEANS

Es un IDE que permite editar código del lenguaje Java, Netbeans es un proyecto de código abierto cuyo patrocinador principal es Sun Microsystem. Comenzó como un proyecto estudiantil en la República Checa en 1996 en la Universidad Carolina en Praga. Jarda Tulach diseñó la arquitectura básica de la IDE propuso la idea de llamarlo NetBeans.

NetBeans permite escribir, compilar, depurar y ejecutar aplicaciones Java. Tambien permite implementar aplicaciones en otros lenguajes como PHP o C++. En la actualidad, NetBeans soporta todos los tipos de aplicaciones Java como J2SE, EJB y aplicaciones móviles.

URL de Descarga: http://netbeans.org/downloads/

Acceso a la aplicación:

CREANDO UNA APLICACIÓN CON NETBEANS

PASO 1:

File > New Project seleccionamos la categoría **Java** *> luego* **Java Application** *como se muestra en la Fig. 2.8. Finalmente presione* **Next>**.

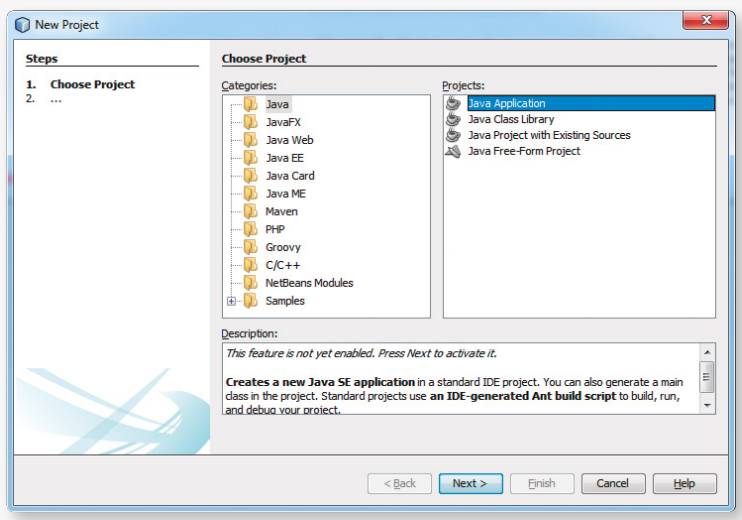


Fig. 2.8

PASO 2:

Luego, debemos asignar un nombre al proyecto y una locación. Para este caso se le asignó el nombre pjEmpresa al proyecto, pero como lo mencionamos anteriormente, se podría asignar un nombre al proyecto con espacios en blanco.

Y la ubicación de los archivos del proyecto se configuran presionando en el botón **Browse***..., luego se debe desactivar el check* **Create Main Class** *ya que se trabajará con aplicaciones cortas. Finalmente, presione Finish como lo muestra la Fig. 2.9.*

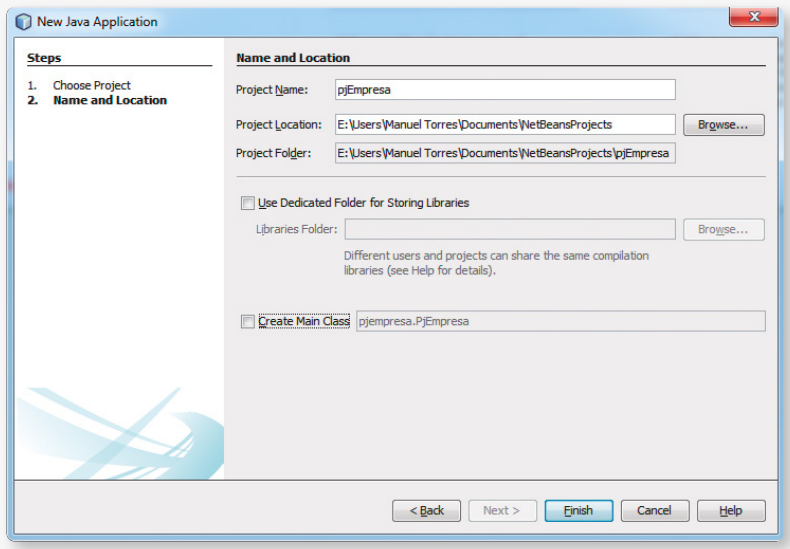


Fig. 2.9

PASO 3:

Hasta el momento el panel de proyectos debe mostrarse como la Fig. 2.10 en la cual NetBeans divide al proyecto en dos carpetas. La primera llamada Source Packages, en la cual se implementarán los paquetes del proyecto como pFormularios(Formularios del proyecto), pImagenes(Imágenes usadas en la aplicación), pClases(Clases que compone el proyecto). Y la carpeta Libreries que contiene los Plugins habilitados para la aplicación; por ejemplo, cuando nos conectemos a una base de datos se debe agregar la librería MySQL JDBC Driver-mysql-connector-java-5.1.6-bin.jar.

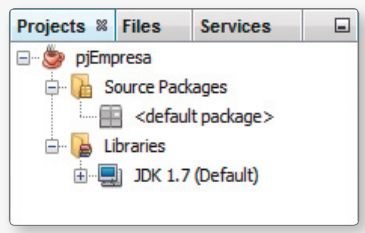


Fig. 2.10

PASO 4:

En vista que debemos organizar el proyecto en paquetes debemos agregarlos, para esto presione clic derecho sobre la carpeta Source Packages > New > Java Package. Como lo muestra la Fig. 2.11.

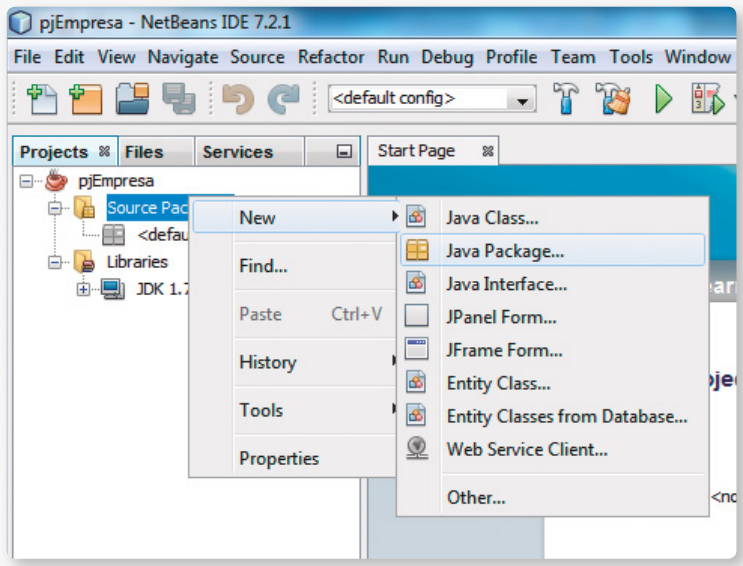


Fig. 2.11

PASO 5:

En vista de que se debe organizar el proyecto en paquetes, agréguelos para esto presionando un clic derecho sobre la carpeta Source Packages > New > Java Package. Asigne el nombre "pFormulario" (ver fig. 2.12).

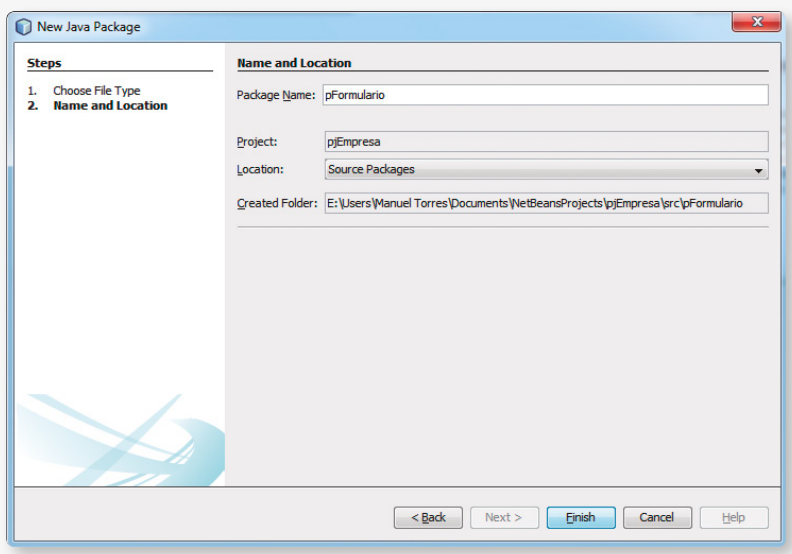


Fig. 2.12

PASO 6:

El proyecto debe quedar como lo muestra la Fig. 2.13 a partir de aquí podremos agregar formularios del proyecto.

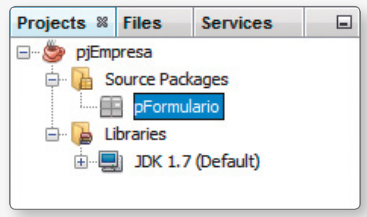


Fig. 2.13

PASO 7:

Para agregar un nuevo formulario al proyecto debemos presionar clic derecho sobre el paquete pFormulario > New > JFrame Form. Como lo muestra la Fig. 2.14.

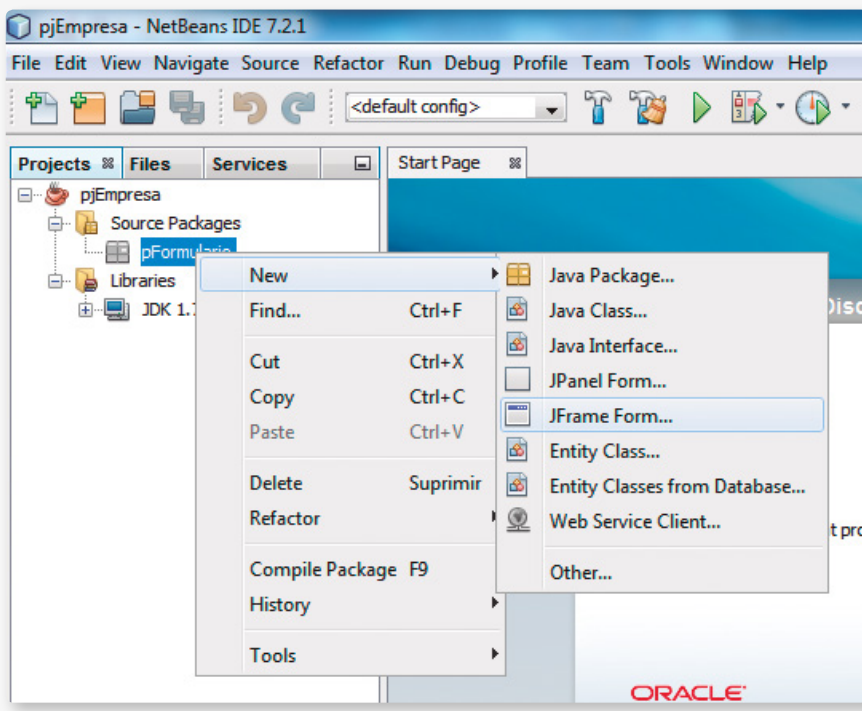


Fig. 2.14

PASO 8:

Asignamos el nombre frmPago como lo muestra la Fig. 2.15, trate de no modificar nada más puesto que ya se configuró todo al iniciar el proyecto. Finalmente, presione Finish.

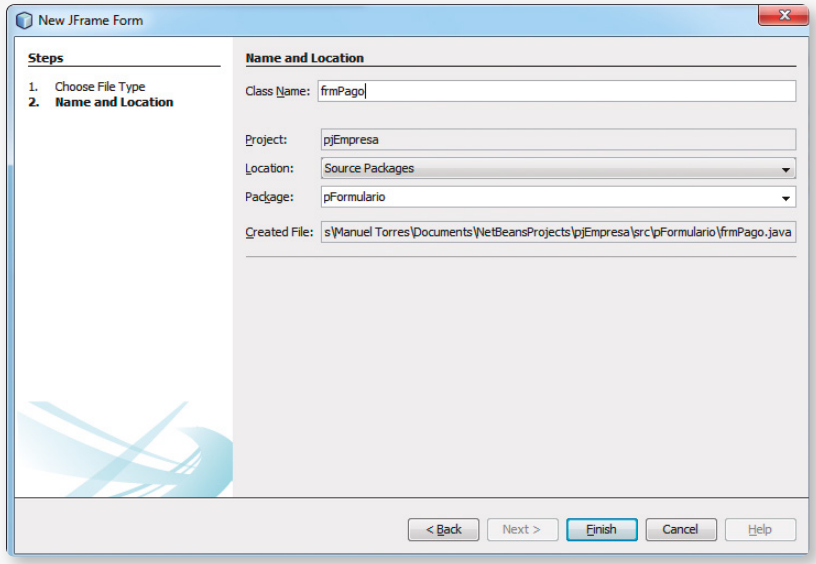


Fig. 2.15

PASO 9:

Finalmente, se muestra el entorno del formulario listo para construir la GUI de la aplicación, note que en el lado derecho aparece la paleta que contiene todos los objetos visuales de Java como Swing Containers que contiene todos los objetos contenedores de Java como ScrollPane, Panel, etc. y Swing Controls que contiene todos los controles para la construcción de una GUI como Label, Button, TextField, etc.

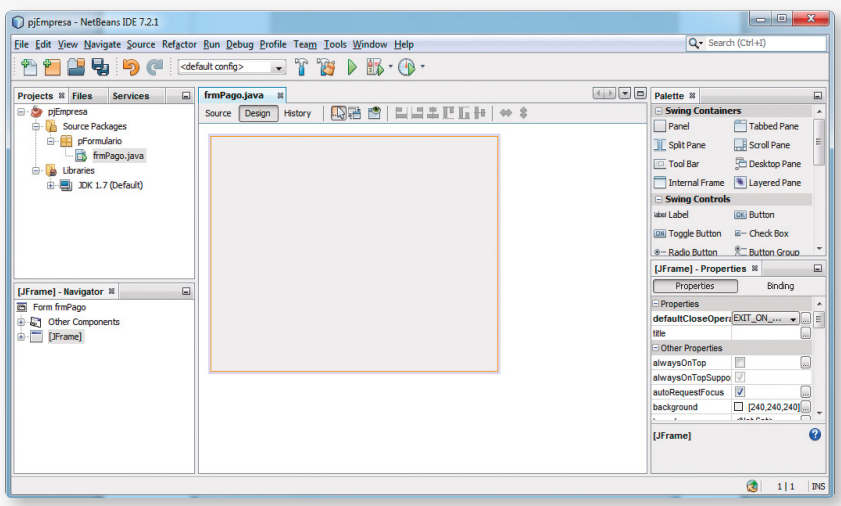


Fig. 2.16

PASO 10:

La Fig. 2.17 muestra la GUI de la aplicación Pago de Empleados.

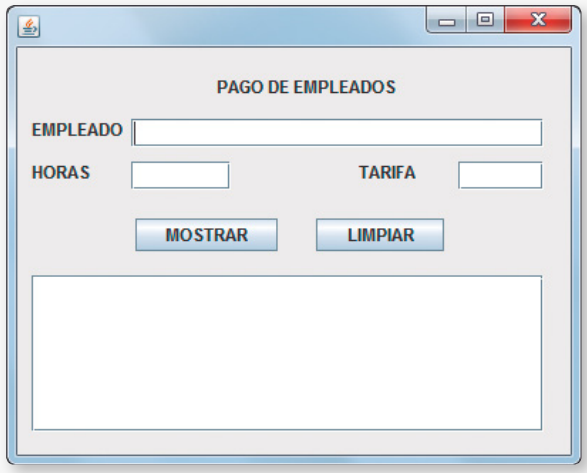


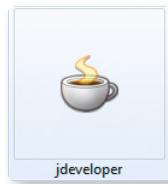
Fig. 2.17

2.21. CREANDO UN PROYECTO CON JDEVELOPER

JDeveloper es un entorno de desarrollo integrado y desarrollado por Oracle Corporation para los lenguajes Java, HTML, XML, SQL, PL/SQL, Javascript, PHP, Oracle ADF, UML y otros. Es un software propietario, pero gratuito desde 2005.

URL de descarga:
http://www.oracle.com/technetwork/developer-tools/jdev/downloads/index.html

Acceso a la aplicación: tenga en cuenta que JDeveloper no se instala.



CREANDO UNA APLICACIÓN CON JDEVELOPER

PASO 1:

File > New seleccionamos la categoría **General** > **Applications** > **Java Desktop Application,** *como se muestra en la Fig. 2.18. Finalmente presione Ok.*

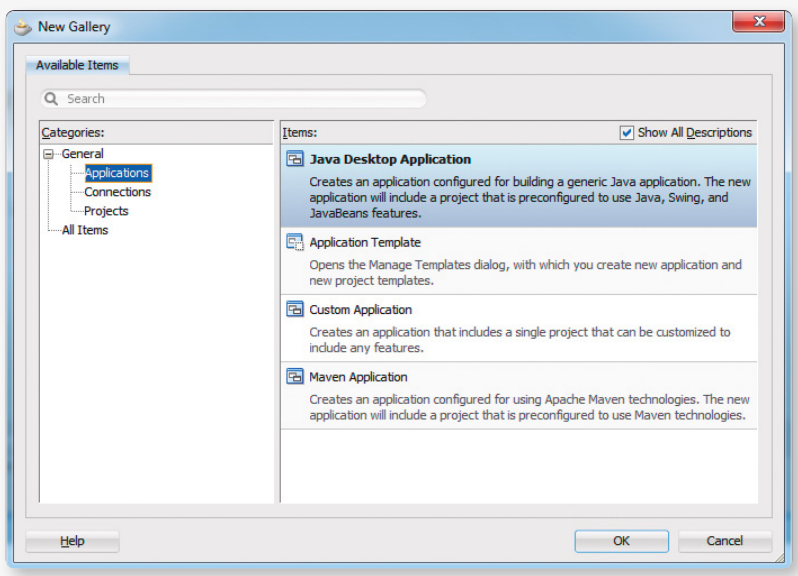


Fig. 2.18

PASO 2:

Luego asignamos un nombre a la aplicación y una ubicación a partir del botón Browse… como lo muestra la Fig. 2.19. Luego presione Next>.

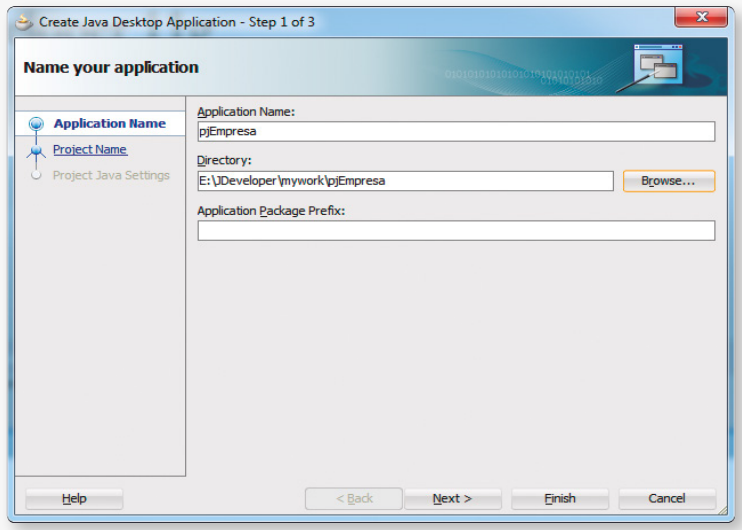


Fig. 2.19

PASO 4:

Desde el navegador de la aplicación podemos ver el proyecto inicial llamado Client como lo muestra la Fig. 2.20.

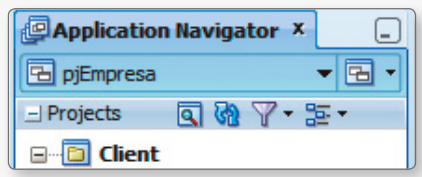


Fig. 2.20

PASO 5:

Ahora agregaremos un Frame al proyecto, para esto presionaremos clic derecho sobre el proyecto inicial Client > **New** >**Client Tier** > **Swing AWT** > **Frame** *como lo muestra la Fig. 2.21*

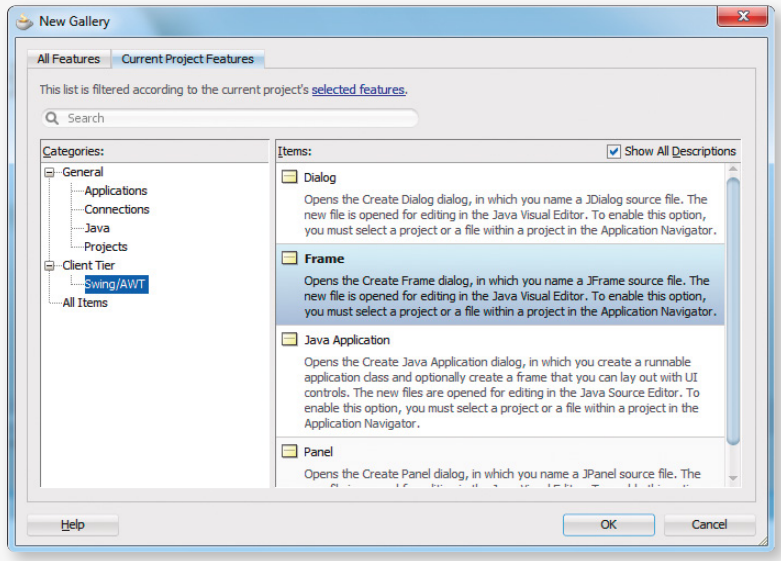


Fig. 2.21

PASO 6:

Luego se le asigna un nombre adecuado al Frame como por ejemplo frmPago > Ok. Como lo muestra la Fig. 2.23.

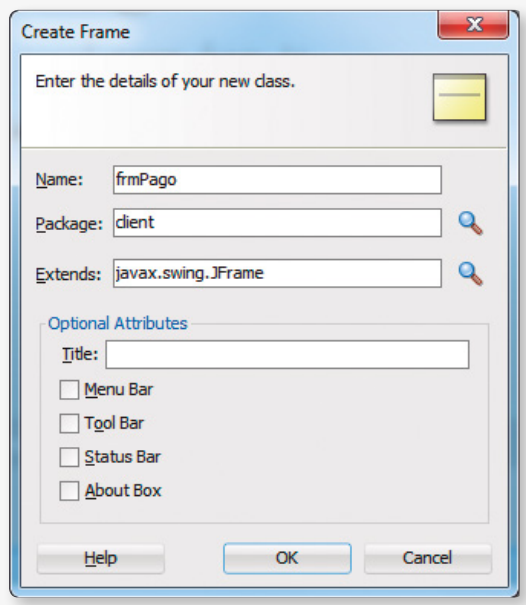


Fig. 2.23

PASO 7:

Finalmente, en entorno de JDeveloper debe quedar de la forma mostrada en la Fig. 2.24.

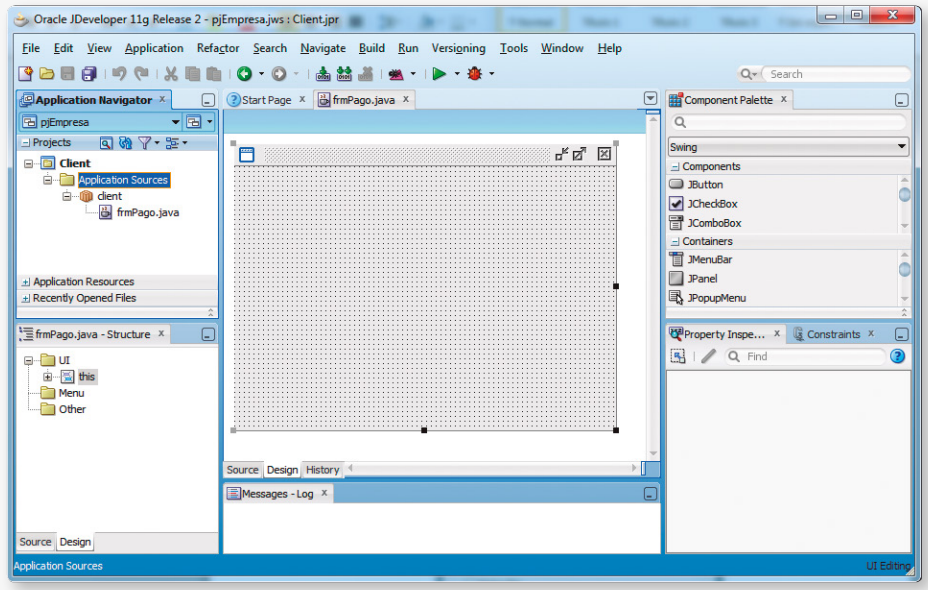


Fig. 2.24

Como notará JDeveloper cuenta con paletas muy parecidas a las mostradas en NetBeans. Solo nos queda diseñar y colocar el código de programación.

CAP.

3

Clases de swing

CAPACIDAD:

- Reconocer las clases que componen la colección Swing.
- Permitir construir aplicaciones básicas con diferentes IDE como JCreator, NetBeans y JDeveloper.

CONTENIDO:

3.1. INTRODUCCIÓN

Hoy en día un usuario está acostumbrado a interactuar con los programas de manera sencilla sin tanto tecnicismo que permita manipular una aplicación. Esto exige contar con interfaces gráficas de usuario (GUI) sencillas, claras y robustas. Las GUI es capaz de interactuar con el usuario haciendo que este se desplace en forma intuitiva sin perder tiempo en estudiar lo que tiene que hacer en la aplicación.
Siendo un factor importante para el usuario la interacción con la aplicación, las GUI usan un conjunto de imágenes y objetos gráficos de tipo AWT o Swing que representan la información de un problema. Esto evoluciona desde la programación en línea de comandos a la modelación visual.

Los objetos GUI por naturaleza son orientados a objetos; por lo tanto, la forma de interactuar con el usuario está sujeta a los eventos que puedan ocurrir en esta actividad llamada "eventos".

```
C:\JAVA_BASICO>java ejem08.calculadora
operando : 78
operador : /
operando : 25
resultado= 3.12
operador : +
operando : 12
resultado= 15.12
operador : -x
resultado= -15.12
operador :

C:\JAVA_BASICO>
```

Fig. 3.1

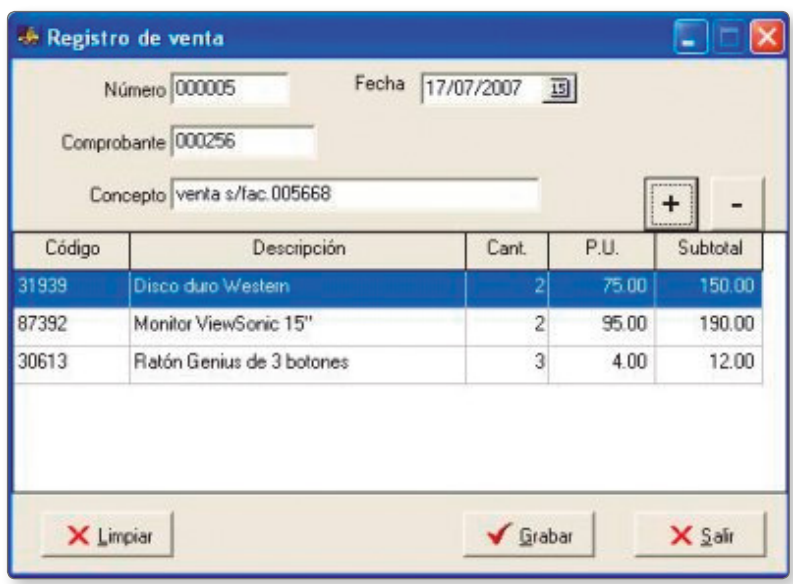
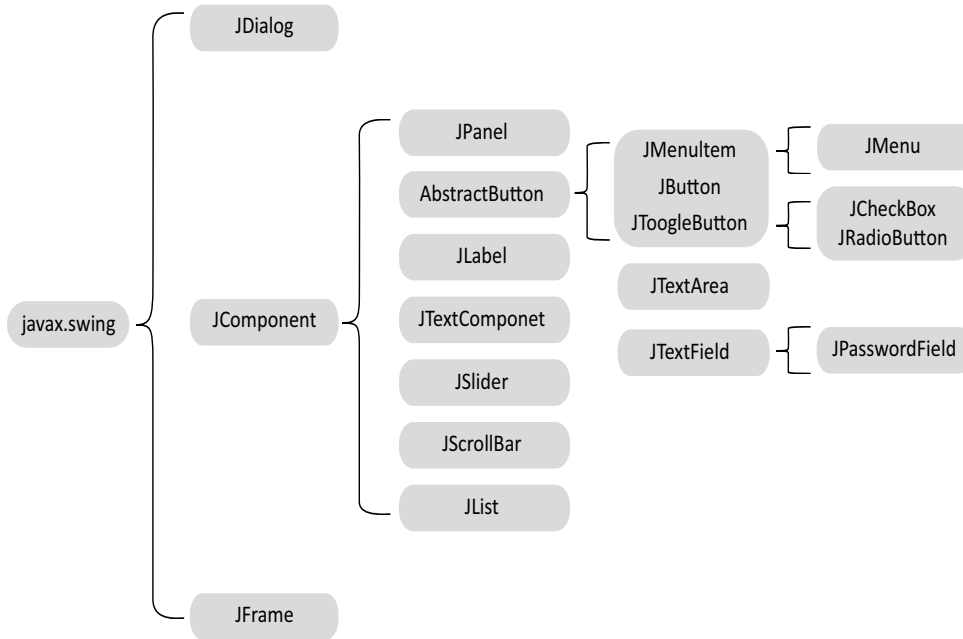


Fig. 3.2

3.2. Clases Swing

La programación basada en GUI determina el uso de un conjunto de clases que serán usadas en una aplicación Java y que desde allí se llamarán componentes u objetos. Java propone un paquete llamado AWT (*Abstract Windows ToolKit*). El AWT 1.0 ofrecía un aspecto poco atractivo para las aplicaciones Java, además de ofrecer muy pocos elementos. Luego se lanza el AWT 1.1 que seguía ofreciendo algunas mejoras, pero no las suficientes para los usuarios Java, este problema se resolvió cuando se dio la versión Java 1.2 al ofrecer el paquete Swing, el cual muestra una mejora en el diseño, tan esperada por los usuarios. Swing no reemplaza al AWT ya que Swing se compone de AWT siendo una capa de esta.

Librería: Javax.Swing.*

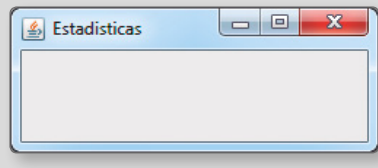


3.3. La clase JFrame

Clase que permite crear un objeto tipo contenedor.

Métodos Constructores:

JFrame()	Construye un objeto de tipo JFrame vacio. Ejm: `JFrame frmEstadisticas = new JFrame();`

JFRAME(STRING TEXTO)	Construye un objeto de tipo JFrame mostrando un texto predeterminado. Ejm: `JFrame frmEstadisticas = new JFrame("Estadisticas");` 

Métodos:

GETCONTENTPANE()	Permite agregar un objeto al control JFrame. Ejm: `frmEstadisticas.getContentPane().add(frmEstadisticas);`

SETBOUNDS(X,Y, ANCHO,ALTO)	Permite posicionar el objeto JFrame dentro del control JFrame.
	
	Ejm:
	
	`frmEstadisticas.setBounds(15, 15, 90, 23);`
	
	o en su defecto usar:
	
	`frmEstadisticas.setLocation(15,15);`
	`frmEstadisticas.setSize(90,23);`

SETLAYOUT()	Permite determinar la forma de administrar los elementos contenidos dentro del control JFrame. `frmEstadisticas.setLayout(null);`
SETTITLE("TITULO")	Permite asignar un titulo al control JFrame. Ejm: `frmEstadisticas.setTitle(" ESTADISTICAS ");`

SETVISIBLE()	Permite mostrar u ocultar el control JFrame.
	
	Ejm:
	
	`Mostrar > frmEstadisticas.setVisible(True);`
	`Ocultar > frmEstadisticas.setVisible(False);`

Opciones:

- Declarar un objeto JFrame > JFrame frmEstadisticas;
- Crear un objeto de tipo JFrame > frmEstadisticas = new JFrame();

Vea, a continuación, el script para implementar un JFrame de Estadísticas:

```
//1.
JLabel lblPorcentaje = new JLabel("Porcentaje");
lblPorcentaje.setBounds(15, 39, 120, 23);

//2.
JFrame frmEstadisticas = new JFrame();
frmEstadisticas.setTitle("Estadisticas");
frmEstadisticas.setLayout(null);
frmEstadisticas.setBounds(10, 10, 200, 100);
frmEstadisticas.setVisible(true);
frmEstadisticas.getContentPane().add(lblPorcentaje);
add(frmEstadisticas);
```

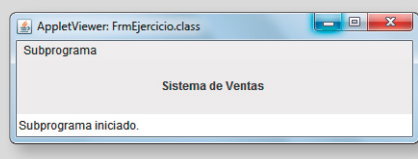
En el punto uno se implementa el objeto lblPorcentaje que será un objeto contenido dentro del JFrame Estadisticas, la posición y tamaño establecido a este objeto será representado cuando se agregue al control JFrame.

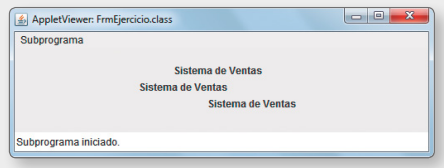
En el punto dos se implementa el objeto frame frmEstadisticas, con setTitle se le asigna un título al control, con setLayout se habilita la distribución manual dentro del JFrame Estadísticas, con setVisible hacemos que el JFrame se visualice; este último es obligatorio colocarlo. Luego se agrega el control lblPorcentaje al JFrame Estadisticas. Finalmente, el panel es agregado al JFrame genérico.

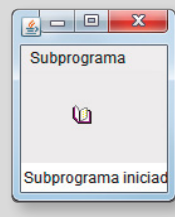
3.4. La clase JLabel

Clase que permite incorporar un control de texto estático no modificable, el cual permite mostrar información de una sola línea de texto o una imagen.

Métodos Constructores:

JLabel()	Construye una etiqueta vacia por lo tanto no muestra texto alguno. Ejm: `JLabel lblTitulo = new JLabel();`
JLabel(String texto)	Construye una etiqueta mostrando un texto. Ejm: `JLabel lblTitulo = new JLabel("Sistema de Ventas");` 

JLABEL(STRING TEXTO, INT ALINEACION)	Construye una etiqueta mostrando un texto con una alineacion especificada.
	
	Ejm:
	
	`JLabel lblTitulo = new JLabel("Sistema de Ventas", JLabel.CENTER);`
	`JLabel lblTitulo = new JLabel("Sistema de Ventas", JLabel.LEFT);`
	`JLabel lblTitulo = new JLabel("Sistema de Ventas", JLabel.RIGHT);`
	
	 

JLABEL(ICON IMAGEN)	Construye una etiqueta mostrando en su contenido una imagen. Ejm: `JLabel lblImagen=new JLabel(new ImageIcon("book1.gif"));`  Hay que tener en cuenta que el archivo de Imagen debe encontrarse en la carpeta Class de la aplicación
JLABEL(ICON IMAGEN, INT ALINEACION)	Construye una etiqueta mostrando en su contenido una imagen con una alineacion especificada. Ejm: `JLabel lblImagen = new JLabel(new ImageIcon("book1.gif"), JLabel.CENTER);`

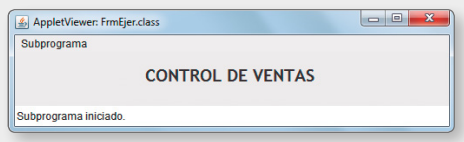
Métodos:

GETTEXT()	Permite devolver el contenido del objeto JLabel.
	
	Ejm: Si el objeto lblN tiene un valor mostrado de 1000 y se necesita capturar dicho valor.
	
	`int numero=Integer.parseInt(lblN.getText());`
	`JOptionPane.showMessageDialog(null, "El valor es: "+numero);`
	
	Resp: 1000

SETBOUNDS(X,Y,ANCHO,ALTO)	Permite posicionar el objeto JLabel dentro del control JFrame.
	
	Ejm:
	
	`lblTitulo.setBounds(15, 15, 90, 23);`
	
	`o en su defecto usar:`
	
	`lblTitulo.setLocation(15,15);`
	`lblTitulo.setSize(90,23);`

SETICON(ICON IMAGEN)	Permite asignar una imagen al control JLabel.
	
	Ejm:
	
	`ImageIcon imagen = new ImageIcon("book1.gif");`
	`LblImagen.setIcon(imagen);`

SETTEXT("MENSAJE")	Permite asignar un texto al control JLabel. Ejm: `lblTitulo.setText("CONTROL DE VENTAS");`

SETFONT()	Permite asignar un tipo de letra, estilo y tamaño al texto mostrado en el control JLabel.
	
	Ejm:
	
	`lblTitulo = new JLabel("CONTROL DE VENTAS");`
	`lblTitulo.setFont(new Font("Trebuchet MS",`
	`Font.BOLD, 20));`
	`lblTitulo.setBounds(150, 15, 250, 23);`
	`getContentPane().add(lblTitulo);`
	
	Resp:
	 

SETFOREGROUND()	Permite asignar un color al control JLabel. Ejm: Asignar el color azul al texto mostrado en lblTitulo `lblTitulo.setForeground(new Color(00,00,255));`

SETVISIBLE()	Permite mostrar u ocultar el control JLabel.
	
	Ejm:
	
	`Mostrar > lblTitulo.setVisible(True);`
	`Ocultar > lblTitulo.setVisible(False);`

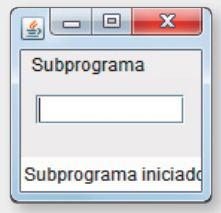
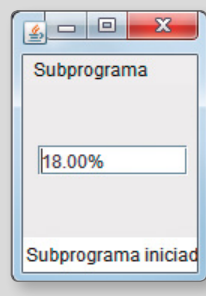
Opciones:

- Declarar un objeto JLabel > JLabel lblTitulo;
- Crear un objeto de tipo JLabel > lblTitulo = new JLabel();

3.5. LA CLASE JTEXTFIELD

Clase que permite incorporar un control de campo de texto en una sola línea, la cual permite ingresar un valor haciendo interactuar al usuario con la aplicación.

Métodos Constructores:

JTEXTFIELD()	Construye un objeto JTextField vacio. Ejm: `JTextField txtIVA = new JTextField();` 
JTEXTFIELD(STRING TEXTO)	Construye un objeto JtextField con un texto predeterminado. Ejm: `JTextField txtIVA = new JTextField("18.00%");` 

Métodos:

ADDACTIONLISTENER()

Permite agregar una accion de evento al ActionListener.

Ejm: Mostrar un mensaje al presionar enter sobre el cuadro de texto:

```
JTextField txtHoras = new JTextField();
txtHoras.setLocation(10,10);
txtHoras.setSize(100,20);
txtHoras.addActionListener(this);
add(txtHoras);

public void actionPerformed( ActionEvent e ){
  if( e.getSource().equals(txtHoras)){
    JOptionPane.showMessageDialog(null,
         "Valor ingresado es: "+
         Integer.parseInt(txtHoras.getText()));
  }
}
```

Debe tener en cuenta que ActionListener de la clase debe estar habilitada de la siguiente forma:

```
public class frmEjercicio
                extends JApplet
                implements ActionListener{
```

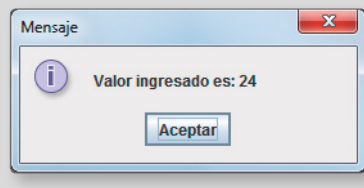
GETTEXT()

Permite devolver el contenido del objeto JTextField.

Ejm: Si el objeto txtHoras tiene un valor mostrado de 24 y se necesita capturar dicho valor.

```
int hora=Integer.parseInt(txtHoras.getText());
JOptionPane.showMessageDialog(null,
                              "El valor es: "+hora);
```

Resp:



SETHORIZONTALALIGNMENT()

Permite alinear el texto contenido dentro del control JTextField.

Ejm:

```
txtHoras.setHorizontalAlignment(JTextField.CENTER);
txtHoras.setHorizontalAlignment(JTextField.LEFT);
txtHoras.setHorizontalAlignment(JTextField.RIGHT);
```

SETBOUNDS(X,Y,ANCHO,ALTO)

Permite posicionar el objeto JTextField dentro del control JFrame.

Ejm:

```
txtHoras.setBounds(15, 15, 90, 23);

o en su defecto usar:

txtHoras.setLocation(15,15);
txtHoras.setSize(90,23);
```

setEditable	Permite bloquear la edicion de texto contenido en un control JTextField. Ejm: `txtHoras.setEditable(false);`
setText(String texto)	Permite asignar un texto al control JTextField. Ejm: `txtHoras.setText("24");`
setFont()	Permite asignar un tipo de letra, estilo y tamaño al texto mostrado en el control JTextField. Ejm: `txtHoras.setFont(new Font("Trebuchet MS", Font.BOLD, 20));`
setForeground()	Permite asignar un color al control JTextField. Ejm: Asignar el color rojo al texto mostrado en txtHoras `txtHoras.setForeground(new Color(255,00,00));`

setVisible()	Permite mostrar u ocultar el control JTextField.
	
	Ejm:
	
	`Mostrar > txtHoras.setVisible(True);`
	`Ocultar > txtHoras.setVisible(False);`

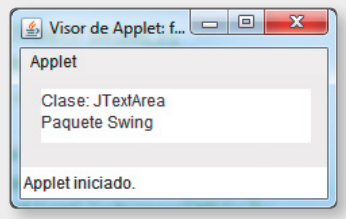
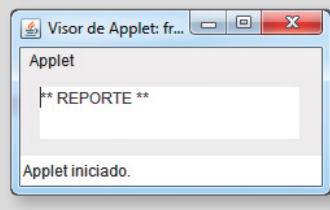
Opciones:

- Declarar un objeto JTextField > JTextField txtHoras;
- Crear un objeto de tipo JtextField > txtHoras = new JTextField();

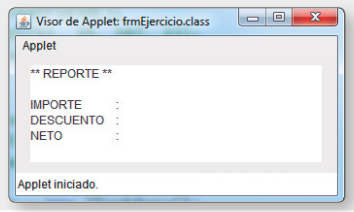
3.6. La clase JTextArea

Clase que permite incorporar un control de campo de texto de líneas múltiples, las cuales permiten ingresar muchos valores textuales dentro del mismo control, su uso se relaciona a impresiones de respuestas o informes.

Métodos Constructores:

JTextArea()	Construye un objeto JTextArea vacio sin bordes ni barras de desplazamiento. Ejm: `JTextArea txtReporte = new JTextArea();` 
JTextArea(String texto)	Construye un objeto JTextArea con un texto predeterminado. Ejm: `JTextArea txtReporte = new JTextArea("** REPORTE **");` 

Metodos:

APPEND()	Permite agregar un texto debajo de la linea actual.
	
	Ejm:
	
	`txtReporte.append("** REPORTE **");`
	`txtReporte.append("\n");`
	`txtReporte.append("\nIMPORTE :");`
	`txtReporte.append("\nDESCUENTO:");`
	`txtReporte.append("\nNETO :");`
	
	 

GETTEXT()	Permite devolver el contenido del objeto JTextArea. Ejm: `String Mensaje=txtReporte.getText());`

SETBOUNDS(X,Y,ANCHO,ALTO)	Permite posicionar el objeto JTextArea dentro del control JFrame.
	
	Ejm:
	
	`txtReporte.setBounds(15, 10, 300, 100);`
	
	o en su defecto usar:
	
	`txtReporte.setLocation(15, 10);`
	`txtReporte.setSize(300, 100);`

SETTEXT(STRING TEXTO)	Permite asignar un texto al control JTextArea. Ejm: `txtReporte.setText("** REPORTE **");` Tiene que considerar que este método imprime en la primera línea del control para una segunda línea de texto tiene que usar el método append.

SETFONT()	Permite asignar un tipo de letra, estilo y tamaño al texto mostrado en el control JTextArea.
	
	Ejm:
	
	`txtReporte.setFont(new Font("Trebuchet MS",`
	`Font.BOLD, 20));`

SETBACKGROUND()	Permite asignar un color al fondo del control JTextArea. Ejm: Asignar color rojo al fondo `txtReporte.setBackground(new Color(255,00,00));`

setForeground()	CPermite asignar un color de texto al control JTextArea. Ejm: Asignar el color blanco al texto mostrado en txtReporte `txtReporte.setForeground(new Color(255,255,255));`

setVisible()	Permite mostrar u ocultar el control J TextArea.
	
	Ejm:
	
	`Mostrar > txtReporte.setVisible(True);`
	`Ocultar > txtReporte.setVisible(False);`

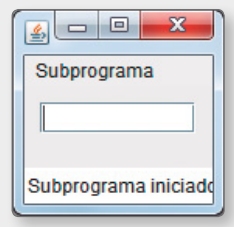
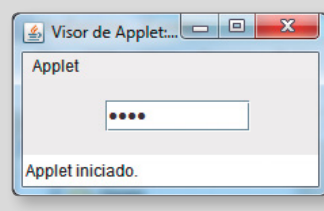
Opciones:

- Declarar un objeto JTextArea > JTextArea txtReporte;
- Crear un objeto de tipo JTextArea > txtReporte = new JTextArea();

3.7. La clase JPasswordField

Clase que permite incorporar un control de campo de texto tipo password, es decir, los caracteres ingresados se enmascaran mostrando símbolos.

Métodos Constructores:

JPasswordField()	Construye un objeto JPasswordField vacio. Ejm: `JPasswordField txtClave = new JPasswordField();` 
JTextField(String texto)	Construye un objeto JPasswordField con un texto predeterminado. Ejm: `JPasswordField txtClave = new JPasswordField("1234");` 

Métodos:

ADDACTIONLISTENER()	Permite agregar una accion de evento al ActionListener.
	
	Ejm: Mostrar la clave ingresada al presionar enter sobre el campo de password:
	
	`JPasswordField txtClave = new JPasswordField();`
	`txtClave.setBounds(61, 15, 107, 23);`
	`txtClave.addActionListener(this);`
	`add(txtClave);`
	
	`public void actionPerformed( ActionEvent e ){`
	`  if( e.getSource().equals(txtClave)){`
	`  JOptionPane.showMessageDialog(null,`
	`  "La clave es: "+ txtClave.getText());`
	`  }`
	`}`
	
	Debe tener en cuenta que ActionListener de la clase debe estar habilitada de la siguiente forma:
	
	`public class frmEjercicio`
	`  extends JApplet`
	`  implements ActionListener{`

GETTEXT()	Permite devolver el contenido del objeto JPasswordField. Ejm: `String clave = txtClave.getText();`

SETBOUNDS(X,Y,ANCHO,ALTO)	Permite posicionar el objeto JPasswordField dentro del control JFrame.
	
	Ejm:
	
	`txtClave.setBounds(61, 15, 107, 23);`
	
	o en su defecto usar:
	
	`txtClave.setLocation(61,15);`
	`txtClave.setSize(107,23);`

SETECHOCHAR	Permite establecer un carácter al control JPasswordField por defecto es asterisco “*”. Ejm: `txtClave.setEchoChar("*");`
SETTEXT(STRING TEXTO)	Permite asignar un texto al control JPasswordField. Ejm: `txtClave.setText("1234");`

SETFONT()	Permite asignar un tipo de letra, estilo y tamaño al texto mostrado en el control JPasswordField.
	
	Ejm:
	
	`txtClave.setFont(new Font("Trebuchet MS",`
	`  Font.BOLD, 20));`

setVisible()	Permite mostrar u ocultar el control JPasswordField.
	
	Ejm:
	
	`Mostrar > txtClave.setVisible(True);`
	`Ocultar > txtClave.setVisible(False);`

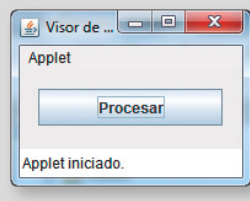
Opciones:

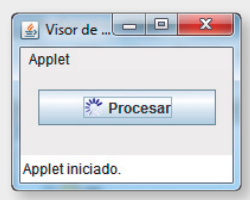
- Declarar un objeto JPasswordField > JPasswordField txtClave;
- Crear un objeto de tipo JPasswordField > txtClave = new JPasswordField();

3.8. La clase JButton

Clase que permite incorporar un control de tipo button, es decir, el usuario podrá ejecutar un proceso mediante el uso de estos botones; hay que tener en cuenta que los botones deben estar incluidos dentro de la lista de eventos ActionListener de Java.

Métodos Constructores:

JButton()	Construye un objeto JButton vacio. Ejm: `btnProcesar = new JButton();`
JButton(String texto)	Construye un objeto JButton con un texto predeterminado. Ejm: `btnProcesar = new JButton("Procesar");` 

JButton(String texto,Icon imagen	Construye un objeto JButton con un texto y una imagen.
	
	Ejm:
	
	`btnProcesar = new JButton("Procesar",`
	`new ImageIcon("procesando.gif"));`
	
	 
	
	Debe tener en cuenta que el archivo imagen debe encontrarse en la carpeta Class del proyecto.

Métodos:

ADDACTIONLISTENER()	Permite agregar una accion de evento al JButton.
	
	Ejm:
	
	`btnProcesar.addActionListener(this);`
	
	`public void actionPerformed( ActionEvent e ){`
	`  if( e.getSource()==btnProcesar){`
	`  JOptionPane.showMessageDialog(null,`
	`  "Usted selecciono Procesar");`
	`  }`
	`}`
	
	Debe tener en cuenta que ActionListener de la clase debe estar habilitada de la siguiente forma:
	
	`public class frmEjercicio`
	`  extends JApplet`
	`  implements ActionListener{`

GETTEXT()	Permite devolver el contenido del objeto JButton. Ejm: `String boton = btnProcesar.getText();`

SETBOUNDS(X,Y,ANCHO,ALTO)	Permite posicionar el objeto JButton dentro del control JFrame.
	
	Ejm:
	
	`btnProcesar.setBounds(15, 15, 150, 30);`
	
	`o en su defecto usar:`
	
	`btnProcesar.setLocation(15,15);`
	`btnProcesar.setSize(150,30);`

SETTEXT(STRING TEXTO)	Permite asignar un texto al control JButton. Ejm: `btnProcesar.setText("Procesar");`

SETFONT()	Permite asignar un tipo de letra, estilo y tamaño al texto mostrado en el control JButton.
	
	Ejm:
	
	`btnProcesar.setFont(new Font("Trebuchet MS",`
	`  Font.BOLD, 20));`

SETVISIBLE()	Permite mostrar u ocultar el control JButton.
	
	Ejm:
	
	`Mostrar > btnProcesar.setVisible(True);`
	`Ocultar > btnProcesar.setVisible(False);`

Opciones:

- Declarar un objeto JButton > JButton btnProcesar;
- Crear un objeto de tipo JButton > btnProcesar = new JButton();

3.9. La clase JPanel

Clase que permite crear un objeto contenedor haciendo una distribución controlada de los controles agregados al JFrame.

Métodos Constructores:

JPanel()	Construye un objeto JPanel vacio. Ejm: `pCriterios = new JPanel();`

Métodos:

add()	Permite agregar un objeto dentro del control JPanel. Ejm: `pCriterios.add(chkPrimaria)`
disable()	Deshabilita el acceso al control JPanel. Ejm: `pCriterios.disable();`
enable()	Habilita el acceso al control JPanel. Ejm: `pCriterios.enable();`
setBackGround()	Permite establecer un color de fondo al control JPanel. Ejm: Establecer el fondo del panel de color rojo. `pCriterios.setBackground(new Color(255,0,0));`

setBorder()	Permite establecer un tipo de borde al JPanel.
	Ejm: Asignaremos un titulo al JPanel
	`pGrado.setBorder(BorderFactory.`
	`createTitledBorder("Grado de Instrucción"));`

SETBOUNDS(X,Y, ANCHO,ALTO)	Permite posicionar el objeto JPanel dentro del control JFrame.
	
	Ejm:
	
	`pCriterios.setBounds(0,0,400,50);`
	
	o en su defecto usar:
	
	`pCriterios.setLocation(15,15);`
	`pCriterios.setSize(150,30);`

SETFONT()	Permite asignar un tipo de letra, estilo y tamaño al texto mostrado en el control JPanel. Ejm: `pCriterios.setFont(new Font("Trebuchet MS", Font.BOLD, 20));`
SETLAYOUT()	Permite establecer el control de los objetos contenidos en el control JPanel. Ejm: `pCriterios.setLayout(null);`
SETTOOLTIPTEXT	Permite asignar un mensaje al posicionar el puntero del mouse por encima del control JPanel. Ejm: `pCriterios.setToolTipText("Seleccione un criterio..!!");`

SETVISIBLE()	Permite mostrar u ocultar el control JPanel.
	
	Ejm:
	
	`Mostrar > pCriterios.setVisible(True);`
	`Ocultar > pCriterios.setVisible(False);`

Opciones:

- Declarar un objeto JPanel > JPanel pCriterios;
- Crear un objeto de tipo JPanel > pCriterios = new JPanel();

3.10. La clase JCheckBox

Clase que permite incorporar un control de tipo cuadro de chequeo, en la cual un usuario podrá seleccionar una o más opciones Check.

Métodos Constructores:

JCheckBox()	Construye un objeto JCheckBox vacio. Ejm: `chkPrimaria = new JCheckBox();`
JCheckBox(String texto)	Construye un objeto JCheckBox con un texto predeterminado. Ejm: `chkPrimaria = new JCheckBox("Primaria");`
JCheckBox(String texto, boolean predeterminado)	Construye un objeto JCheckBox con un texto y un estado predeterminado es decir el check estará establecido al iniciar la aplicación. Ejm: `chkPrimaria = new JCheckBox("Primaria", true);`

JCheckBox(String texto, Icon imagen, boolean predertеminado)	Construye un objeto JCheckBox con un texto y una imagen.
	
	Ejm:
	
	`chkPrimaria = new JCheckBox("Primaria",`
	`new ImageIcon("primaria.gif"),true);`
	
	Debe tener en cuenta que el archivo imagen debe encontrarse en la carpeta Class del proyecto.

Métodos:

AddItemListener()	Permite agregar una accion de evento al JCheckBox.
	
	Ejm:
	
	`chkSoltero.addItemListener(this);`
	
	`public void itemStateChanged( ItemEvent e ){`
	`  if(chkPrimaria.isSelected()){`
	`  JOptionPane.showMessageDialog(null,`
	`  "Usted selecciono Primaria..!!");`
	`  }`
	`}`
	
	Debe tener en cuenta que ItemListener de la clase debe estar habilitada de la siguiente forma:
	
	`public class frmEjercicio`
	`extends JApplet`
	`implements ItemListener{`

ISSELECTED()	Permite determinar si el objeto JCheckBox ha sido seleccionado. Ejm: `if (chkPrimaria.isSelected())`

SETBOUNDS(X,Y,ANCHO,ALTO)	Permite posicionar el objeto JCheckBox dentro del control JFrame.
	
	Ejm:
	
	`chkPrimaria.setBounds(15, 15, 150, 30);`
	
	o en su defecto usar:
	
	`chkPrimaria.setLocation(15,15);`
	`chkPrimaria.setSize(150,30);`

SETTEXT(STRING TEXTO)	Permite asignar un texto al control JCheckBox. Ejm: `chkPrimaria.setText("Primaria");`

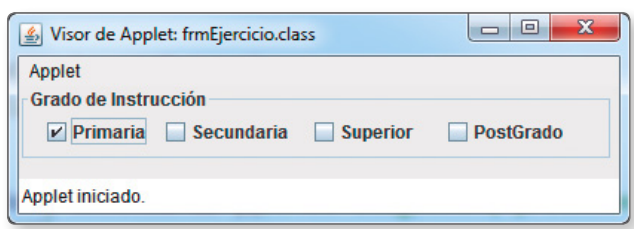
SETFONT()	Permite asignar un tipo de letra, estilo y tamaño al texto mostrado en el control JCheckBox.
	
	Ejm:
	
	`chkPrimaria.setFont(new Font("Trebuchet MS",`
	`Font.BOLD, 20));`

SETVISIBLE()	Permite mostrar u ocultar el control JCheckBox.
	
	Ejm:
	
	`Mostrar > chkPrimaria.setVisible(True);`
	`Ocultar > chkPrimaria.setVisible(False);`

Opciones:

- Declarar un objeto JCheckBox > JCheckBox chkPrimaria;
- Crear un objeto de tipo JCheckBox > chkPrimaria = new JCheckBox();

Veamos el script para implementar el siguiente modelo:



```
//1.
JCheckBox chkPrimaria = new JCheckBox("Primaria",true);
chkPrimaria.setBounds(15, 15, 80, 30);
getContentPane().add(chkPrimaria);

JCheckBox chkSecundaria = new JCheckBox("Secundaria");
chkSecundaria.setBounds(95, 15, 100, 30);
getContentPane().add(chkSecundaria);

JCheckBox chkSuperior = new JCheckBox("Superior");
chkSuperior.setBounds(195, 15, 80, 30);
getContentPane().add(chkSuperior);

JCheckBox chkPostgrado = new JCheckBox("PostGrado");
chkPostgrado.setBounds(285, 15, 100, 30);
getContentPane().add(chkPostgrado);

//2.
JPanel pGrado = new JPanel();
pGrado.setBorder(BorderFactory.createTitledBorder("Grado de Instrucción"));
pGrado.setLayout(null);
pGrado.setBounds(0,0,400,50);
pGrado.add(chkPrimaria);
pGrado.add(chkSecundaria);
pGrado.add(chkSuperior);
pGrado.add(chkPostgrado);
add(pGrado);
```

En el punto uno se implementan los cuadros de chequeos Primaria, Secundaria, Superior y PostGrado haciendo que el check Primaria sea predeterminada; para esto se le adicionó true al crear el objeto.

En el punto dos se implementa un panel que permitirá contener los cuadros de chequeo ya creados, para esto se crea el objeto panel grado (pGrado) asignando un texto a su marco con BorderFactory, además se tiene que anular el control de objetos internos del panel grado con setLayout(null). Finalmente se agregan los objetos chk al panel con el método add.

3.11. La clase JRadioButton

Clase que permite incorporar un control de tipo opción.

Métodos Constructores:

JRadioButton ()	Construye un objeto JRadioButton vacio. Ejm: `rbSoltero = new JRadioButton();`
JRadioButton (String texto)	Construye un objeto JRadioButton con un texto predeterminado. Ejm: `rbSoltero = new JRadioButton ("Soltero");`

JRADIOBUTTON (STRING TEXTO, BOOLEAN PREDETERMINADO)	Construye un objeto JRadioButton con un texto y un estado predeterminado es decir aparecera como seleccionado al ejecutar la aplicación. Ejm: `rbSoltero = new JRadioButton ("Soltero",true);`

JRADIOBUTTON (STRING TEXTO, ICON IMAGEN, BOOLEAN PREDERTEMINADO)	Construye un objeto JRadioButton con un texto, una imagen y un estado.
	
	Ejm:
	
	`rbSoltero = new JRadioButton ("Soltero",`
	`new ImageIcon("soltero.gif"),true);`
	
	Debe tener en cuenta que el archivo imagen debe encontrarse en la carpeta Class del proyecto.

Métodos:

ADDITEMLISTENER()	Permite agregar una accion de evento al JRadioButton.
	
	Ejm:
	
	`rbSoltero.addItemListener(this);`
	
	`public void itemStateChanged( ItemEvent e ){`
	`if(rbSoltero.isSelected()){`
	`JOptionPane.showMessageDialog(null,`
	`"Usted selecciono Soltero..!!");`
	`}`
	`}`
	
	Debe tener en cuenta que ItemListener de la clase debe estar habilitada de la siguiente forma:
	
	`public class frmEjercicio`
	`extends JApplet`
	`implements ItemListener{`

ISSELECTED()	Permite determinar si el objeto JRadioButton ha sido seleccionado. Ejm: `if (rbSoltero.isSelected())`

SETBOUNDS(X,Y, ANCHO,ALTO)	Permite posicionar el objeto JRadioButton dentro del control JFrame.
	
	Ejm:
	
	`rbSoltero.setBounds(15, 15, 150, 30);`
	
	o en su defecto usar:
	
	`rbSoltero.setLocation(15,15);`
	`rbSoltero.setSize(150,30);`

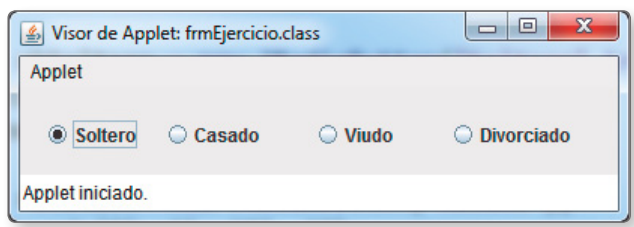
setText(String texto)	Permite asignar un texto al control JRadioButton. Ejm: `rbSoltero.setText("Soltero");`
setFont()	Permite asignar un tipo de letra, estilo y tamaño al texto mostrado en el control JRadioButton. Ejm: `rbSoltero.setFont(new Font("Trebuchet MS", Font.BOLD, 20));`

setVisible()	Permite mostrar u ocultar el control JRadioButton.
	
	Ejm:
	
	`Mostrar > rbSoltero.setVisible(True);`
	`Ocultar > rbSoltero.setVisible(False);`

Opciones:

- Declarar un objeto JRadioButton > JRadioButton rbSoltero;
- Crear un objeto de tipo JRadioButton > rbSoltero = new JRadioButton();

Veamos el script para implementar el siguiente modelo:




```
//1.
JRadioButton rbSoltero = new JRadioButton("Soltero",true);
rbSoltero.setBounds(15, 15, 80, 30);
getContentPane().add(rbSoltero);

JRadioButton rbCasado= new JRadioButton("Casado");
rbCasado.setBounds(95, 15, 100, 30);
getContentPane().add(rbCasado);

JRadioButton rbViudo = new JRadioButton("Viudo");
rbViudo.setBounds(195, 15, 80, 30);
getContentPane().add(rbViudo);

JRadioButton rbDivorciado = new JRadioButton("Divorciado");
rbDivorciado.setBounds(285, 15, 100, 30);
getContentPane().add(rbDivorciado);
```

```
//2.
ButtonGroup bgrEstados=new ButtonGroup();
bgrEstados.add(rbSoltero);
bgrEstados.add(rbCasado);
bgrEstados.add(rbViudo);
bgrEstados.add(rbDivorciado);
add(rbSoltero);
add(rbCasado);
add(rbViudo);
add(rbDivorciado);
```

En el punto uno se implementan los botones de opción para los estados civil soltero, casado, viudo y divorciado, haciendo que rbSoltero sea la radio predeterminada.

En el punto dos se implementa un objeto creado que sea grupo de los estados para que se pueda seleccionar un elemento por grupo, de otra manera todos los botones se activarán; para esto creamos el grupo de botones llamado bgrEstados y le agregamos por medio del método add todos los botones implementados. Finalmente, todos los botones son agregados también al getContentPane() del JFrame.

3.12. CONSTRUIR GUI CON JCREATOR

En el capitulo anterior vimos como crear un espacio de trabajo con JCreator, ahora nos concentraremos en los objetos Swing de Java. En una aplicación Applet desde JCreator debe considerar la posición y el espacio proporcionado para la aplicación, puesto que podemos también configurarlo. A continuación veremos el espacio de trabajo del Applet en JCreator:

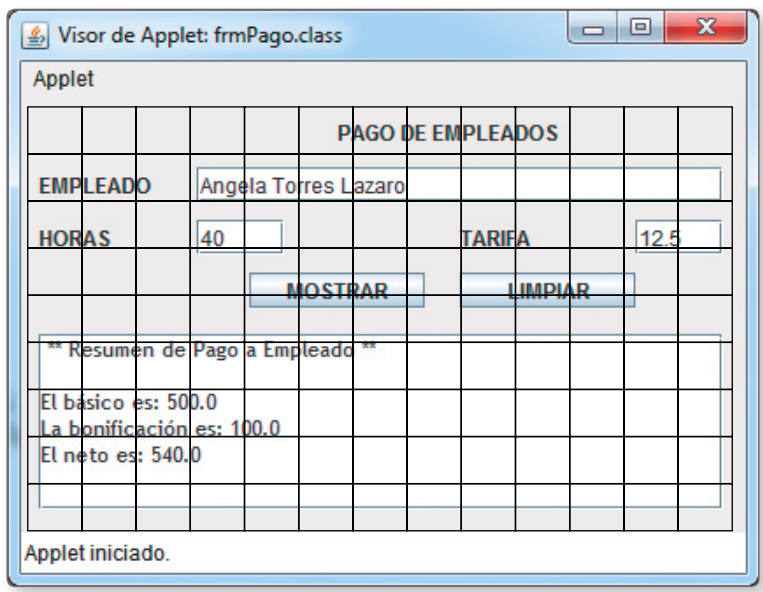


Fig. 3.3

En la Fig. 3.4 se muestra el aspecto que tiene la GUI con respecto al espacio de trabajo dentro del Applet. Consideremos la siguiente distribución solo si la aplicación está configurada a 500x300:

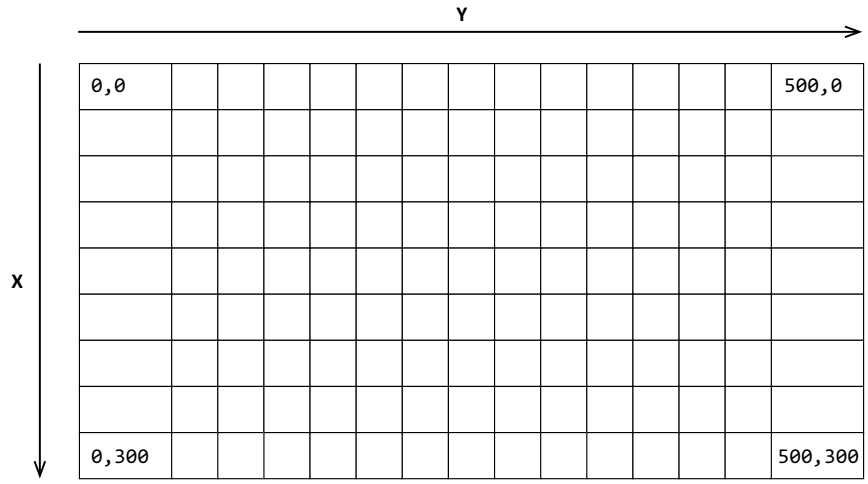


Fig. 3.4

Si consideramos que el ancho es 500px y el alto es 300px, así lo especifica el archivo htm, para ver el script presione doble clic sobre el archivo frmPago.htm:

```
<html>
    <head>
    </head>
    <body bgcolor="000000">
            <center>
                    <applet
                            code    = "frmPago.class"
                            width   = "500"
                            height= "300"
                            >
                    </applet>
            </center>
    </body>
</html>
```

Como vemos la estructura es la misma de una página básica en html lo que lo diferencia es la invocación al applet de Java gracias a la etiqueta <applet> </applet> en la cual se define:

- Code: es el archivo compilado de la aplicación.
- Width: es el ancho expresado en pixeles definido para el applet.
- Height: es el alto expresado en pixeles definido para el applet.

3.13. VISUALIZAR EL APPLET JAVA EN UN BROWSER

Si se necesita visualizar la página web en el mismo JCreator siga los siguientes pasos:

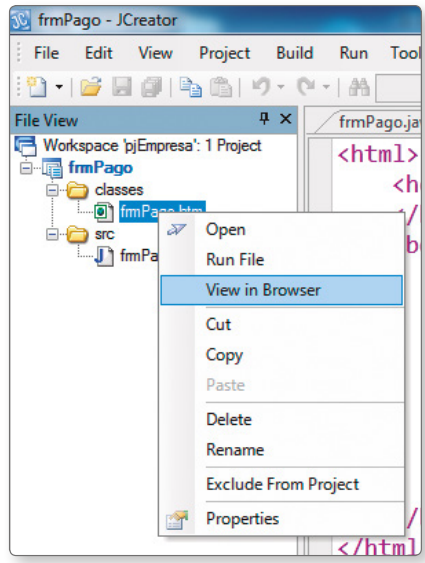


Fig. 3.5

Como lo muestra la Fig. 3.6 se debe presionar clic derecho sobre el archivo htm > View in Browser, el resultado sería:

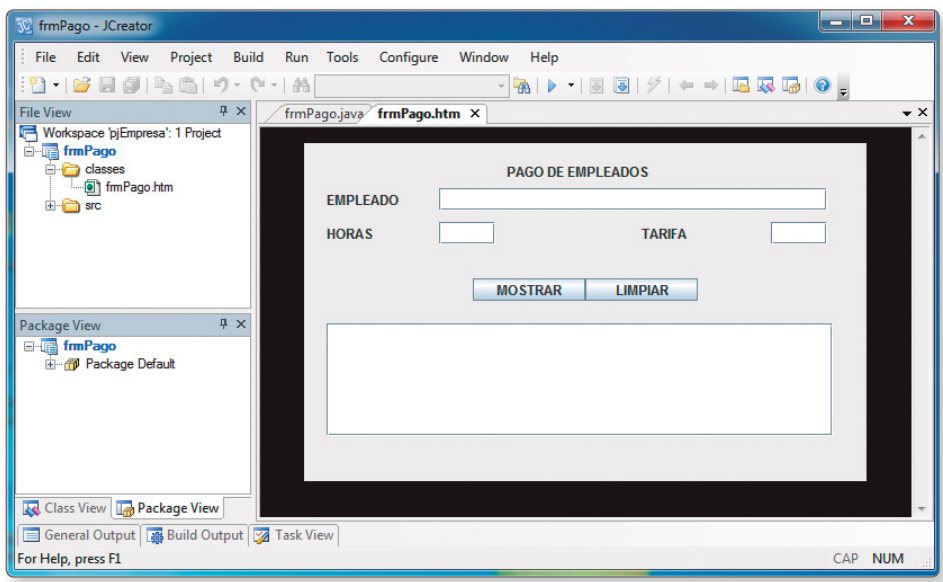


Fig. 3.6

Para visualizar el Applet sin necesidad del IDE JCreator debemos acceder a los archivos creados dentro de la carpeta classes de la aplicación:

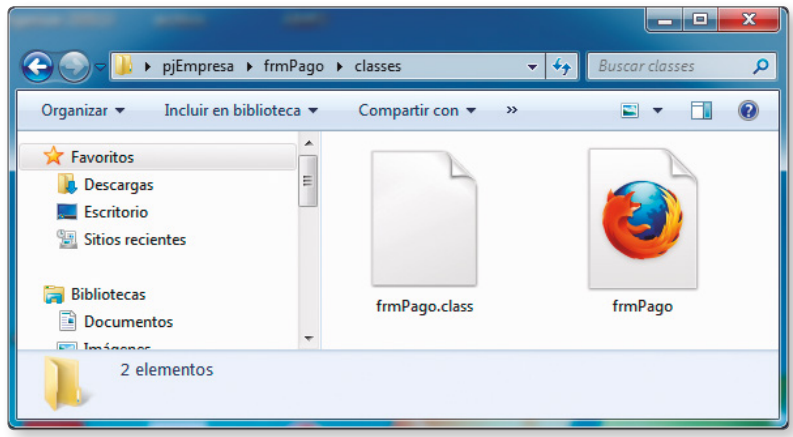


Fig. 3.7

Aquí encontraremos dos archivos:

- frmPago.class: Archivo compilado
- frmPago.htm: Archico Web de la aplicación

Para poder visualizar el applet dentro de una página web presione doble clic sobre el archivo frmPago.htm.

CASO DESARROLLADO 1

Se va a crear una aplicación que permita mostrar el sueldo básico, bonificación y neto de los empleados de una empresa, por lo cual debe ingresar su nombre, horas de trabajo y la tarifa por hora trabajada. La bonificación es el 20% del sueldo básico y al sueldo neto se le debe descontar el 10%.

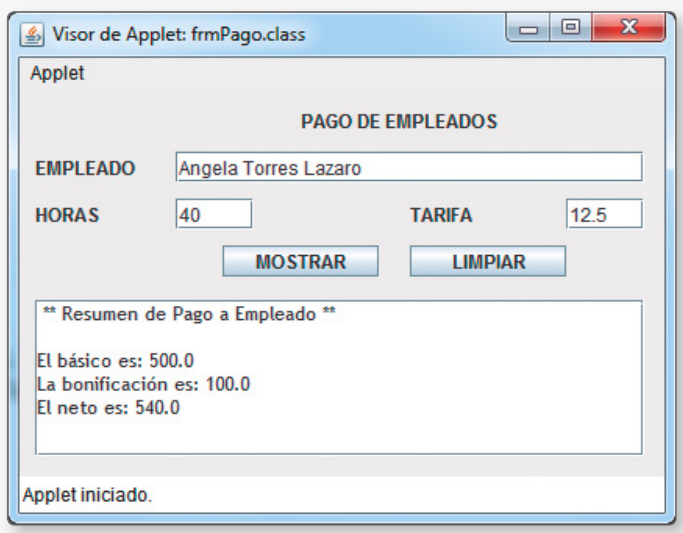


Fig. 3.8

Solución:

PASO 1:

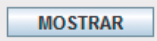
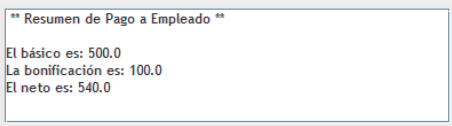
Veamos una lista de los objetos encontrados con sus respectivas Clases.

OBJETO	CLASE SWING
PAGO DE EMPLEADOS	JLabel
TARIFA	
HORAS	
EMPLEADO	
Angela Torres Lazaro	JTextField
40	
12.5	
MOSTRAR	JButton
LIMPIAR	
** Resumen de Pago a Empleado ** El básico es: 500.0 La bonificación es: 100.0 El neto es: 540.0	JtextArea

PASO 2:

Veremos el script para cada objeto de la aplicación.

OBJETO	SCRIPT
PAGO DE EMPLEADOS	*lblTitulo = new JLabel("PAGO DE EMPLEADOS");* *lblTitulo.setBounds(180,15,200,20);* *add(lblTitulo);*
EMPLEADO	*lblEmpleado = new JLabel("EMPLEADO");* *lblEmpleado.setBounds(20,40,200,20);* *add(lblEmpleado);*
HORAS	*lblHoras = new JLabel("HORAS");* *lblHoras.setBounds(20,70,200,20);* *add(lblHoras);*
TARIFA	*lblTarifa = new JLabel("TARIFA");* *lblTarifa.setBounds(300,70,200,20);* *add(lblTarifa);*
Angela Torres Lazaro	*txtEmpleado = new JTextField();* *txtEmpleado.setBounds(120,40,345,20);* *add(txtEmpleado);*

	*txtHoras = new JTextField();* *txtHoras.setBounds(120,70,50,20);* *add(txtHoras);*
	*txtTarifa= new JTextField();* *txtTarifa.setBounds(415,70,50,20);* *add(txtTarifa);*
	*btnMostrar = new JButton("MOSTRAR");* *btnMostrar.setBounds(150,120,100,20);* *btnMostrar.addActionListener(this);* *add(btnMostrar);*
	*btnLimpiar = new JButton("LIMPIAR");* *btnLimpiar.setBounds(250,120,100,20);* *btnLimpiar.addActionListener(this);* *add(btnLimpiar);*
	*txtSalida = new JTextArea();* *txtSalida.setFont(new Font("Trebuchet MS",0,12));* *spDesplaza = new JScrollPane(txtSalida);* *spDesplaza.setBounds(20, 160, 450, 100);*

PASO 3:

Veamos el script completo del applet:

```
//1.
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

//2.
public class frmPago extends JApplet implements ActionListener {

    //3.
    JLabel lblTitulo,lblEmpleado,lblHoras,lblTarifa;
    JTextField txtEmpleado,txtHoras,txtTarifa;
    JButton btnMostrar, btnLimpiar;
    JTextArea txtSalida;
    JScrollPane spDesplaza;

      //4.
    public void init() {
            this.setLayout(null);

            lblTitulo = new JLabel("PAGO DE EMPLEADOS");
            lblTitulo.setBounds(180,15,200,20);
            add(lblTitulo);

            lblEmpleado = new JLabel("EMPLEADO");
            lblEmpleado.setBounds(20,40,200,20);
            add(lblEmpleado);
```

```
        lblHoras = new JLabel("HORAS");
        lblHoras.setBounds(20,70,200,20);
        add(lblHoras);

        lblTarifa = new JLabel("TARIFA");
        lblTarifa.setBounds(300,70,200,20);
        add(lblTarifa);

        txtEmpleado = new JTextField();
        txtEmpleado.setBounds(120,40,345,20);
        add(txtEmpleado);

        txtHoras = new JTextField();
        txtHoras.setBounds(120,70,50,20);
        add(txtHoras);

        txtTarifa= new JTextField();
        txtTarifa.setBounds(415,70,50,20);
        add(txtTarifa);

        btnMostrar = new JButton("MOSTRAR");
        btnMostrar.setBounds(150,120,100,20);
        btnMostrar.addActionListener(this);
        add(btnMostrar);

        btnLimpiar = new JButton("LIMPIAR");
        btnLimpiar.setBounds(250,120,100,20);
        btnLimpiar.addActionListener(this);
        add(btnLimpiar);

        txtSalida = new JTextArea();
        txtSalida.setFont(new Font("Trebuchet MS", 0, 12));

        spDesplaza = new JScrollPane(txtSalida);
        spDesplaza.setBounds(20, 160, 450, 100);
        add(spDesplaza);

    }
```

En el punto uno se importan las librerías necesarias para la aplicación java.awt.event.* que permitan tener acceso a todos los eventos que puedan ocurrir en la aplicación como un clic del mouse o presionar enter desde el teclado.

Java.awt.* permite tener acceso a todas las clases estándar de Java, mientras que javax.swing.* permite tener acceso a todas las clases swing que permitirán construir la GUI de Java.

Estos 3 son los mínimos requeridos para la aplicación, hay que considerar que conforme vamos implementando una aplicación se necesitaran más librerías.

En el punto dos se implementa la clase frmPago que contendrá:

- **extends JApplet**

  Esto indica que la aplicación hereda comportamiento y atributos de la clase JApplet. Se extiende de la clase JApplet porque nosotros así lo definimos al momento de crear el proyecto dentro del espacio de trabajo.

- **ActionListener**

  Es usado para detectar y manejar eventos de acción, mejor dicho cuando se produce una acción sobre un control de la aplicación como por ejemplo:

  - Cuando el usuario presione clic sobre un botón.
  - Cuando el usuario haga doble clic en un elemento de lista.
  - Cuando el usuario presione ENTER en una caja de texto.
  - Cuando el usuario seleccione un elemento de un menú de opciones.

Cuando un ActionListener detecta una acción sobre algún control de la aplicación genera un evento de acción llamado ActionEvent, estos invocan al método actionPerformed(ActionEvent e) que realiza las acciones programadas ante ese evento.

En el punto tres se declaran las variables globales de la aplicación, en este caso declare a todos los controles que contendrá nuestra aplicación.

```
JLabel lblTitulo,lblEmpleado,lblHoras,lblTarifa;
JTextField txtEmpleado,txtHoras,txtTarifa;
JButton btnMostrar, btnLimpiar;
JTextArea txtSalida;
JScrollPane spDesplaza;
```

Se declaró lblTitulo, lblEmpleado, lblhoras y lblTarifa del tipo Jlabel, tenga en cuenta que solo se está declarando mas no creando el objeto de dicha clase.

En el punto cuatro se declara el método void init que permitirá inicializar los valores que caracterizarán a los controles de la aplicación, aquí debemos crear el objeto declarado, asignar una posición en el applet, definir su ActionListener si lo tuviera y agregarlo al contenedor principal.

La instrucción this.setLayout(null) permite que los controles se puedan desplazar a las posiciones especificadas en el método setBounds.

Ahora debemos crear el método que permitirá devolver los resultados esperados en la aplicación:

```
public void actionPerformed(ActionEvent e){
    if (e.getSource()==btnMostrar){
                    //Entrada
                    int horas=Integer.parseInt(txtHoras.getText());
                    double tarifa=Double.parseDouble(txtTarifa.getText());

                    //Proceso
                    double basico = horas * tarifa;
                    double bonificacion = basico * 0.2;
                    double neto = (basico + bonificacion) * 0.9;

                    //Salida
                    txtSalida.setText(" ** Resumen de Pago a Empleado ** ");
                    txtSalida.append("El básico es: "+basico);
                    txtSalida.append("La bonificación es: "+bonificacion);
                    txtSalida.append("El neto es: "+neto);
```

```
            }
        if (e.getSource()==btnLimpiar){
                        txtEmpleado.setText("");
                        txtHoras.setText("");
                        txtTarifa.setText("");
                        txtSalida.setText("");
                        txtEmpleado.requestFocus();
        }
     } //Cierra el método actionPerformed
    } //Cierra la clase frmPago
```

Para ejecutar la aplicación en JCreator:

F7: Compilar la aplicación

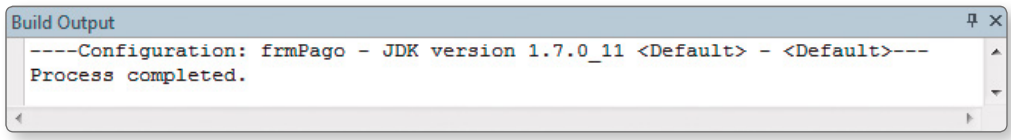


Fig. 3.9

F5: Ejecutar

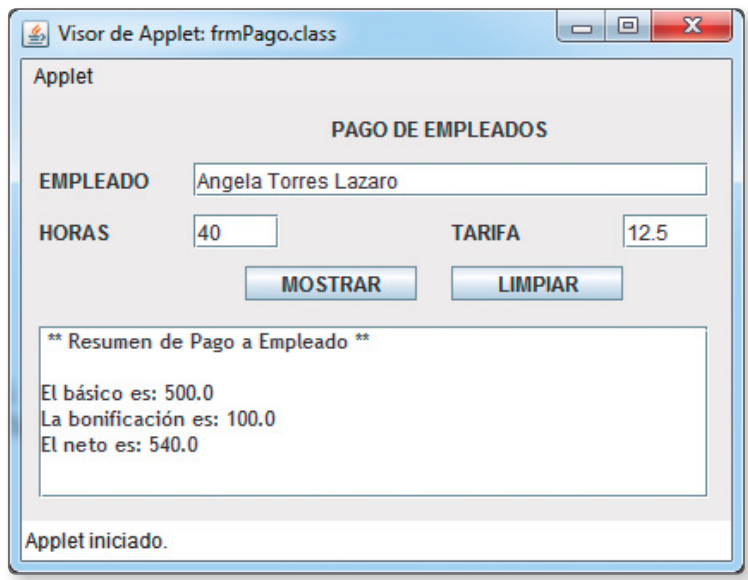


Fig. 3.10

Debe ingresar los valores como lo muestra la Fig. 3.10 y presionar el botón Mostrar.

3.14. CONSTRUIR GUI CON NETBEANS

Creará la misma aplicación mostrada con JCreator, esta vez usando el IDE NetBeans:

Primero debe conocer las paletas que componen el entorno de NetBeans:

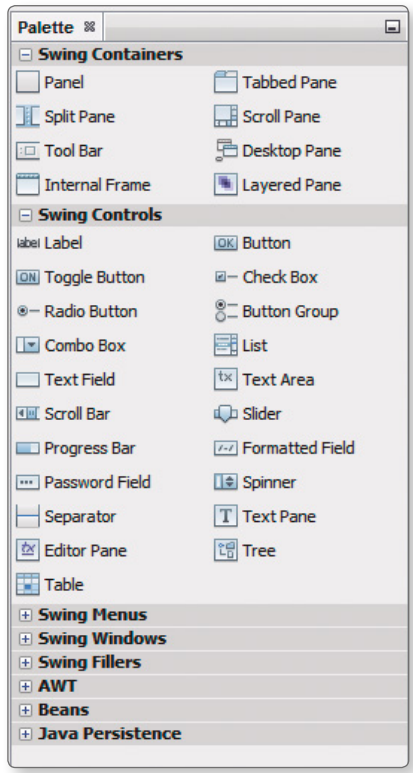


Fig. 3.11

Para visualizar la paleta:

- Ctrl +Shift+8
- Windows > Pallete

Swing Container: se nombrará los principales contenedores:

Panel	Crea un objeto Panel contenedor de más controles.
Scroll Pane	Crea un objeto contenedor para un cuadro de lista (JList) o un área de texto (JTextArea) en la cual permite incluir barras de desplazamiento a dichos controles.
Internal Frame	Crea un objeto tipo Frame hijo, normalmente usado para construir aplicaciones que invocan a otros Frame dentro de la misma aplicación.
Tool Bar	Crea un objeto de barra de herramientas.

Swing Controls:

Label	Permite crear un objeto de la clase JLabel > jLabel1
Button	Permite crear un objeto de la clase JButton > jButton1
Check Box	Permite crear un objeto de la clase JCheckBox > jCheckBox1
Radio Button	Permite crear un objeto de la clase JRadioButton > jRadioButton1
Combo Box	Permite crear un objeto de la clase JComboBox > Item 1
List	Permite crear un objeto de la clase JList > Item 1 Item 2 Item 3 Item 4 Item 5 Para mostrar una barra de desplazamiento se tiene que incluir dentro del control ScrollPane.
Text Field	Permite crear un objeto de la clase JTextField > jTextField1
Text Area	Permite crear un objeto de la clase JTextArea >
Password Field	Permite crear un objeto de la clase JPasswordField>
Table	Permite crear un objeto de la clase JTable > Title 1 Title 2 Title 3 Title 4

Por cada control colocado dentro del contenedor Frame se tiene un conjunto de propiedades, veamos dicha paleta:

- Para visualizar la paleta de propiedades:
  - Ctrl+Shift+7
  - Windows > Properties

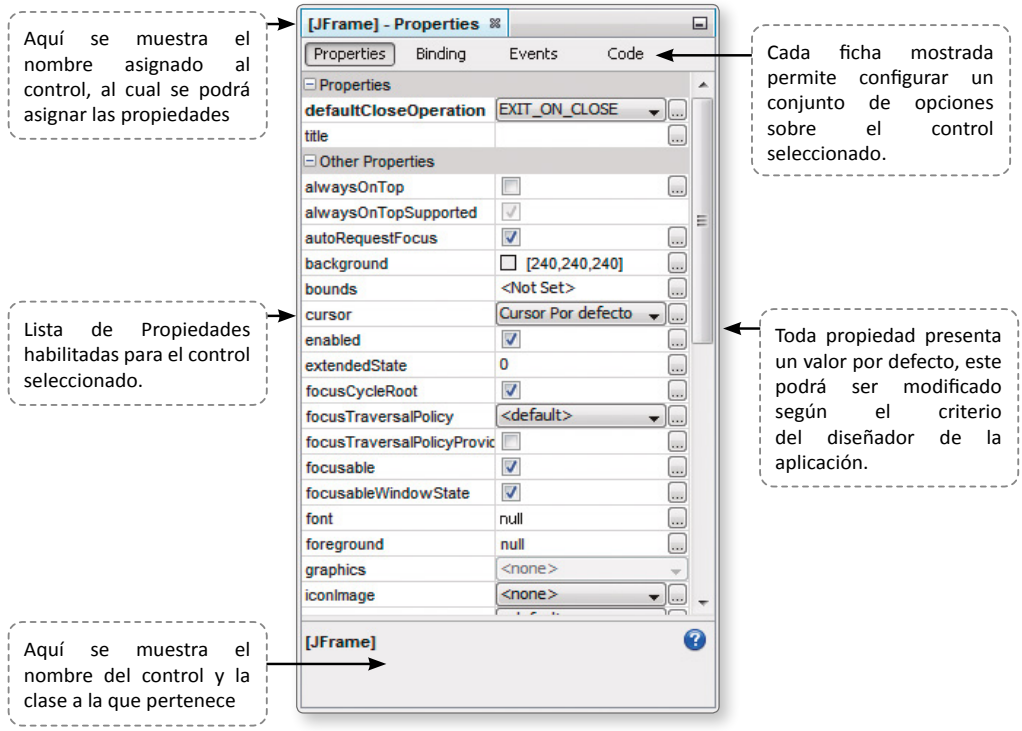


Fig. 3.12

Ahora que conocemos los controles y las propiedades podemos agregar los controles al Frame y componer la GUI de la aplicación, para esto solo se debe arrastrar desde la paleta swing container o swing controls hacia el frame de modo que al soltarlo tenga el aspecto que desee el usuario.

Entonces, una vez colocados los controles tendremos la siguiente estadística de elementos dentro de la paleta navegador. Como lo muestra la Fig. 3.13.

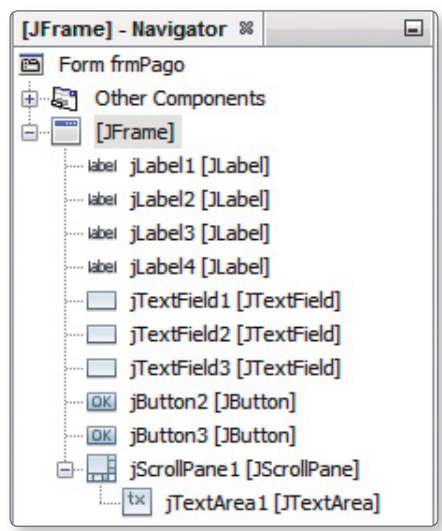


Fig. 3.13

3.14.1. Cambiando el contenido visual de los controles

En una GUI normalmente se cambia el contenido textual de aquellos controles que necesitan informar algo al usuario, en este caso modificará todos los controles Label y Button, mientras que los controles TextField deben estar vacíos, para esto se debe presionar clic derecho sobre el control > Edit Text y modifique el texto. Como lo muestra la Fig. 3.14.

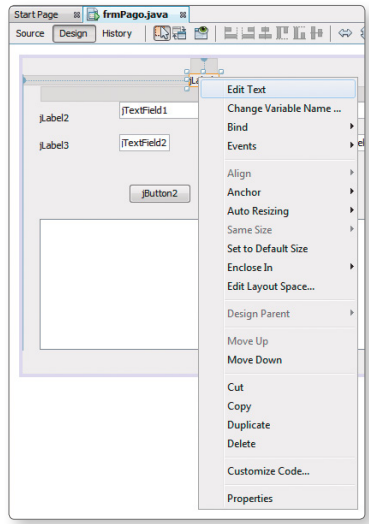


Fig. 3.14

Por las propiedades también se puede modificar el contenido de los controles, así lo muestra la Fig. 3.15:

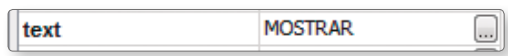


Fig. 3.15

Luego de cambiar el aspecto de los controles le toca asignarle un nombre a todos los objetos que sean necesarios. Para esto puede presionar clic derecho sobre el control > Change Variable Name… y colocar el nombre al control según la nomenclatura de nombres para los objetos Java.

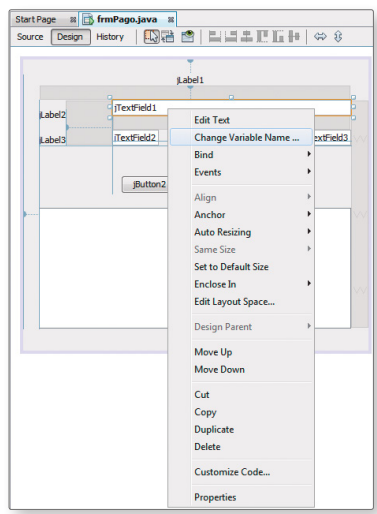


Fig. 3.16

Al seleccionar Change Variable Name aparece una ventana de renombre como lo muestra la Fig. 3.17. Asígnele un nombre adecuado al control y presione Ok.

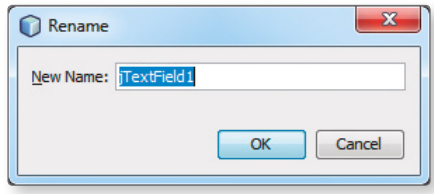


Fig. 3.17

Luego desde la paleta navegación verá los nombres asignados a los controles como lo muestra la Fig. 3.18.

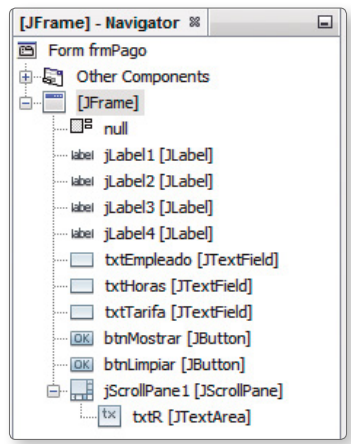


Fig. 3.18

Luego de los cambios de aspecto que se le puede dar a los controles, y especialmente los nombres de los objetos, debe tener el siguiente aspecto de la aplicación. Como lo muestra la Fig. 3.19.

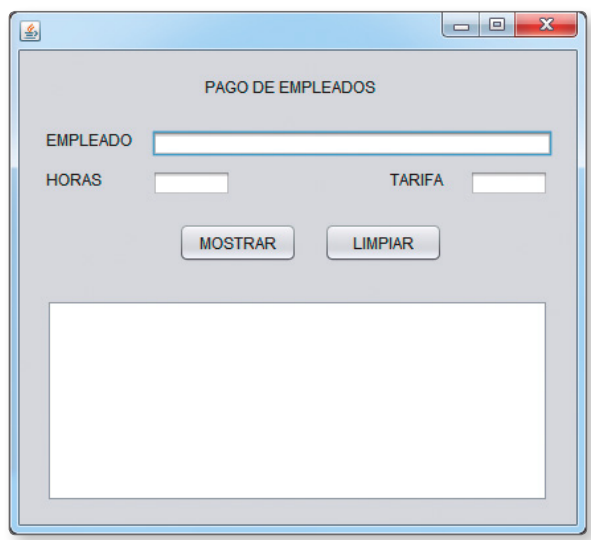


Fig. 3.19

Para asignar el script que dará funcionamiento a la aplicación debe presionar doble clic sobre el botón Mostrar y colocar el siguiente código:

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
      //Entrada
    int horas=Integer.parseInt(txtHoras.getText());
    double tarifa=Double.parseDouble(txtTarifa.getText());

    //Proceso
    double basico = horas * tarifa;
    double bonificacion = basico * 0.2;
    double neto = (basico + bonificacion) * 0.9;

    //Salida
    txtR.setText(" ** Resumen de Pago a Empleado ** ");
    txtR.append("El básico es: "+basico);
    txtR.append("La bonificación es: "+bonificacion);
    txtR.append("El neto es: "+neto);
```

Ahora presione doble clic sobre el botón Limpiar y coloque el siguiente código:

```
  private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
     txtEmpleado.setText("");
    txtHoras.setText("");
    txtTarifa.setText("");
    txtR.setText("");
    txtEmpleado.requestFocus();
  }
```

Antes de ejecutar la aplicación debe configurar la propiedad **Form Size Policy** del objeto Frame con la opción **Generate Resize Code**, con esto se logrará visualizar el formulario.

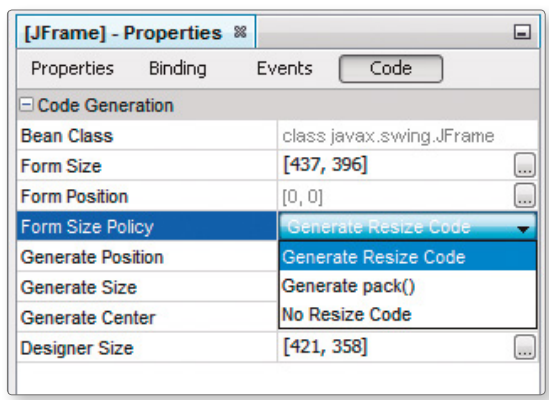


Fig. 3.20

Para ejecutar una aplicación en NetBeans presionará Shift + F6 e ingresará los valores mostrados en la Fig. 3.21.

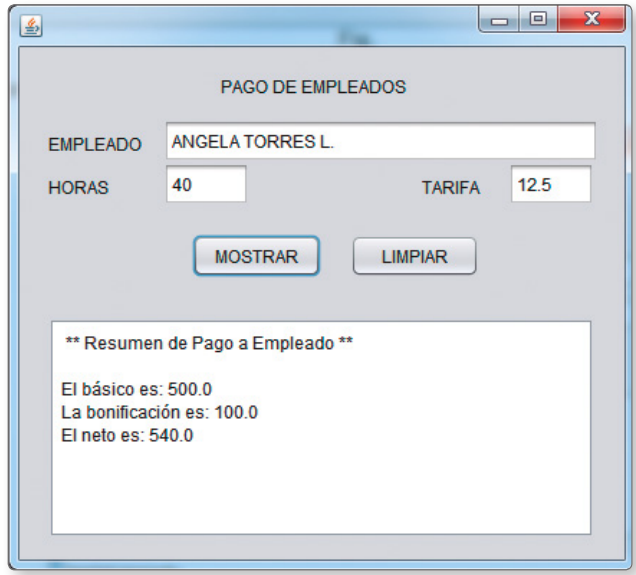


Fig. 3.21

3.15. CONSTRUIR GUI CON JDEVELOPER

Creará la misma aplicación mostrada con JCreator y NetBeans, pero esta vez usando JDEveloper:

Primero debe conocer las paletas que componen el entorno de JDeveloper:

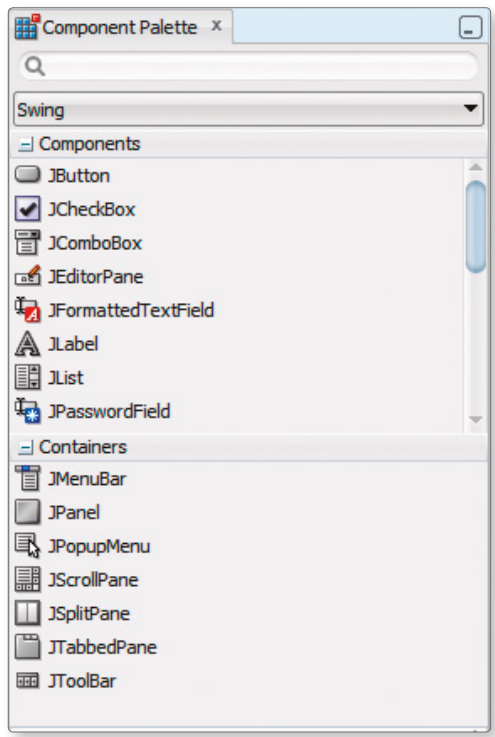


Fig. 3.22

- Para visualizar la paleta:
  - Ctrl +Shift+P
  - View > Component Pallete

Swing Container: se nombrarán los principales contenedores:

JPanel	Crea un objeto Panel contenedor de más controles.
JScrollPane	Crea un objeto contenedor para un cuadro de lista (JList) o un área de texto (JTextArea), en la cual permite incluir barras de desplazamiento a dichos controles.
JTabbedPane	Crea un objeto tipo Tabbet que permite tener una aplicación en fichas.

Swing Controls:

JLabel	Permite crear un objeto de la clase JLabel.
JButton	Permite crear un objeto de la clase JButton.
JCheckBox	Permite crear un objeto de la clase JCheckBox.
JComboBox	Permite crear un objeto de la clase JRadioButton.
JList	Permite crear un objeto de la clase JComboBox.
	Permite crear un objeto de la clase JList. Para mostrar una barra de desplazamiento se tiene que incluir dentro del control ScrollPane.
JTextField	Permite crear un objeto de la clase JTextField.
JTextArea	Permite crear un objeto de la clase JTextArea.
JPasswordField	Permite crear un objeto de la clase JPasswordField.
JTable	Permite crear un objeto de la clase JTable.

Por cada control colocado dentro del contenedor Frame se tiene un conjunto de propiedades, veamos dicha paleta:

- Para visualizar la paleta de propiedades:
  - Ctrl+Shift+i
  - View > Property Inspector

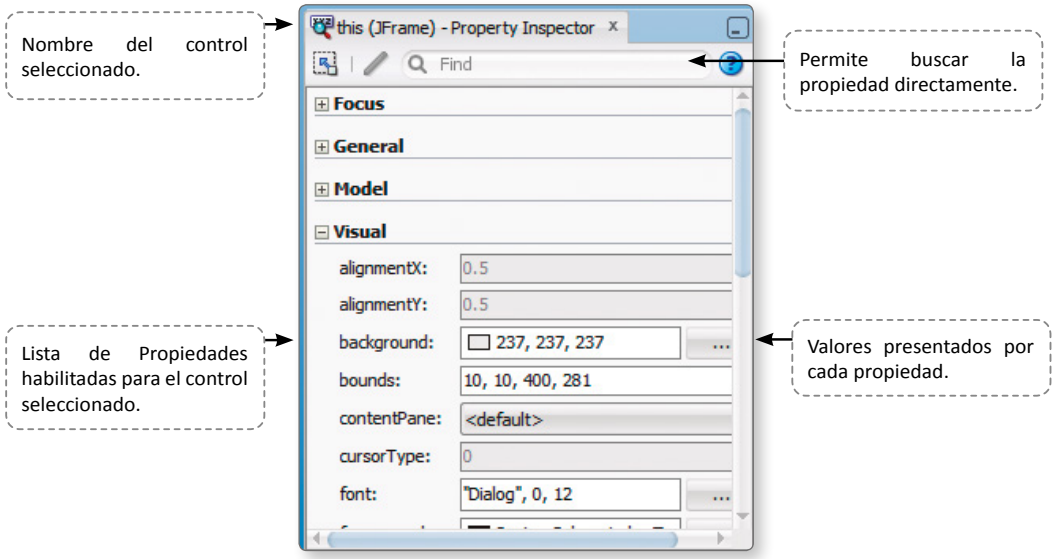


Fig. 3.23

Como sucedió con NetBeans para construir la GUI solo debe arrastrar los controles del component pallete y asignar las propiedades necesarias.

Entonces, una vez colocados los controles tendremos la siguiente estadística de elementos dentro de la paleta structure. Como lo muestra la Fig. 3.24:

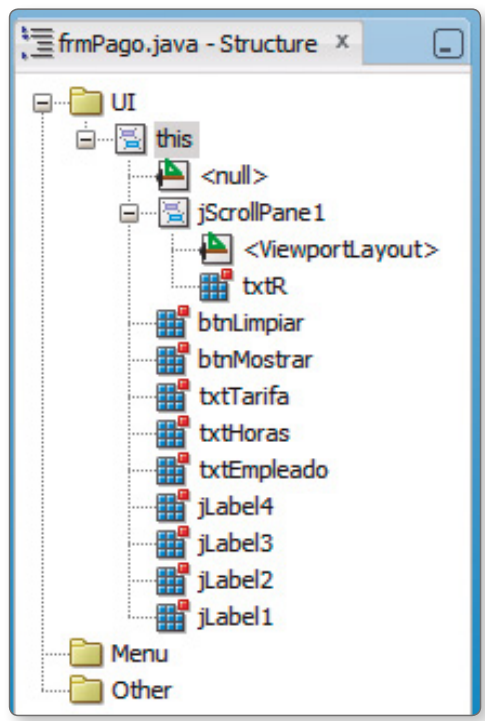


Fig. 3.24

3.15.1. Cambiando el contenido visual de los controles

Para cambiar el aspecto visual primero seleccione el control y luego desde las propiedades seleccione la ficha Visual >

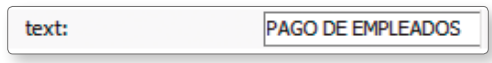


Fig. 3.25

Luego de cambiar el aspecto de los controles le toca asignarle un nombre a todos los objetos que sean necesarios. Para esto puede presionar la combinación de teclas CTRL+ALT+R así como lo muestra la Fig. 3.26.

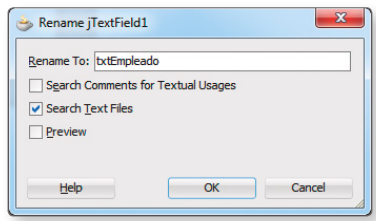


Fig. 3.26

Una vez terminado el diseño de la GUI colocará los códigos Java en sus respectivos botones, para esto presionará doble clic sobre el botón mostrar y agregará el siguiente código:

```
    private void btnMostrar_actionPerformed(ActionEvent e) {
      //Entrada
      int horas=Integer.parseInt(txtHoras.getText());
      double tarifa=Double.parseDouble(txtTarifa.getText());

      //Proceso
      double basico = horas * tarifa;
      double bonificacion = basico * 0.2;
      double neto = (basico + bonificacion) * 0.9;

      //Salida
      txtR.setText(" ** Resumen de Pago a Empleado ** \n");
      txtR.append("\nEl básico es: "+basico);
      txtR.append("\nLa bonificación es: "+bonificacion);
      txtR.append("\nEl neto es: "+neto);
    }
```

Seguidamente, presione doble clic sobre el botón Limpiar y coloque el siguiente código:

```
private void btnLimpiar_actionPerformed(ActionEvent e) {
      txtEmpleado.setText("");
      txtHoras.setText("");
      txtTarifa.setText("");
      txtR.setText("");
      txtEmpleado.requestFocus();
    }
```

Antes de ejecutar la aplicación, cree un método principal que permita inicializar qué formulario deberá mostrarse al ejecutar la aplicación.

```
public static void main(String args[]){
    frmPago fraVista=new frmPago();
    fraVista.setVisible(true);
}
```

El método void main(String args[]) no devuelve ningún valor, pero tiene por función hacer visible al Frame. Luego se crea un objeto de la clase frmPago llamado fraVista que luego será visible por el método setVisible(true).

Finalmente, para compilar y ejecutar la aplicación presionaremos F11 e ingresaremos los valores, así como se muestra en la Fig. 3.27:

Fig. 3.27

CAP.

4

Estructura de secuencia

CAPACIDAD:

- Implementar una entrada y salida en Java.
- Reconocer los elementos de la clase Math.
- Desarrollar aplicaciones básicas, usando estructuras de secuencia.

CONTENIDO:

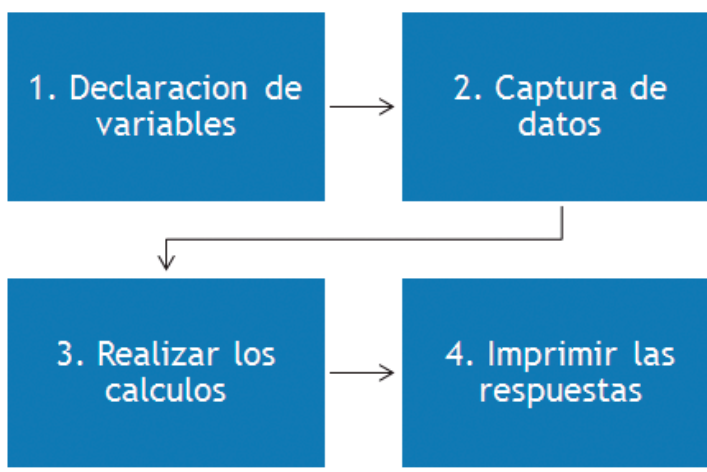
4.1. INTRODUCCIÓN

El cuerpo de un programa en Java está compuesto por un conjunto de sentencias que tienen por misión realizar una o más acciones según lo especificado por el programador. Estas sentencias se implementan en forma de secuencia dependiendo del objetivo de la aplicación, aquí hay que recalcar que el orden de estas será responsabilidad directa del programador.

También se debe considerar que una sentencia Java hace referencia a un cálculo, captura de datos, impresión de valores, almacenado de datos, etc, y estos se ejecutan uno a uno siguiendo el orden implementado. Veamos un ejemplo de una estructura secuencial:

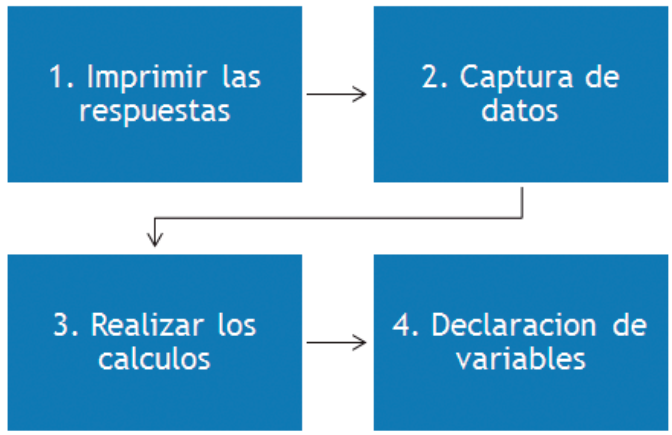
```
public void actionPerformed( ActionEvent e ){
  //1.
  double horasTrab, tarifaHor;
  double sueldoBas, montoBoni, sueldoBru, montoDesc, sueldoNet;

  //2.
  horasTrab = Double.parseDouble(txtHoras.getText());
  tarifaHor = Double.parseDouble(txtTarifa.getText());

  //3.
  sueldoBas = horasTrab*tarifaHor;
  montoBoni = 0.20*sueldoBas;
  sueldoBru = sueldoBas+montoBoni;
  montoDesc = 0.10*sueldoBru;
  sueldoNet = sueldoBru-montoDesc;

  //4.
  txtS.setText("Sueldo básico : $ " + sueldoBas + "\n");
  txtS.append ("Bonificación : $ " + montoBoni + "\n");
  txtS.append ("Sueldo bruto : $ " + sueldoBru + "\n");
  txtS.append ("Descuentos : $ " + montoDesc + "\n");
  txtS.append ("Sueldo neto : $ " + sueldoNet);
 }
```

Veamos un gráfico genérico de la aplicación:



Como verá la secuencia es proporcional al objetivo de la aplicación ya que para todo programa se debe declarar variables que se usarán en la aplicación, capturar los datos desde los objetos swing, una vez capturados empezar con los cálculos necesarios en la aplicación y finalmente la impresión de los resultados.

Si cambiamos el orden; por ejemplo, el 1 por el 4:



Tendríamos que en el primer punto debemos imprimir una respuesta que aún no se ha implementado, esto acarrea muchos errores puesto que el punto dos se captura los datos, pero como no se ha declarado generaría errores de declaración. En el punto tres se realizan los cálculos con variables aún no declaradas y, finalmente, para concluir el caos en el punto cuatro recién se declaran las variables. Como notará la secuencialidad define el orden lógico de la aplicación.

4.2. Entrada y salida de datos

Cuando se desarrolla aplicaciones Java siempre se tendrá la necesidad de interactuar la aplicación con el usuario, esto se realiza mediante la interfaz implementada por el programador. Veremos a continuación las diferentes formas que la aplicación puede responder al usuario ya sea mediante un mensaje o solicitud de algún dato para la aplicación, para esto usaremos la clase JOptionPane.

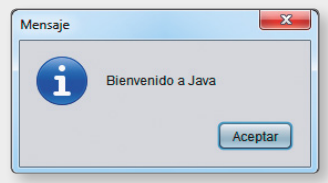
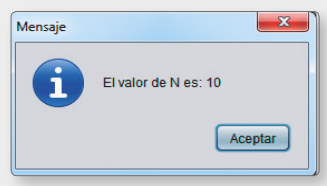
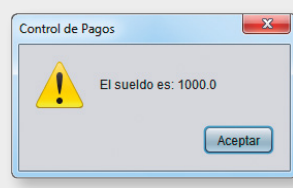
La clase JOptionPane permite:

- Enviar mensajes informativos al usuario
- Mostrar mensajes de error
- Mostrar mensajes de advertencia
- Enviar solicitudes de selección
- Solicitar datos de entrada para la aplicación

Librería: javax.swing.JOptionPane

Métodos de la clase JOptionPane

MÉTODO	ESPECIFICACIÓN Y EJEMPLO

showMessageDialog	`JOptionPane.showMessageDialog(Destino,"Mensaje", "Titulo",Icono)`
	
	Este método nos permite mostrar un mensaje personalizado por el usuario y un botón de forma predeterminada. El método tiene como argumentos:
	
	◆ Destino del Diálogo: este determina en donde se va a colocar el cuadro de diálogo, se le asigna null para que pueda aparecer en el centro de la pantalla.
	◆ Mensaje: es el texto que se mostrará al usuario.
	◆ Título: es el texto mostrado en la barra de título del mensaje.
	◆ Ícono: define qué tipo de ícono se mostrará al usuario.
	
	Caso 1. Mostrar un mensaje de diálogo con el mensaje "Bienvenido a Java".
	
	`JOptionPane.showMessageDialog(null, "Bienvenido a Java");`
	
	 
	
	Caso 2. Mostrar el mensaje "El valor de N es: " y que imprima el valor contenido en la variable N.
	
	`int n=10;`
	`JOptionPane.showMessageDialog(null, "El valor de N es: "+n);`
	
	 
	
	Como notará el mensaje que se muestra al usuario se concatena con la variable n como valor; esto es gracias al operador de concatenación +.
	
	Caso 3. Mostrar el valor contenido en la variable sueldo, con el título "Control de Pagos" y el ícono Warning.
	
	`double sueldo=1000;`
	`JOptionPane.showMessageDialog(null, "El sueldo es: "+sueldo, "Control de Pagos", JOptionPane.WARNING_MESSAGE);`
	
	 

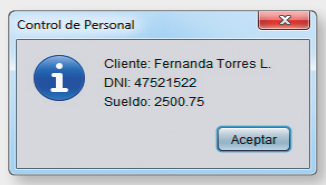
<table>
<tr><th>Método</th><th>Especificación y Ejemplo</th></tr>
<tr><td></td><td>

Caso 4. Mostrar un mensaje de error al usuario cuando la calificación ingresada no sea válida.

```
byte calificacion=-10;
if (calificacion<0 || calificacion>20)
   JOptionPane.showMessageDialog(null,
          "Calificacion no Valida",
          "Control de notas",
          JOptionPane.ERROR_MESSAGE);
```

Caso 5. Mostrar los datos de un empleado en un solo mensaje haciendo un cambio de línea por cada dato.

```
String cliente="Fernanda Torres L.";
String DNI="47521522";
double sueldo=2500.75;
JOptionPane.showMessageDialog(null,
                    "Cliente: "+cliente+"\n"
                    +"DNI: "+DNI+"\n"
                    +"Sueldo: "+sueldo,
                    "Control de Personal",
                    JOptionPane.INFORMATION_MESSAGE);
```



</td></tr>
<tr><td>showOptionDialog</td><td>

```
JOptionPane.showOptionDialog(Destino,"Mensaje",
                             "Titulo",TipoOpcion,
                             TipoMensaje,Icono,
                             Opciones,valorInicial )
```

Este método nos permite mostrar un cuadro de diálogo en el cual se puede tener un mejor control de los elementos que se mostrarán. Cuenta con los siguientes parámetros:

- Destino: aquí se asigna el contenedor del cuadro de diálogo, en forma predeterminada se le asigna null.
- Mensaje: es el texto que visualizará el usuario.
- Título: es el texto que se mostrará como título de la ventana de diálogo.
- TipoOpcion: aquí se indica que tipo de opción visualizará el usuario, como por ejemplo: DEFAULTOPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION o OK_CANCEL_OPTION.
- TipoMensaje: aquí se indica que tipo de mensaje se mostrará al usuario, desde aquí se determina qué tipo de ícono está asignado al mensaje, como por ejemplo ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE o PLAIN_MESSAGE.

</td></tr>
</table>

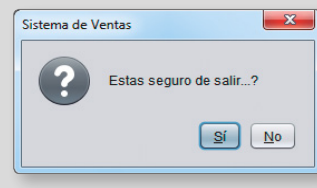
<table>
<tr><th>MÉTODO</th><th>ESPECIFICACIÓN Y EJEMPLO</th></tr>
<tr><td></td><td>

Opciones: el valor predeterminado es null, el cual permite definir los textos de los botones según el tipoOpcion. En otros casos se puede implementar un array de objetos.
valorInicial: es la selección por defecto, la cual define que uno de los objetos del cuadro de diálogo tiene el foco activo.

El método showOptionDialog devuelve un valor entero que determinará qué tipo de botón ha seleccionado el usuario. Como por ejemplo, el botón SI tiene el valor 0, el botón No tiene el valor 1 y si no se selecciona ningún botón, el valor es -1.

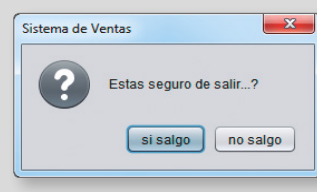
Caso 1. Mostrar un mensaje de salida al usuario con valores predeterminados, en caso seleccione SI deberá salir de la aplicación, caso contrario seguirá en la aplicación.

<pre>int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Sistema de Ventas",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);

if (r==0) System.exit(0);</pre>


Caso 2. Mostrar un mensaje de salida al usuario con valores personalizados en los botones y predeterminando al botón de salida.

<pre>int r=JOptionPane.showOptionDialog(this,
           "Estas seguro de salir...?",
           "Sistema de Ventas",
           JOptionPane.YES_NO_OPTION,
           JOptionPane.QUESTION_MESSAGE,
           null,
           new Object[] { "si salgo", "no salgo"},
           "si salgo");

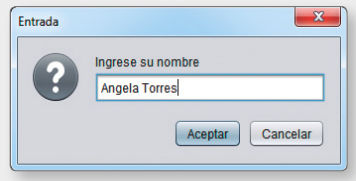
if (r==0) System.exit(0);</pre>


</td></tr>
<tr><td>showInputDialog</td><td>
<pre>JOptionPane.showInputDialog(Object mensaje)</pre>
Este método permite solicitar un valor de tipo String al usuario, el cual podrá administrarlo de la mejor forma que crea posible.

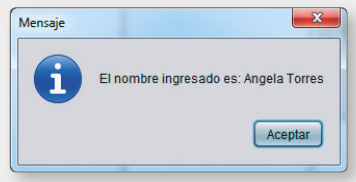
Caso 1. Mostrar un cuadro de diálogo que solicite el nombre de un determinado empleado en forma básica.
</td></tr>
</table>

MÉTODO	ESPECIFICACIÓN Y EJEMPLO

```
String nombre = JOptionPane.showInputDialog(
                                "Ingrese su nombre");
JOptionPane.showMessageDialog(null,
                  "El nombre ingresado es: "+nombre);
```

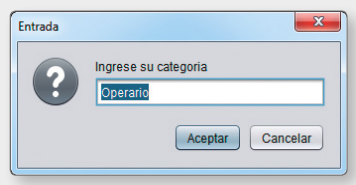


El método showInputDialog permite el ingreso de un valor, en este caso el nombre es de tipo String así que no necesita conversión. La siguiente ventana de diálogo muestra el valor capturado a partir del diálogo anterior.



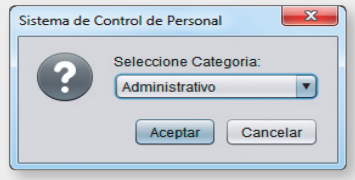
Caso 2. Mostrar un cuadro de diálogo que solicite el ingreso de una categoría al empleado mostrando en forma predeterminada el valor "Operario".

```
String categoria = JOptionPane.showInputDialog(
                        "Ingrese su categoria",
                        "Operario");

JOptionPane.showMessageDialog(null,
      "La categoria ingresada es: "+categoria);
```



Caso 3: Mostrar un cuadro de diálogo que permita seleccionar la categoría del empleado por medio de un cuadro combinado.

```
String categoriaSeleccionada=String.valueOf(
         JOptionPane.showInputDialog(null,
             "Seleccione Categoria: ",
             "Sistema de Control de Personal",
             JOptionPane.QUESTION_MESSAGE,
             null,
             new Object[] {"Operario",
                           "Administrativo",
                           "Servicio"},
                            "Administrativo"));

JOptionPane.showMessageDialog(null,
       "La categoria seleccionada es: "+
       categoriaSeleccionada);
```

MÉTODO	ESPECIFICACIÓN Y EJEMPLO
	

4.3. La clase Math

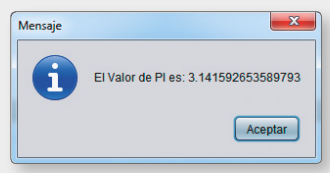
Pertenece a java.lang.Math el cual contiene métodos estáticos que nos permitirán realizar cálculos sobre funciones matemáticas en una aplicación Java. Esta clase es de visibilidad pública, es decir, podremos tener acceso a todos sus métodos en cualquier parte de la aplicación. Consideremos los siguientes aspectos:

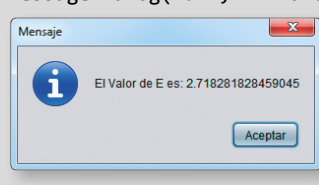
- Los métodos por ser estáticos no necesitan inicialización de valores.
- Los métodos seno(sin), coseno(cos) y tangente(tan) trabajan con ángulos radianes, por tanto al usarlos deberá convertirlos previamente.
- Un logaritmo neperiano (Ln) tiene la siguiente expresión en Java Math.log(x).
- Una exponenciación neperiana expresada matemáticamente como e^x tiene la siguiente expresión en Java Math.exp(x), donde e tiene un valor de 2.7182…

4.4. Métodos que representan a PI y E

La clase Math hace referencia especialmente a estas dos constantes más usadas en funciones matemáticas, asumiendo una precisión de 15 decimales:

MÉTODO	DESCRIPCIÓN

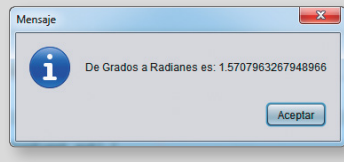
PI	Veamos un ejemplo del uso del método PI.
	`double PI=Math.PI;`
	`JOptionPane.showMessageDialog(null,"El Valor de PI es: "+PI);`
	 

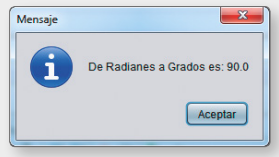
e	Veamos un ejemplo del uso del método E.
	`double e=Math.E;`
	`JOptionPane.showMessageDialog(null,"El Valor de E es: "+e);`
	 

4.5. MÉTODOS DE CONVERSIÓN ENTRE GRADOS Y RADIANES

Haremos referencia a la conversión entre grados y radianes.

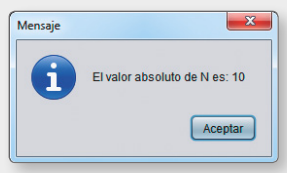
MÉTODO	DESCRIPCIÓN

De grados a radianes	`double grados = 90;`
	`double radianes = Math.toRadians(grados);`
	`JOptionPane.showMessageDialog(null,`
	`"De Grados a Radianes es: "+radianes);`
	 

De radianes a grados	`double radianes = 1.5707963267948966;`
	`double grados = Math.toDegrees(radianes);`
	`JOptionPane.showMessageDialog(null,`
	`"De Radianes a Grados es: "+grados);`
	 

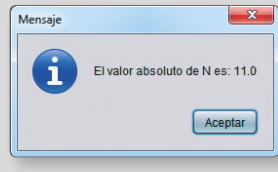
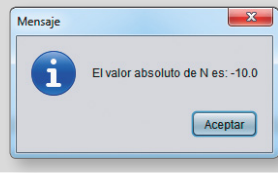
4.6. MÉTODOS DE LA CLASE MATH

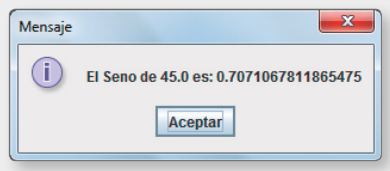
MÉTODO	DESCRIPCIÓN

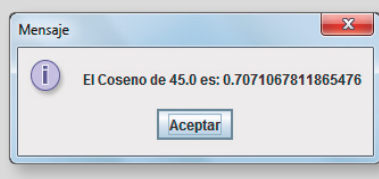
abs	Determine el valor absoluto de un número.
	
	Ejm. Determinar el valor absoluto del valor -10.
	
	`int n = -10;`
	`JOptionPane.showMessageDialog(null,`
	`"El valor absoluto de N es: "+Math.abs(n));`
	 

ceil	Redondea un valor decimal al entero más pequeño no menor a sí mismo.
	
	Ejm. Determinar el valor redondeado de 10.254
	
	`double d = 10.254;`
	`JOptionPane.showMessageDialog(null,`
	`"El valor redondeado d es: "+Math.ceil(d));`

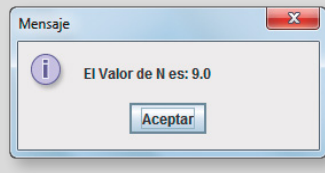
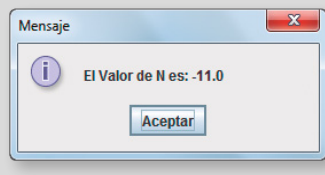
MÉTODO	DESCRIPCIÓN

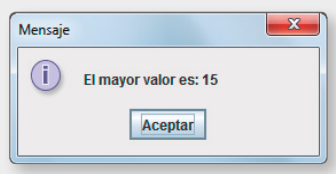
	
	Probaremos con un valor negativo -10.95
	`double d = -10.95;`
	`JOptionPane.showMessageDialog(null,`
	`"El valor absoluto de N es: "+Math.ceil(d));`
	 

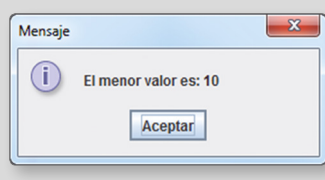
sin	Determina el seno trigonométrico de un ángulo expresado en radianes.
	Ejm. Determinar el seno de 45°.
	`double grados = 45;`
	`double radianes = Math.toRadians(grados);`
	`JOptionPane.showMessageDialog(null,`
	`"El Seno de "+grados +" es: "+Math.sin(radianes));`
	 

cos	Determina el coseno trigonométrico de un ángulo expresado en radianes.
	Ejm. Determinar el coseno de 45°.
	`double grados = 45;`
	`double radianes = Math.toRadians(grados);`
	`JOptionPane.showMessageDialog(null,`
	`"El Coseno de "+grados +" es: "+Math.cos(radianes));`
	 

MÉTODO	DESCRIPCIÓN

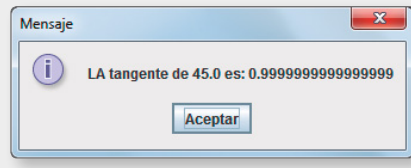
floor	Redondea un valor decimal al entero más grande no mayor a sí mismo.
	
	Ejm. Determinar el redondeo del valor 9.2
	
	`double n=9.2;`
	`JOptionPane.showMessageDialog(null,`
	`"El Valor de N es: "+Math.floor(n));`
	
	 
	
	`double n=-10.5;`
	`JOptionPane.showMessageDialog(null,`
	`"El Valor de N es: "+Math.floor(n));`
	
	 

max	Determina el máximo valor entre dos valores numéricos.
	
	Ejm. Determinar el mayor valor entre 10 y 15.
	
	`int a=10;`
	`int b=15;`
	`JOptionPane.showMessageDialog(null,`
	`"El mayor valor es: "+Math.max(a,b));`
	
	 

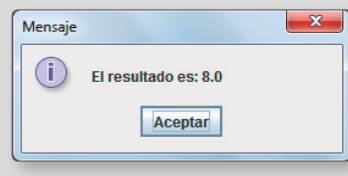
min	Determina el mínimo valor entre dos valores numéricos.
	
	Ejm. Determinar el menor valor entre 10 y 15.
	
	`int a=10;`
	`int b=15;`
	`JOptionPane.showMessageDialog(null,`
	`"El menor valor es: "+Math.min(a,b));`
	
	 

<table>
<tr><th>MÉTODO</th><th>DESCRIPCIÓN</th></tr>
<tr><td>tan</td><td>Determina la tangente trigonométrica de un ángulo expresado en radianes.

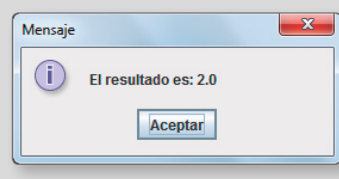
Ejm. Determinar la tangente de 45°.

<pre>
double grados = 45;
double radianes = Math.toRadians(grados);
JOptionPane.showMessageDialog(null,
         "La tangente de "+grados +" es: "+Math.tan(radianes));
</pre>

</td></tr>
<tr><td>pow</td><td>Determina la potencia n de un valor numérico entero.

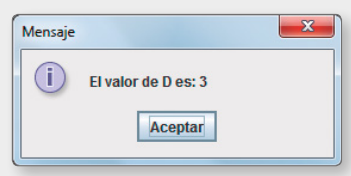
Ejm. Determinar la potencia 3 del valor 2.

<pre>
int base=2;
int potencia=3;
JOptionPane.showMessageDialog(null,
           "El resultado es: "+Math.pow(base,potencia));
</pre>


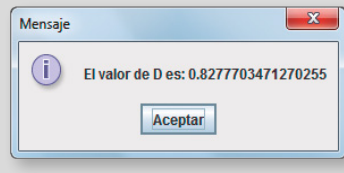
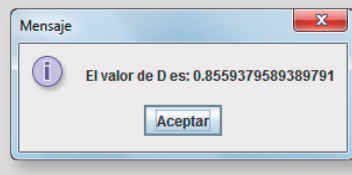
Probaremos con la raíz cúbica de 8.

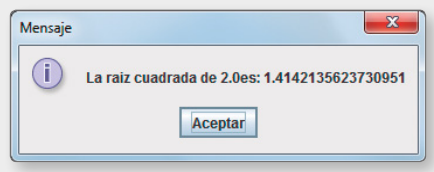
<pre>
int base=8;
JOptionPane.showMessageDialog(null,
           "El resultado es: "+Math.pow(base,(1/3.0)));
</pre>

</td></tr>
<tr><td>round</td><td>Permite expresar un número real con un número específico de decimales aplicando el exceso y defecto matemático.

Ejm. Aplicar el redondeo a cero decimales del valor 2.55

<pre>
double d=2.55;
JOptionPane.showMessageDialog(null,
           "El valor de D es: "+Math.round(d));
</pre>

</td></tr>
</table>

MÉTODO	DESCRIPCIÓN

random	Permite devolver un número aleatorio entre 0 y 1 excluyendo a estos valores.
	
	Ejm. Determinar un número aleatorio cualquiera.
	
	`double d=Math.random();`
	`JOptionPane.showMessageDialog(null, "El valor de D es: "+d);`
	
	 
	 

sqrt	Devuelve la raíz cuadrada de un número entero.
	
	Ejm. Determinar la raíz cuadrada del valor 2.
	
	`double d=2;`
	`JOptionPane.showMessageDialog(null,`
	`"La raiz cuadrada de "+d+"es: "+Math.sqrt(d));`
	
	 

CASO DESARROLLADO 1: CONVERSIONES

Implemente una aplicación Java que permita leer una temperatura en grados Centígrados (C) y la convierta a su equivalente en grados Fahrenheit (F), grados Kelvin (K) y grados Rankine (R). Utilice las siguientes fórmulas:

Farenheit	9 Celsius/5 + 32
Kelvin	Rankine - 187
Rankine	Celsius + 460

Con JCreator:

PASO 1:

Crear un espacio de trabajo llamado pjSecuencial en JCreator, como lo muestra la Fig. 4.1.

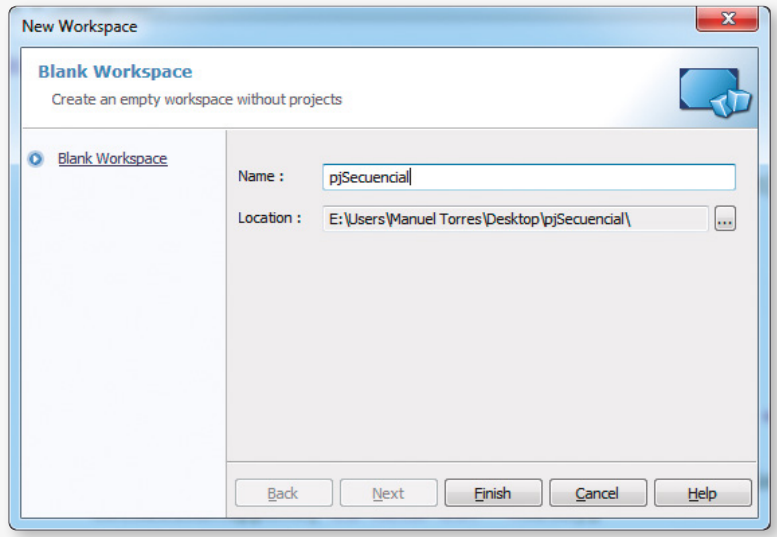


Fig. 4.1

PASO 2:

Agregamos el proyecto frmTemperatura al espacio de trabajo pjSecuencial, como lo muestra la Fig. 4.2.

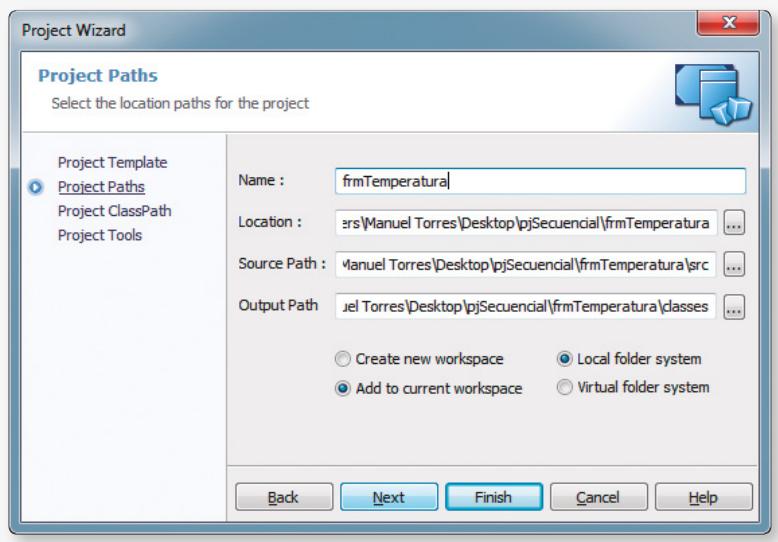


Fig. 4.2

PASO 3:

Dentro de la clase frmTemperatura agregaremos el siguiente script:

```
//1.
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

public class frmTemperatura extends JApplet implements ActionListener {
    //2.
    JLabel lblTitulo,lblCelsius;
    JTextField txtCelsius;
    JButton btnMostrar, btnLimpiar;
    JTextArea txtR;
    JScrollPane spDesplaza;

    public void init() {
            //3.
            this.setLayout(null);

            lblTitulo = new JLabel("CONVERSION DE TEMPERATURAS");
            lblTitulo.setBounds(180,15,200,20);
            add(lblTitulo);

            lblCelsius = new JLabel("TEMPERATURA EN CELSIUS ");
            lblCelsius.setBounds(20,50,200,20);
            add(lblCelsius);

            txtCelsius = new JTextField();
            txtCelsius.setBounds(200,50,100,20);
            add(txtCelsius);

            btnMostrar = new JButton("MOSTRAR");
            btnMostrar.setBounds(320,50,100,20);
            btnMostrar.addActionListener(this);
            add(btnMostrar);

            btnLimpiar = new JButton("LIMPIAR");
            btnLimpiar.setBounds(20,180,100,20);
            btnLimpiar.addActionListener(this);
            add(btnLimpiar);

            txtR = new JTextArea();
            txtR.setFont(new Font("Trebuchet MS", 0, 12));

            spDesplaza = new JScrollPane(txtR);
            spDesplaza.setBounds(20, 80, 450, 80);
            add(spDesplaza);

    }
    public void actionPerformed(ActionEvent e){
    if (e.getSource()==btnMostrar){
            //4.
            double celsius=Double.parseDouble(txtCelsius.getText());

            //5.
            double farenheit=(9 * celsius)/5 + 32;
            double rankine=celsius + 460;
            double kelvin=rankine - 187;

            //6.
            txtR.setText(" ** RESUMEN DE TEMPERATURAS ** ");
            txtR.append("\nGRADOS FARENHEIT  : "+farenheit);
            txtR.append("\nGRADOS KELVIN     : "+kelvin);
            txtR.append("\nGRADOS RANKINE    : "+rankine);
```

En el punto uno se importan las librerías necesarias para la aplicación.

En el punto dos se declaran las variables globales de la aplicación.

En el punto tres se implementan los objetos que serán visibles por el usuario.

En el punto cuatro se declara la variable Celsius que tiene por misión almacenar el valor ingresado por el usuario mediante el control txtCelsius.

En el punto cinco se declaran las variables Farenheit, Rankine y Kelvin según la fórmula proporcionada por el problema.

En el punto seis se imprime las respuestas en el objeto txtR.

En el punto siete se implementa el código de limpieza de los controles usados en la aplicación.

Una vez terminada la aplicación, tanto en la GUI como en el código se deben compilar y luego ejecutar, presionando F7 y seguidamente F5.

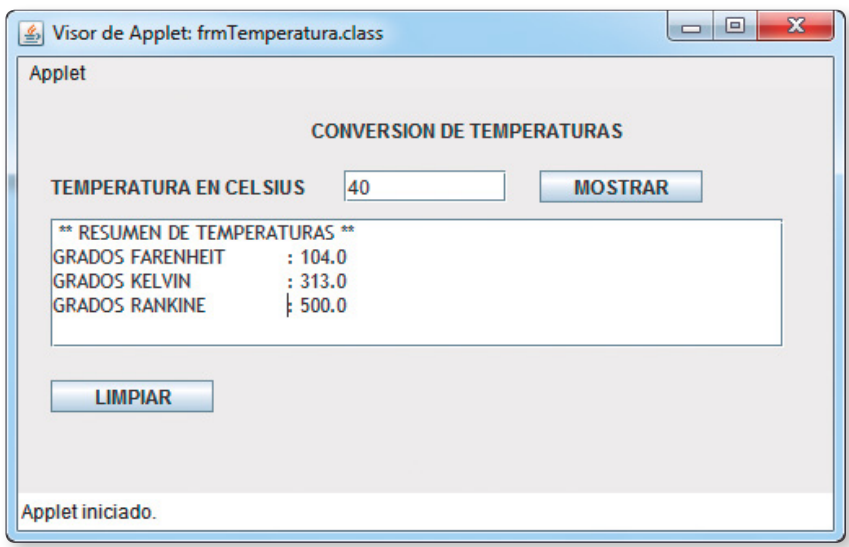


Fig. 4.3

Ingrese una temperatura válida en la caja de texto y presione el botón Mostrar para ver los resultados de la aplicación.

Con Netbeans

PASO 1:

Crear una nueva aplicación en NetBeans llamado pjSecuencial y desactivar el check Create Main Class, como lo muestra la Fig. 4.4.

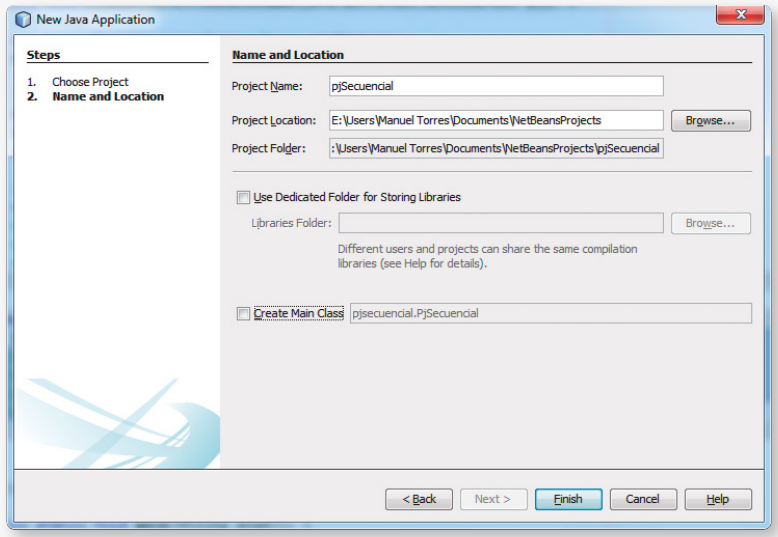


Fig. 4.4

PASO 2:

Agregamos el paquete pFormularios al proyecto pjSecuencial, como lo muestra la Fig. 4.5.

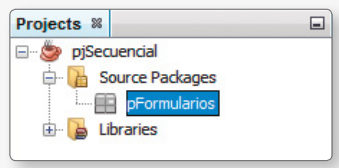


Fig. 4.5

PASO 3:

Dentro del paquete pFormularios agregamos un Frame llamado frmTemperatura, como lo muestra la Fig. 4.6.

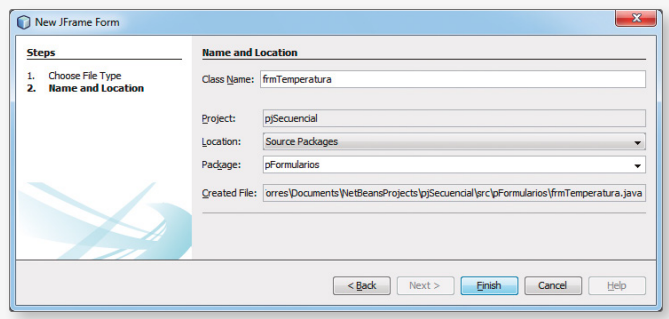


Fig. 4.6

PASO 4:

Diseñe la siguiente GUI:

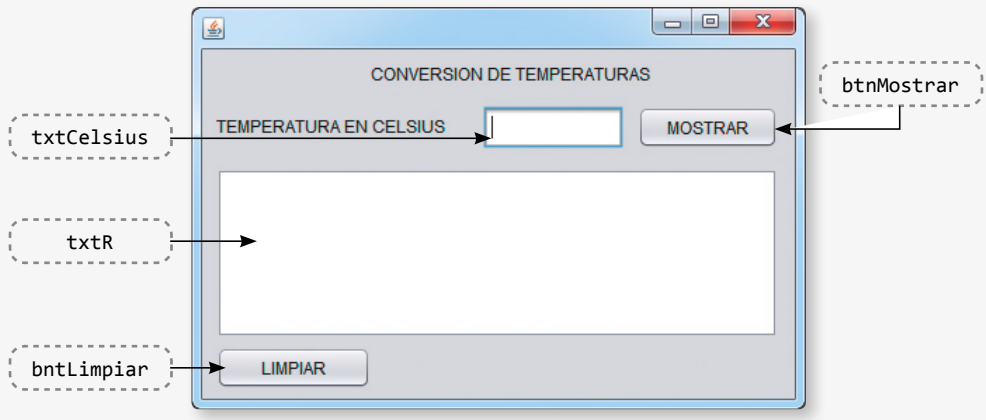


Fig. 4.7

PASO 5:

Coloque el siguiente código dentro del botón btnMostrar

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
      double celsius=Double.parseDouble(txtCelsius.getText());

    double farenheit=(9 * celsius)/5 + 32;
    double rankine=celsius + 460;
    double kelvin=rankine - 187;

    txtR.setText(" ** RESUMEN DE TEMPERATURAS ** ");
    txtR.append("\nGRADOS FARENHEIT  : "+farenheit);
    txtR.append("\nGRADOS KELVIN     : "+kelvin);
    txtR.append("\nGRADOS RANKINE    : "+rankine);
}
```

Dentro del boton btnLimpiar colocar el siguiente código:

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
      txtCelsius.setText("");
    txtR.setText("");
    txtCelsius.requestFocus();
}
```

Finalmente, presione Shift+F6 para ejecutar la aplicación e ingresaremos el valor 40 y presionamos el botón Mostrar. Como lo muestra la Fig. 4.8.

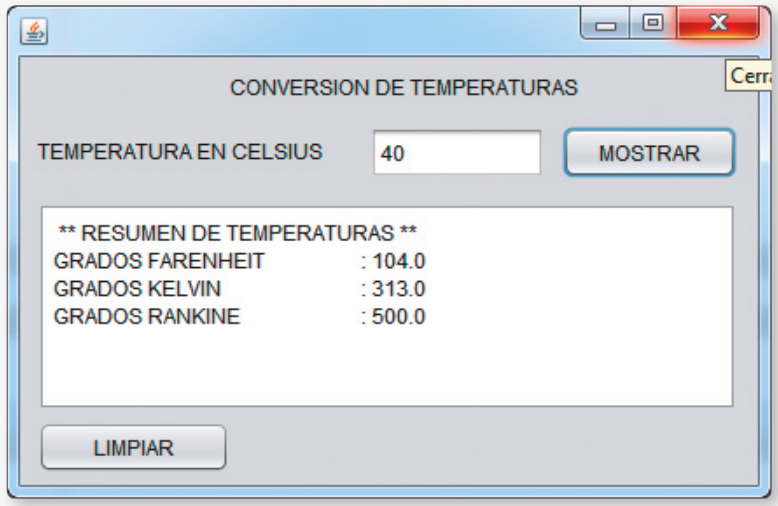


Fig. 4.8

Con JDeveloper

PASO 1:

Crear una nueva aplicación Java Desktop Application en JDeveloper, como lo muestra la Fig. 4.9.

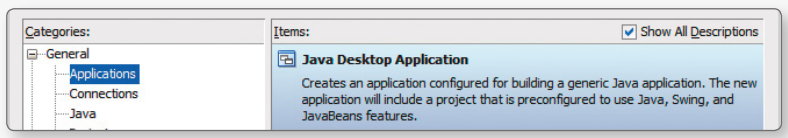


Fig. 4.9

Luego, asignamos el nombre pjSecuencial y se define el lugar destino de nuestra aplicación con el botón Browse… al final debemos presionar Next>

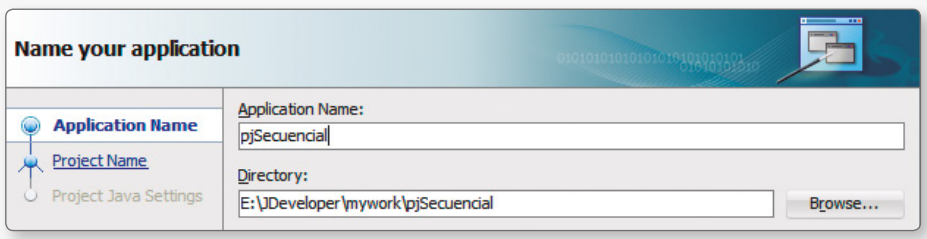


Fig. 4.10

En la siguiente ventana debemos ingresar el nombre del proyecto, en este caso se le asignó pEjercicios. Como lo muestra al Fig. 4.11, luego presionar Finish.

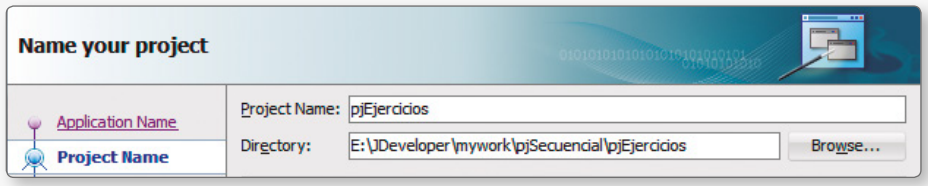


Fig. 4.11

PASO 2:

El navegador de la aplicación debe quedar como lo muestra la Fig. 4.12.

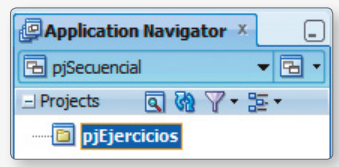


Fig. 4.12

PASO 3:

Presionamos clic derecho sobre el proyecto pjEjercicios > New > Cliente Tier > Swing/AWT > Frame, como lo muestra la Fig. 4.13.

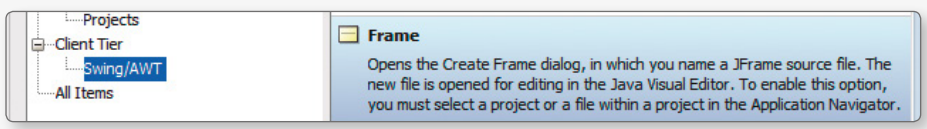


Fig. 4.13

Luego debe asignar un nombre al Frame como lo muestra la Fig. 4.14.

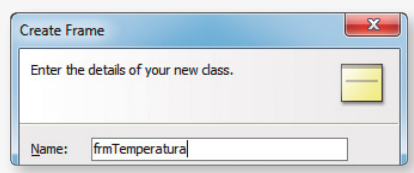


Fig. 4.14

PASO 4:

Diseñe la siguiente GUI:

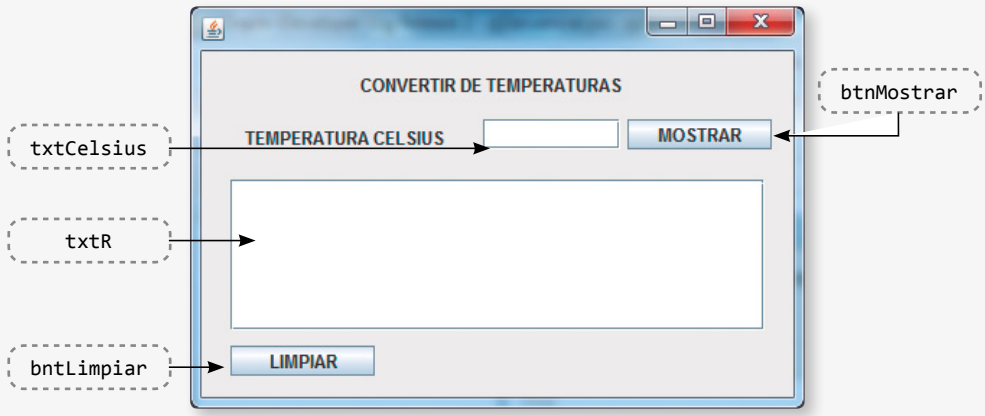


Fig. 4.15

PASO 5:

Coloque el siguiente código dentro del botón btnMostrar.

```
    private void btnMostrar_actionPerformed(ActionEvent e) {
        double celsius=Double.parseDouble(txtCelsius.getText());

        double farenheit=(9 * celsius)/5 + 32;
        double rankine=celsius + 460;
        double kelvin=rankine - 187;

        txtR.setText(" ** RESUMEN DE TEMPERATURAS ** ");
        txtR.append("\nGRADOS FARENHEIT : "+farenheit);
        txtR.append("\nGRADOS KELVIN    : "+kelvin);
        txtR.append("\nGRADOS RANKINE   : "+rankine);
    }
```

Dentro del botón btnLimpiar colocar el siguiente código:

```
        private void btnLimpiar_actionPerformed(ActionEvent e) {
        txtCelsius.setText("");
        txtR.setText("");
        txtCelsius.requestFocus();
    }
```

Luego se debe crear el método void main que permitirá inicializar el formulario a ejecutar en la aplicación:

```
    public static void main(String args[]){
            frmTemperatura fraVista=new frmTemperatura();
            fraVista.setVisible(true);
    }
```

Finalmente, presionamos F11 para ejecutar la aplicación e ingresaremos el valor 40 y presionamos el botón Mostrar. Como lo muestra la Fig. 4.16.

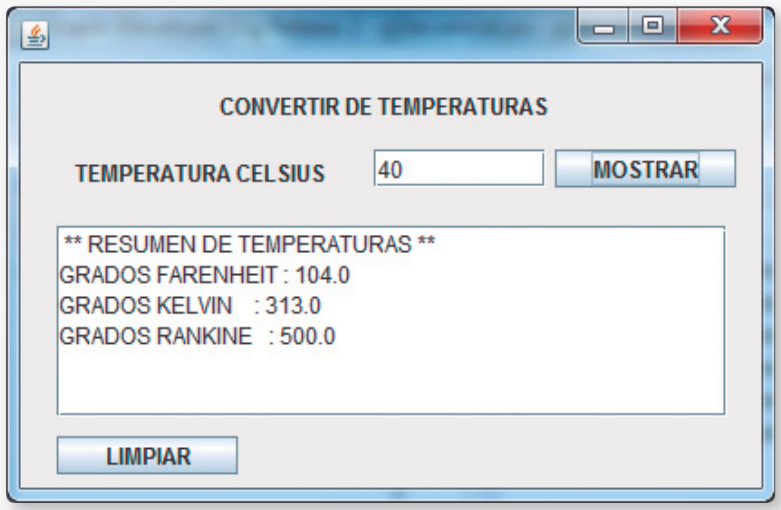


Fig. 4.16

CASO DESARROLLADO 2: CASA DE CAMBIO

Se cuenta con tres cantidades de dinero en soles, dólares y marcos, respectivamente. Diseñe una aplicación que determine el monto total del dinero en euros. Considere los siguientes tipos de cambio:

1 dólar	3.51 soles
1 dólar	1.09 euros
1 dólar	2.12 marcos

Para este caso usaremos JCreator en el proyecto del caso 1 llamado pjSecuencial.

PASO 1:

Clic derecho sobre el proyecto pjSecuencial > Add New Project y coloque el nombre frmCasaCambio > Finish. Como lo muestra la Fig. 4.17.

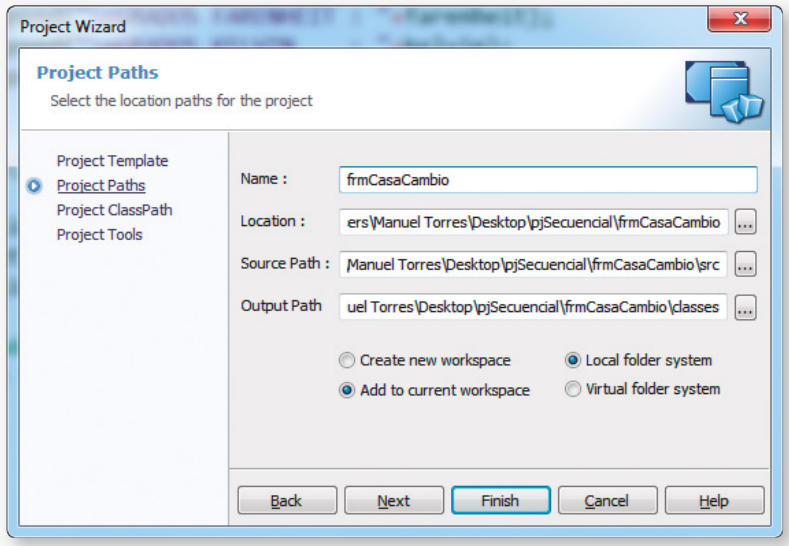


Fig. 4.17

PASO 2 :

Diseñe la GUI como lo muestra la Fig. 4.18
Para esto colocamos el siguiente código dentro de la clase frmCasaCambio:

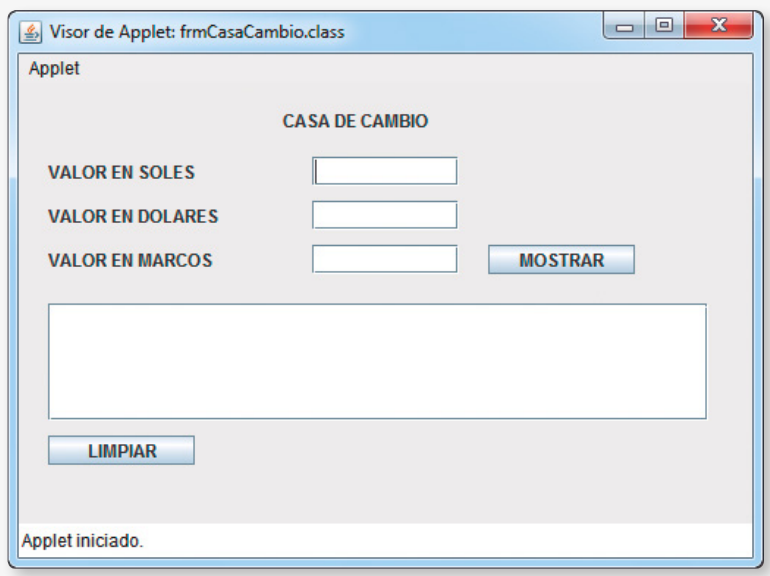


Fig. 4.18

Luego agregaremos el código que permite determinar los resultados de la aplicación:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

public class frmCasaCambio extends JApplet implements ActionListener{
JLabel lblTitulo,lblSoles,lblDolares,lblMarcos;
JTextField txtSoles,txtDolares,txtMarcos;
JButton btnCalcular, btnLimpiar;
JTextArea txtR;
JScrollPane spDesplaza;

public void init() {
    this.setLayout(null);

    lblTitulo = new JLabel("CASA DE CAMBIO");
    lblTitulo.setBounds(180,15,200,20);
    add(lblTitulo);

    lblSoles = new JLabel("VALOR EN SOLES ");
    lblSoles.setBounds(20,50,200,20);
    add(lblSoles);

    txtSoles = new JTextField();
    txtSoles.setBounds(200,50,100,20);
    add(txtSoles);

    lblDolares = new JLabel("VALOR EN DOLARES ");
    lblDolares.setBounds(20,80,200,20);
    add(lblDolares);

    txtDolares = new JTextField();
    txtDolares.setBounds(200,80,100,20);
    add(txtDolares);

    lblMarcos = new JLabel("VALOR EN MARCOS ");
    lblMarcos.setBounds(20,110,200,20);
    add(lblMarcos);

    txtMarcos = new JTextField();
    txtMarcos.setBounds(200,110,100,20);
    add(txtMarcos);

    btnCalcular = new JButton("MOSTRAR");
    btnCalcular.setBounds(320,110,100,20);
    btnCalcular.addActionListener(this);
    add(btnCalcular);

    btnLimpiar = new JButton("LIMPIAR");
    btnLimpiar.setBounds(20,240,100,20);
    btnLimpiar.addActionListener(this);
    add(btnLimpiar);

    txtR = new JTextArea();
    txtR.setFont(new Font("Trebuchet MS", 0, 12));

    spDesplaza = new JScrollPane(txtR);
    spDesplaza.setBounds(20, 150, 450, 80);
    add(spDesplaza);
    }
```

```
public void actionPerformed(ActionEvent e){
if (e.getSource()==btnCalcular){

     double soles=Double.parseDouble(txtSoles.getText());
     double dolares=Double.parseDouble(txtDolares.getText());
     double marcos=Double.parseDouble(txtMarcos.getText());

     double montoTotal=(soles/3.51 + marcos/2.12 + dolares) * 1.09;

     txtR.setText(" ** RESUMEN DE CASA DE CAMBIO ** ");
     txtR.append("\n");
     txtR.append("\nEL MONTO TOTAL EN EUROS ES: "+

String.format("%.2f",montoTotal));
}
if (e.getSource()==btnLimpiar){
     txtSoles.setText("");
     txtDolares.setText("");
     txtMarcos.setText("");
     txtR.setText("");
     txtSoles.requestFocus();
}
 } //Cierra el método actionPerformed
```

Finalmente, presionamos F7 y F5 e ingresaremos los valores en soles, dólares y marcos para poder obtener el monto total en euros:

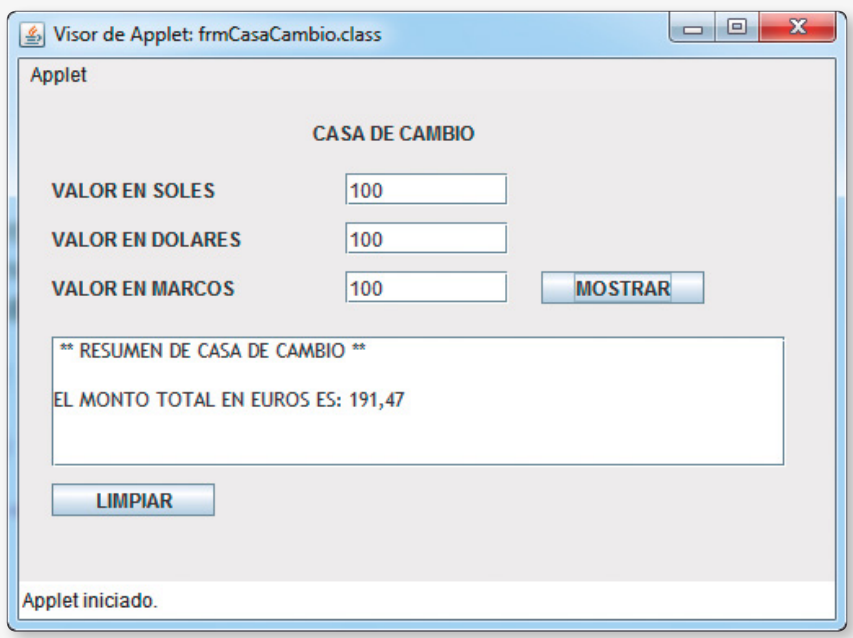


Fig. 4.19

CASO DESARROLLADO 3: MEDIDAS

Implementar una aplicación Java que permita convertir una cantidad especificada en pulgadas a pies, a yardas, a centímetros y a metros, debemos considerar la siguiente tabla:

1 yarda	3 pies
1 pie	12 pulgadas
1 pulgada	2.54 centimetros
1 metro	100 centimetros

PASO 1:

Clic derecho sobre el proyecto pjSecuencial > Add New Project y coloque el nombre frmMedidas > Finish. Como lo muestra la Fig. 4.20.

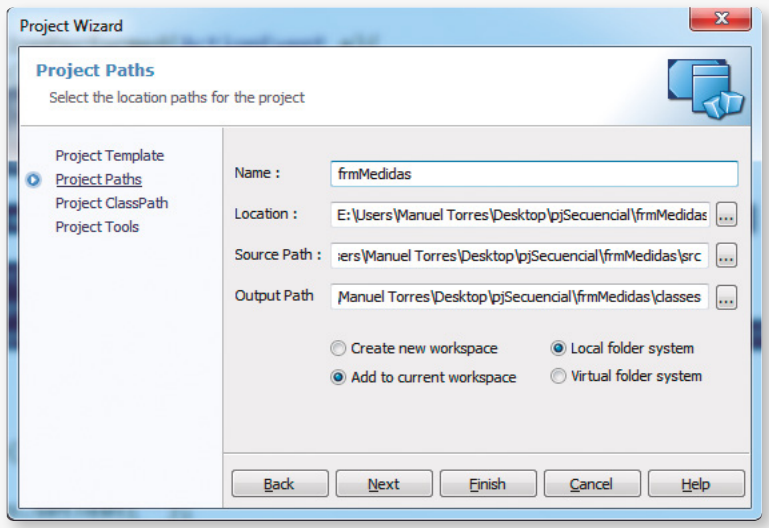


Fig. 4.20

PASO 2:

Diseñe la GUI como lo muestra la Fig. 4.21.

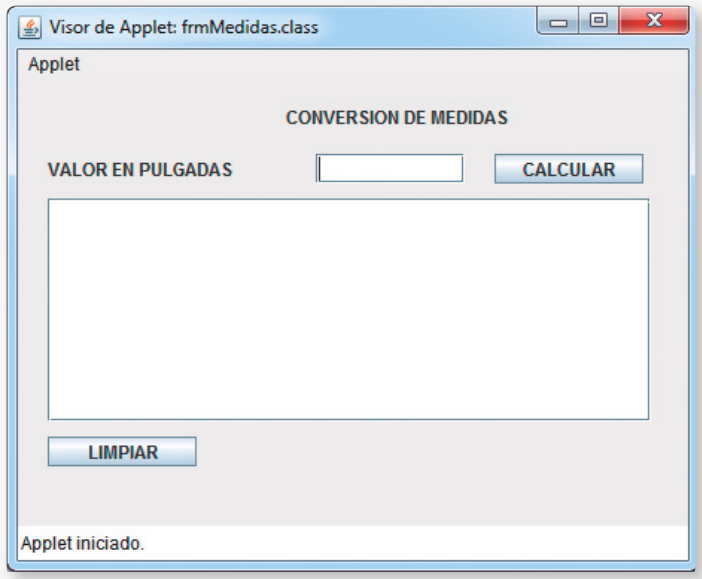


Fig. 4.21

Para esto colocamos el siguiente código dentro de la clase frmMedidas:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

public class frmMedidas extends JApplet implements ActionListener {

    JLabel lblTitulo,lblPulgadas;
    JTextField txtPulgadas;
    JButton btnCalcular, btnLimpiar;
    JTextArea txtR;
    JScrollPane spDesplaza;

    public void init() {

            this.setLayout(null);

            lblTitulo = new JLabel("CONVERSION DE MEDIDAS");
            lblTitulo.setBounds(180,15,200,20);
            add(lblTitulo);

            lblPulgadas = new JLabel("VALOR EN PULGADAS");
            lblPulgadas.setBounds(20,50,200,20);
            add(lblPulgadas);

            txtPulgadas = new JTextField();
            txtPulgadas.setBounds(200,50,100,20);
            add(txtPulgadas);
```

```
            btnCalcular = new JButton("CALCULAR");
            btnCalcular.setBounds(320,50,100,20);
            btnCalcular.addActionListener(this);
            add(btnCalcular);

            btnLimpiar = new JButton("LIMPIAR");
            btnLimpiar.setBounds(20,240,100,20);
            btnLimpiar.addActionListener(this);
            add(btnLimpiar);

            txtR = new JTextArea();
            txtR.setFont(new Font("Trebuchet MS", 0, 12));

            spDesplaza = new JScrollPane(txtR);
            spDesplaza.setBounds(20, 80, 405, 150);
            add(spDesplaza);

      }
```

Luego agregaremos el código que permite determinar los resultados de la aplicación:

```
public void actionPerformed(ActionEvent e){
if (e.getSource()==btnCalcular){
     double pulgadas=Double.parseDouble(txtPulgadas.getText());

     double pies=pulgadas/12;
     double yardas= 3 * pies;
     double centimetros=pulgadas*2.54;
     double metros = centimetros/100;

     txtR.setText(" ** RESUMEN DE MEDIDAS ** ");
     txtR.append("\n");
     txtR.append("\nPIES: "+String.format("%.2f",pies));
     txtR.append("\nYARDAS: "+String.format("%.2f",yardas));
txtR.append("\nCENTIMETROS: "+String.format("%.2f",centimetros));
     txtR.append("\nMETROS: "+String.format("%.2f",metros));
}
if (e.getSource()==btnLimpiar){
     txtPulgadas.setText("");
     txtR.setText("");
     txtPulgadas.requestFocus();
}
   } //Cierra el método actionPerformed
}
```

Finalmente, presione F7 y F5 e ingrese el valor en pulgadas para mostrar los resultados mediante el botón Calcular:

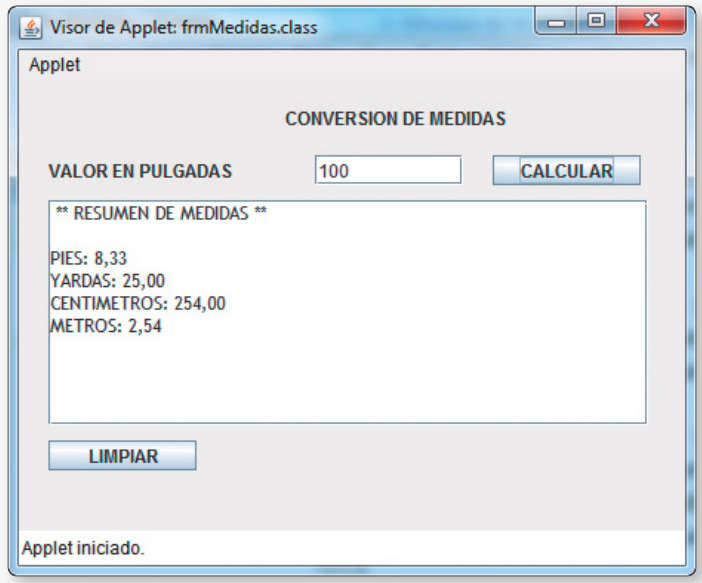

Visor de Applet: frmMedidas.class
Applet
CONVERSION DE MEDIDAS
VALOR EN PULGADAS
100
CALCULAR
** RESUMEN DE MEDIDAS **
PIES: 8,33
YARDAS: 25,00
CENTIMETROS: 254,00
METROS: 2,54
LIMPIAR
Applet iniciado.

Fig. 4.22

CAP.

5

Estructura de selección

CAPACIDAD:

- Implementar aplicaciones usando la sentencia if de forma simple, doble y doblemente enlazada.
- Implementar aplicaciones usando la sentencia switch.

CONTENIDO:

5.1. INTRODUCCIÓN

Las aplicaciones con desarrollo secuencial es de gran utilidad cuando una aplicación así lo requiera. Hay que tener en cuenta que en un determinado momento la aplicación requerirá un número de posibles alternativas de solución a un caso particular.

De allí se desprenden las estructuras de decisión o también llamadas de selección, alternativas o condicional, que se utilizan para tomar decisiones lógicas mediante la evaluación de una condición y en función del resultado de realizarán las instrucciones en una aplicación.

¿A quién no le ha pasado estar con su auto y encontrarse en una carretera bifurcada? Si toma la decisión de ir por un determinado camino no sabrá qué habría pasado por el otro camino o a dónde lo hubiese conducido, así que tomar el camino correcto será su decisión lógica.

Veamos el algoritmo clásico que muestra cómo se aplica una sentencia condicional; el caso será cambiar un foco en mal estado:

Lo primero que piensa es:

1.- Sacar el foco malogrado

2.- Colocar el foco nuevo

Genéricamente es correcto, pero esto genera una serie de cuestiones como por ejemplo: ¿Cómo hago para sacar el foco?, ¿El foco es el correcto?, etc. Para esto podría ser más específico en los pasos:

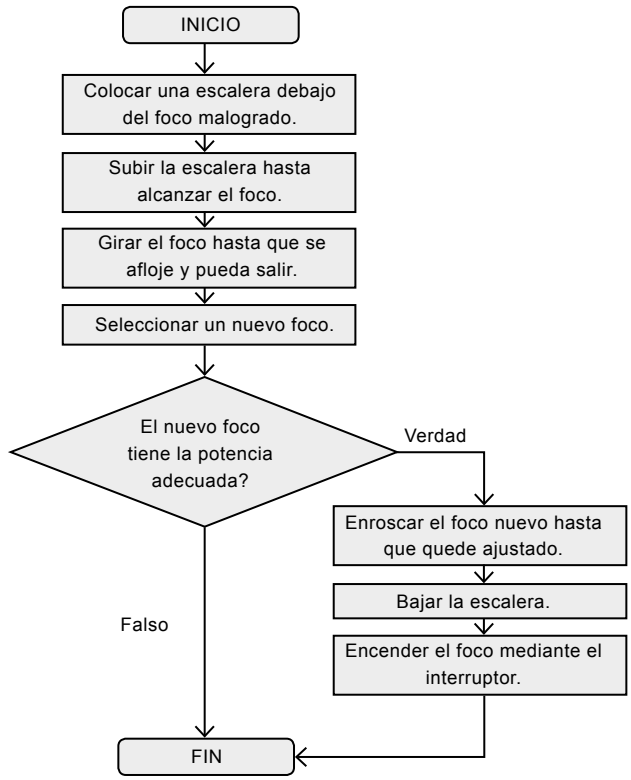
1. Sacar el foco malogrado
- **1.1.** Colocar una escalera debajo del foco malogrado
- **1.2.** Subir la escalera hasta alcanzar el foco
- **1.3.** Girar el foco hasta que se afloje y pueda salir
- **1.4.** Colocar el nuevo foco

2. Seleccionar un nuevo foco
- **2.1.** ¿El nuevo foco tiene la potencia adecuada?
- **2.2.** Enroscar el foco nuevo hasta que quede ajustado
- **2.3.** Bajar la escalera

2.4. Encender el foco mediante el interruptor

3. Terminar el caso

Vea la gráfica que representa el mismo caso:



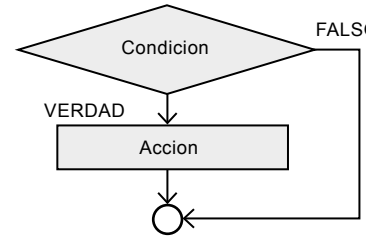
5.2. Sentencia if simple

La sentencia if en una aplicación Java permite tomar una decisión para ejecutar una o más acciones dependiendo del resultado de la condición verdadero o falso. Se le llama simple porque solo evalúa un lado de la condición, es decir, se evalúa la lógica verdadera.

Formato:

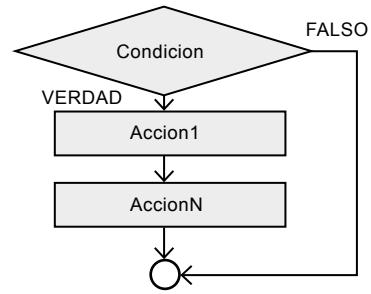
- If de una sola acción:

```
if (Condicion) accion;

O también

if (Condicion)
   accion;
```



- If de varias acciones

```
if (Condicion) {
    Accion1;
    Accion2;
    Accion3;
    …
    AccionN;
}
```



Una sentencia if simple se ejecuta de la siguiente forma:

- Se evalúa la condición especificada, esta tendrá que usar los operadores lógicos y relacionales necesarios. Dicha condición emitirá un resultado verdadero o falso según la condición implementada, recuerde que if simple solo evalúa la verdad.
- Si el resultado de la condición es verdadero entonces se ejecutará la acción implementada.
- Si el resultado de la condición es falsa no se ejecuta ninguna acción y se continúa con la instrucción siguiente a la expresión. Vea el siguiente script:

```
double sueldo=1500
double descuento=0;
double AFP=0;
if (sueldo>2500)
    descuento = 0.15*sueldo;
AFP=0.09*sueldo;
```

¿Cuál es el valor asignado a las siguientes variables después de la condicional?

- Sueldo > 1500.00
- Descuento > 0.00
- AFP > 135.00 (9% del sueldo)

Si evalúa el código verá que la variable sueldo tiene un valor inicial de 1500, el descuento de 0 y AFP también con 0. Al entrar en la evaluación sueldo no supera a 2500 de la condición; por lo tanto, el resultado es falso así es que no hay descuento. El valor asignado a descuento después de la evaluación es el mismo que tenía asignado antes de entrar a la condicional, es decir, cero. Y finalmente AFP tiene un valor de 135 al multiplicarse 0.09*1500 esta expresión no está en condición, a pesar de que esta debajo de la expresión de descuento que si está condicionada. Vea la diferencia del script si colocamos unas llaves:

```
double sueldo=1500
double descuento=0;
double AFP=0;
if (sueldo>2500){
    descuento = 0.15*sueldo;
AFP=0.09*sueldo;
}
```

¿Cuál es el valor asignado a las siguientes variables después de la condicional?

- Sueldo > 1500.00
- Descuento > 0.00
- AFP > 0.00

Si observamos la condición sigue siendo falsa, pero las expresiones se encuentran encerradas con llaves; por tanto, solo serán accesibles cuando la condición sea verdadera, en este caso los valores para descuento y AFP serán los mismos definidos en la declaración.

5.3. ¿Cómo implementar una condición lógica?

La parte más importante de la sentencia if es componer una condición lógica adecuada, se verá mediante casos como implementarlas, para esto se usarán los operadores de comparación y lógicos, descritos en el capítulo 2 referente a fundamentos de programación.

Caso 1

Una empresa renta autos de tres clases: pequeños, medianos y grandes. La tarifa del alquiler por día es: $ 15 en el auto pequeño, $ 20 en el auto mediano y $ 30 en el auto grande. Además la empresa cobra $ 0.20/km recorrido en auto pequeño, $ 0.30/km recorrido en auto mediano y $ 0.40/km recorrido en auto grande. Si la cantidad de km recorridos por el auto supera los 10 km por día se le aumentara un 2.5% sobre el monto a pagar por el cliente.

Solución:

¿Cómo determinar la tarifa de alquiler por dia según el tipo de auto?	if (auto.equals("pequeño"))
	if (auto.equals("mediano"))
	if (auto.equals("grande"))

¿Cómo determinar si la cantidad de km supera a 10?	if (kilometros>10)
	
	O
	
	if (kilometros>=11)

Caso 2

Determinar la cantidad de dinero que recibirá un trabajador por concepto de las horas extras trabajadas en una empresa, sabiendo que cuando las horas de trabajo exceden de 40, el resto se consideran horas extras y que éstas se pagan al doble de una hora normal cuando no exceden de 8; si las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se paga por una hora normal y el resto al triple.

Solución:

¿Cómo determinar las horas extras?	if (horas>=40) o if (horas>39)
¿Cómo determinar si exceden las 8 horas extras?	if (horasExtras>=8) o if (horasExtras>7)

CASO 3

Aplicación que permita emitir una factura correspondiente a la venta de un artículo del cual se adquieren varias unidades. El IVA a aplicar es del 12% y si el monto total (precio de ventas + IVA) es menor o igual a $30.00 se aplicará un 10% de descuento, si el monto total es mayor a $30.00 pero menor a $70.00 se le aplicará un 25% y si el monto total es superior a $70.00 entonces se le aplicará un descuento del 35%.

Solución:

¿Cómo determinar si el monto total es menor o igual a $30.00?	if (montoTotal<=30) o if (montoTotal<31)
¿Cómo determinar si el monto total es mayor a $30.00 pero menor a $70.00?	if (montoTotal>=30 && montoTotal<=70) if (montoTotal>29 && montoTotal<71)
¿Cómo determinar si el monto total es mayor a $70.00?	if (montoTotal>70)

CASO 4

Implementar una aplicación que permita agregar una letra para representar la calificación de los estudiantes, las calificaciones son notas entre 1 y 10; use los siguientes parámetros: A para calificaciones mayores o iguales a 9, B para calificaciones mayores o iguales a 8, C para calificaciones mayores o iguales a 7, D para calificaciones mayores o iguales a 6 y F para todas las demás calificaciones.

Solución:

¿Cómo determinar si la calificacion es mayor o igual a 9?	if (calificacion>=9) if (calificacion>=9 && calificacion<=10)
¿Cómo determinar si la calificacion es mayor o igual a 8?	if (calificacion>=8 && calificacion<9)
¿Cómo determinar si la calificacion es mayor o igual a 7?	if (calificacion>=7 && calificacion<8)
¿Cómo determinar si la calificacion es mayor o igual a 6?	if (calificacion>=6 && calificacion<7)
¿Cómo determinar si la calificacion es menor a 6?	if (calificacion<6)

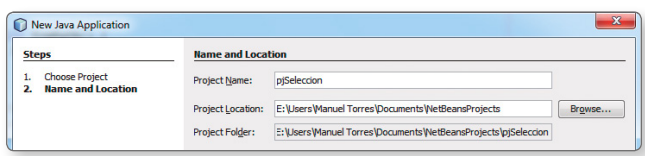
CASO DESARROLLADO 1: RENTA DE AUTOS

Una empresa renta autos de tres clases: pequeños, medianos y grandes. La tarifa del alquiler por día es: $ 15 en el auto pequeño, $ 20 en el auto mediano y $ 30 en el auto grande. Además la empresa cobra $ 0.20/km recorrido en auto pequeño, $ 0.30/km recorrido en auto mediano y $ 0.40/km recorrido en auto grande. Si la cantidad de km recorridos por el auto supera los 10 km se le aumentara un 2.5% sobre el monto a pagar por el cliente.

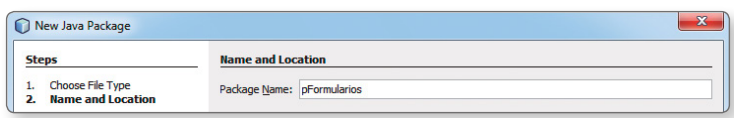
GUI Propuesto:

Debe considerar:

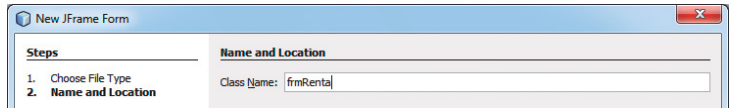
- Crear un nuevo proyecto en NetBeans llamado pjSeleccion.



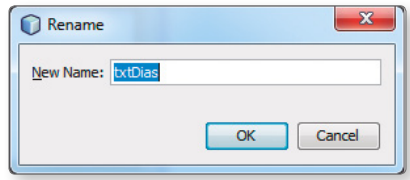
- Agregar un nuevo paquete al proyecto pjSeleccion llamado pFormularios.



- Agregar un nuevo Formulario al paquete pFormulario llamada frmRenta.



- Asigne nombres a todos los controles, para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…



- Se debe agregar un control ButtonGroup para agrupar los botones rbPequeño, rbMediano y rbGrande. Arrastre el control Button Group al Frame, le asignaremos un nombre a dicho control presionando clic derecho sobre el control buttonGroup1 desde el panel Navigator y asígnele el nombre bgClases.

- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la Fig. 5.2.

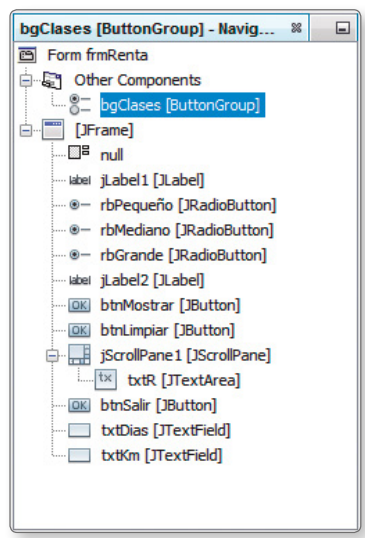


Fig. 5.2

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el Form Size Police con el valor Generate Resize Code.
- Asigne la siguientes propiedades a los controles:

txtDias	**Border** **Text**	TitledBorder=Dias Alquilados Dejar vacio
txtKm	**Border** **Text**	TitledBorder=Kilometros recorridos Dejar vacio
txtR	**Border** **Text**	TitledBorder=Resumen de la Renta Dejar vacio
rbPequeño	**ButtonGroup** **Text**	bgClases Pequeño
rbMediano	**ButtonGroup** **Text**	bgClases Mediano
rbGrande	**ButtonGroup** **Text**	bgClases Grande
btnMostrar	**Text**	MOSTRAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Veamos el script de botones

En el siguiente script se muestra como el método constructor de la clase principal inicializa el botón btnMostrar como el botón por defecto y el rbPequeño como control activo.

```
public frmRenta() {
initComponents();
getRootPane().setDefaultButton(btnMostrar);
rbPequeño.isSelected();
}
```

El siguiente script muestra el código que se le asigna al botón btnMostrar como botón de respuesta al ingreso de valores en el Frame.

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
//1
int dias = Integer.parseInt(txtDias.getText());
double km = Double.parseDouble(txtKm.getText());

//2
double tarifa=0.0;
if (rbPequeño.isSelected()) tarifa=15;
if (rbMediano.isSelected()) tarifa=20;
if (rbGrande.isSelected()) tarifa=30;

//3
double costo=0.0;
if (rbPequeño.isSelected()) costo=0.2*km;
if (rbMediano.isSelected()) costo=0.3*km;
if (rbGrande.isSelected()) costo=0.4*km;

//4
double monto=(tarifa*dias)+costo;

double aumento=0;
if (km>10) aumento=monto*(25/100.0);
```

```
//5
monto=monto+aumento;

//6
txtR.setText("");
txtR.append("La cantidad de dias es: "+dias);
txtR.append("\nLa tarifa asignada es: $ "+
                                String.format("%.2f",tarifa));
txtR.append("\nEl costo por Km es: $ "+
                                String.format("%.2f",costo));
txtR.append("\nEl Monto a Pagar es: $ "+
                                String.format("%.2f",monto));
}
```

En el punto uno se declara la variable días de tipo entero y es asignada con el valor ingresado desde el control txtDias, a la variable km de tipo decimal doble se le asigna el valor ingresado desde el control txtkm.

En el punto dos se asigna la tarifa según la clase seleccionada por el usuario, para esto declaramos la variable tarifa de tipo decimal doble e inicializado en 0 por no tener un valor por defecto en la evaluación. Luego se comprueba qué radio está activa, para esto usamos el método isSelected.

En el punto tres se asigna el costo por kilómetro recorrido según la clase de auto seleccionada. Para esto se declara la variable costo de tipo decimal doble y se le asigna el valor 0 por no tener un valor por defecto en la evaluación. Luego se comprueba la clase de auto seleccionada por el usuario y se calcula el costo según los datos del caso.

En el punto cuatro se calcula el monto total a pagar por el cliente. para esto se multiplica la tarifa y los días de renta más el costo calculados por los kilómetros recorridos. Allí mismo se calcula el aumento a dicho monto solo si los kilómetros sobrepasan los 10.

En el punto cinco se asigna al monto el nuevo valor adicionando, el aumento al monto actual. En el punto seis se imprimen las respuestas obtenidas dentro del control txtR.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt)
{
    txtDias.setText("");
    txtKm.setText("");
    txtR.setText("");
    txtDias.requestFocus();
}
```

En el siguiente script se muestra como salir de la aplicación preguntando al usuario si está seguro de salir; dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
//
int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Sistema de Ventas",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

CASO DESARROLLADO 2: PAGO DE TRABAJADORES

Determinar la cantidad de dinero que recibirá un trabajador por concepto de las horas extras trabajadas en una empresa, sabiendo que cuando las horas de trabajo exceden de 40, el resto se consideran horas extras y que estas se pagan al doble de una hora normal cuando no exceden de 8; si las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se paga por una hora normal y el resto al triple.

GUI Propuesto:

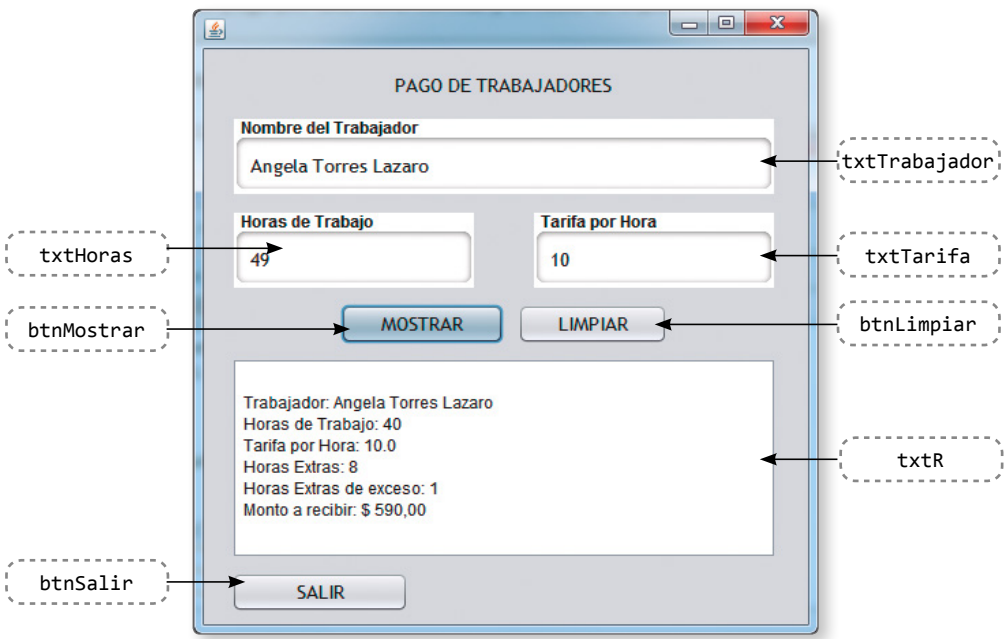


Fig. 5.3

Debe considerar:

- Agregar al proyecto pjSeleccion un Frame llamado frmPago.

- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la siguiente Fig. 5.4:

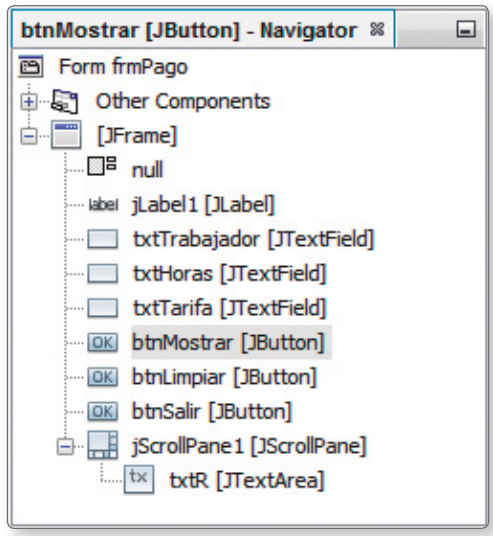


Fig. 5.4

- Asignar **Null Layout** al **setLayout** del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtTrabajador	**Border**	TitledBorder=Nombre del Trabajador
	Text	Dejar vacio
txtHoras	**Border**	TitledBorder=Horas de Trabajo
	Text	Dejar vacio
txtTarifa	**Border**	TitledBorder=Tarifa por Hora
	Text	Dejar vacio
txtR	**Text**	Dejar vacio
btnMostrar	**Text**	MOSTRAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Veamos el script de botones:

En el siguiente script se muestra el método constructor de la clase principal en donde se inicializa el botón btnMostrar como el botón por defecto.

```
public frmPago() {
initComponents();
getRootPane().setDefaultButton(btnMostrar);
}
```

El siguiente script muestra el código que se le asigna al botón btnMostrar como botón de respuesta al ingreso de valores en el Frame.

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
//1
String trabajador=txtTrabajador.getText();
double tarifa=Double.parseDouble(txtTarifa.getText());
int horas=Integer.parseInt(txtHoras.getText());

//2
int extras=0;
if (horas>40) {
     extras=horas-40;
     horas=40;
}

//3
int excede=0;
if (extras>8) {
     excede=extras-8;
     extras=8;
}

//4
double monto=horas*tarifa+extras*(tarifa*2)+excede*(tarifa*3);

//5
txtR.setText("");
txtR.append("\nTrabajador: "+trabajador);
txtR.append("\nHoras de Trabajo: "+horas);
txtR.append("\nTarifa por Hora: "+tarifa);
txtR.append("\nHoras Extras: "+extras);
txtR.append("\nHoras Extras de exceso: "+excede);
txtR.append("\nMonto a recibir: $ "+
                                    String.format("%.2f",monto));
}
```

En el punto uno se declara la variable trabajador y se le asigna el valor ingresado, así mismo para tarifa y las horas de trabajo, estas dos últimas con sus respectivos parseos.

En el punto dos se calculan las horas extras del trabajador, para esto se evalúan las horas y de acuerdo a ello se le asigna a la variable extra la diferencia entre las horas de trabajo y 40, y a la vez asignar directamente a 40 las horas de trabajo, puesto que las horas extras son pagadas de otra forma. Como verá por tener dos expresiones para la condicional se tiene que encerrar entre llaves.

En el punto tres se determina si las horas extras no sobrepasan a 8, en caso sea así se calcula el valor de exceso y se le asigna directamente 8 a la variable extras. Cabe resaltar que si extras no sobrepasa el límite de 8 seguirá con su valor original.

En el punto cuatro se calcula el monto multiplicando y sumando las horas extras y las horas de exceso si las hubiera. Se estará preguntando qué pasaría si no hay horas extras y, por tanto; tampoco horas de exceso, pues esas variables tienen una asignación inicial de 0 que aquí serán participes.

En el punto cinco se imprimen los resultados obtenidos en las instrucciones anteriores.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt)
{
    txtTrabajador.setText("");
    txtHoras.setText("");
    txtTarifa.setText("");
    txtR.setText("");
    txtTrabajador.requestFocus();
    }
```

En el siguiente script se muestra como salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
 private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
 int r=JOptionPane.showOptionDialog(this,
             "Estas seguro de salir...?",
             "Control de Pagos",
             JOptionPane.YES_NO_OPTION,
             JOptionPane.QUESTION_MESSAGE,
             null,null,null);
 if (r==0) System.exit(0);
 }
```

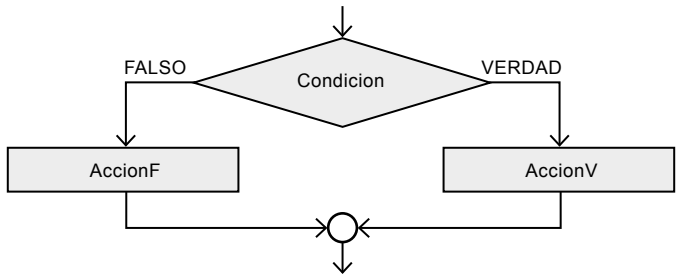
5.4. SENTENCIA IF DOBLE

La sentencia if doble en una aplicación Java permite tomar una decisión para ejecutar una o más acciones dependiendo del resultado de la condición verdadero o falso. Se le llama doble porque si la condición es verdadera ejecuta las acciones programadas en el lado verdadero, en caso sea falso ejecuta las acciones programadas en el lado falso.

Formato:

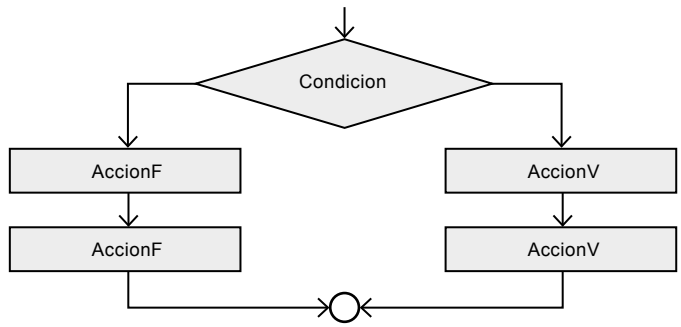
- If doble de una sola acción:

```
if (Condicion) accionV; else accionF;

O también

if (Condicion)
   accionV;
  else
  accionF;
```



- If doble de varias acciones

```
if (Condicion) {
   AccionV1;
   AccionV2;
   AccionV3;
   …
   AccionVN;
} else {
   AccionF1;
   AccionF2;
   AccionF3;
   …
   AccionFN;
}
```



Una sentencia if doble se ejecuta de la siguiente forma:

- Se evalúa la condición y dependiendo del resultado se ejecutará una u otra opción.
- Si el resultado de la condición es falsa se ejecutará acciones implementadas para dicha condición, también es conocida como acciones por defecto.
- Cuando una variable se encuentra dentro de una condicional doble recibirá la asignación automática solo si se define dentro del bloque else, esto quiere decir que no necesita inicialización. Veamos el siguiente script:

```
double sueldo;
String categoria="B";
if (categoria.equals("A"))
     sueldo=2100;
     else
     sueldo=2500;
```

¿Cuál es el valor asignado sueldo después de la evaluación en la condición doble?

- Sueldo > 2500.00

Veamos un script errado:

```
double sueldo;
String categoria="B";
if (categoria.equals("A")) sueldo=2500;
if (categoria.equals("B")) sueldo=2100;
```

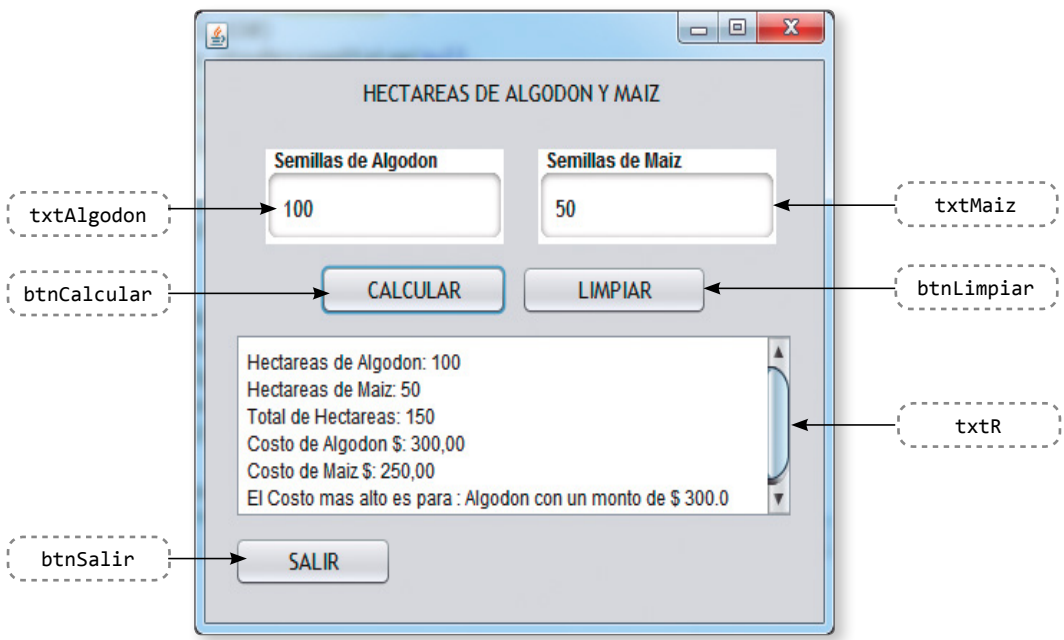
Si observamos la variable sueldo solo se asignará un valor cuando la categoría sea A o B, pero igualmente genera un error de inicialización con respecto a la variable sueldo, para que no genera el error el código script debe ser de la siguiente forma:

```
double sueldo=1500;
String categoria="B";
if (categoria.equals("A")) sueldo=2500;
if (categoria.equals("B")) sueldo=2100;
```

CASO DESARROLLADO 3: HECTÁREAS DE ALGODÓN Y MAÍZ

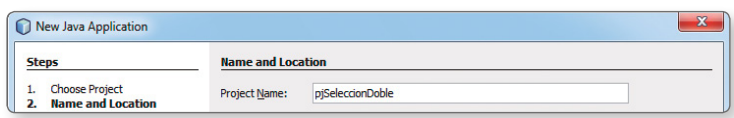
Un granjero tiene 150 hectáreas disponibles para plantar maíz y algodón. Las semillas de algodón le cuestan $ 3.00 por hectárea y las de maíz $ 5.00 por hectárea. Implemente una aplicación que permita determinar cuál es el costo que invertirá el granjero por la compra de cierta cantidad de hectáreas de algodón y maíz, sabiendo que la suma de las hectáreas no debe exceder a 150 hectáreas. Además, deberá determinar en qué semilla se invirtió más dinero.

GUI Propuesto:

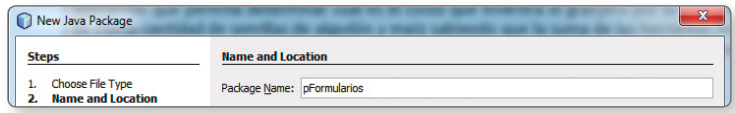


Debe considerar:

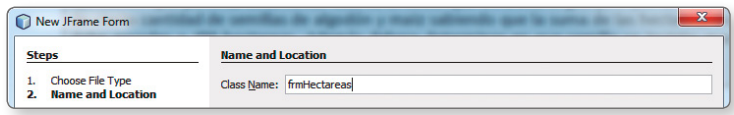
- Crear un nuevo proyecto en NetBeans llamado pjSeleccionDoble.



- Agregar un nuevo paquete al proyecto pjSeleccion llamado pFormularios.



- Agregar un nuevo Formulario al paquete pFormulario llamada frmHectareas



- Asigne nombres a todos los controles para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…
- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la siguiente Fig. 5.5.

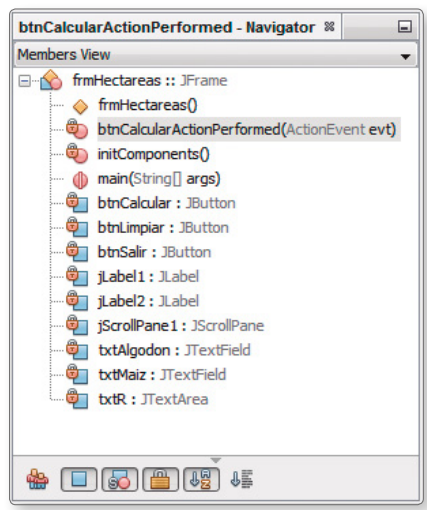


Fig. 5.5

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtAlgodon	**Border**	TitledBorder=Semillas de Algodon
	Text	Dejar vacio
txtMaiz	**Border**	TitledBorder=Semillas de Maiz
	Text	Dejar vacio
txtR	**Text**	Dejar vacio
btnCalcular	**Text**	CALCULAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de botones:

El siguiente script muestra el código que se le asigna al botón btnCalcular como botón de respuesta al ingreso de valores en el Frame.

```
private void btnCalcularActionPerformed(…) {
//1
int algodon=Integer.parseInt(txtAlgodon.getText());
int maiz=Integer.parseInt(txtMaiz.getText());

//2
double costoAlgodon=0,costoMaiz=0;
if (algodon+maiz>150)
     JOptionPane.showMessageDialog(null,
     "Modifique los valores para que no exceda a 150 hectareas",
     "Sistema de control",
     JOptionPane.ERROR_MESSAGE);
   else{
     //3
     costoAlgodon=3*algodon;
     costoMaiz=5*maiz;

     //4
     String masAlto;
     double costoAlto;
     if (costoAlgodon>costoMaiz){
         masAlto="Algodon";
         costoAlto=costoAlgodon;
       }else {
         masAlto="Maiz";
         costoAlto=costoMaiz;
     }

     //5
     txtR.setText("");
     txtR.append("Hectareas de Algodon: "+algodon);
     txtR.append("\nHectareas de Maiz: "+maiz);
     txtR.append("\nTotal de Hectareas: "+(algodon+maiz));
     txtR.append("\nCosto de Algodon $: "+
                 String.format("%.2f",costoAlgodon));
     txtR.append("\nCosto de Maiz $: "+
                 String.format("%.2f",costoMaiz));
     txtR.append("\nEl Costo mas alto es para : "+masAlto+
                 " con un monto de $ "+costoAlto);
}
```

En el punto uno se declara las variables algodón y maiz y se inicializan con los valores ingresados en los controles txtAlgodon y txtMaiz respectivamente, tenga en cuenta que estos valores representan a hectáreas.

En el punto dos se evalúa el acumulado entre las hectáreas de algodón y maíz, pues si superan a 150 entonces se mostrará un mensaje de error al usuario. Como verá en la condición se suman los valores y allí mismo se evalúa con 150, en caso el acumulado supere los 150 el mensaje a mostrar se visualiza en la Fig. 5.6.

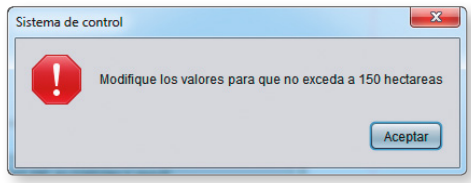


Fig. 5.6

En el punto tres se trata de instrucciones que se ejecutarán solo si la condición especificada fuera falsa, es decir, el acumulado de las hectáreas no supera los 150, entonces calculamos el costo del algodón (costoAlgodon) y el costo de maíz (costoMaiz).

En el punto cuatro se evalúa ambos costos y se determinara quién de los dos tiene el costo más alto, para esto se declara la variable masAlto que tiene por misión almacenar el texto algodón o maíz, dependiendo de quién tiene el más alto costo. Como verá tanto en las instrucciones verdaderas como las falsas se usan las mismas variables ya que solo uno cumple con la condición especificada.

En el punto cinco se imprimen los resultados obtenidos en las instrucciones anteriores.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
        txtAlgodon.setText("");
        txtMaiz.setText("");
        txtR.setText("");
        txtAlgodon.requestFocus();
}
```

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Control de Hectareas",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

CASO DESARROLLADO 4: FIESTA DE 15 AÑOS

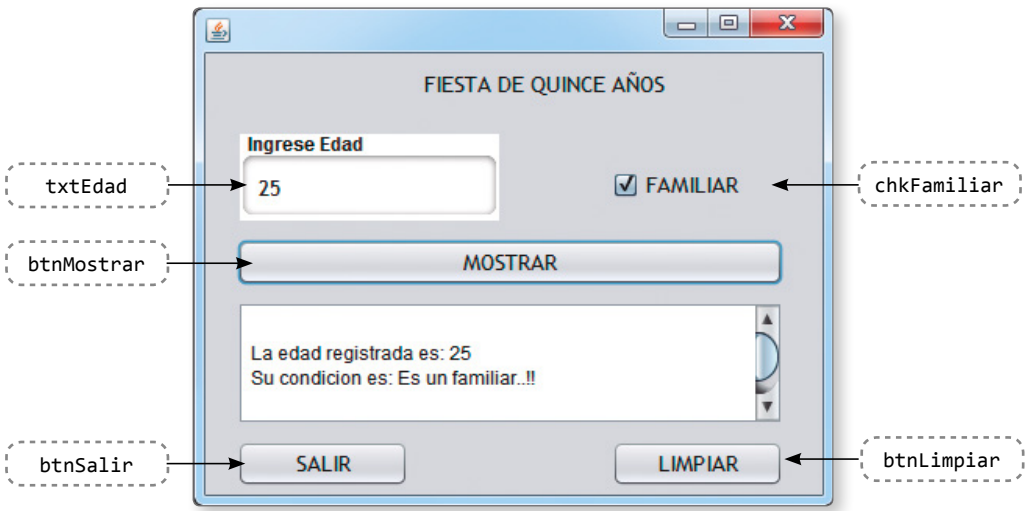
Martha va a organizar la fiesta de quince años de su hija. Para lo cual ha invitado a una gran cantidad de personas entre familiares y amigos. Pero también ha decidido algunas reglas:

Todas las personas con edades mayores o iguales a los quince años, solo pueden entrar si traen regalos.

Jóvenes menores a quince años entran gratis a la fiesta.
Si es un familiar no tiene restricciones de entrada.

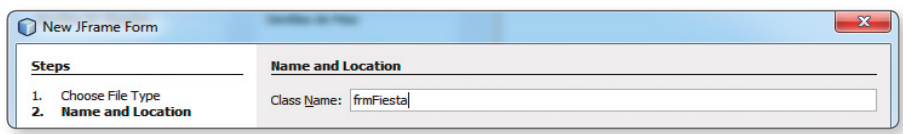
Implemente una aplicación que permita ingresar la edad de la persona invitada y determine si puede o no pasar a la fiesta.

GUI Propuesto:



Debe considerar:

- Agregar al proyecto pjSeleccion un Frame llamado frmFiesta.



- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la Fig. 5.7.

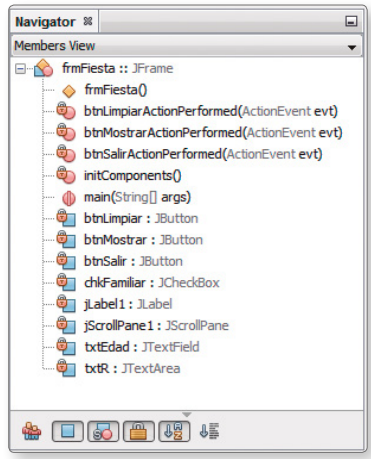


Fig. 5.7

- Asignar **Null Layout** al **setLayout** del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtEdad	**Border**	TitledBorder=Nombre del Trabajador
	Text	Dejar vacio
chkFamiliar	**Text**	FAMILIAR
txtR	**Text**	Dejar vacio
btnMostrar	**Text**	MOSTRAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de botones:

El siguiente script muestra el código que se le asigna al botón btnMostrar como botón de respuesta al ingreso de valores en el Frame.

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
//1
int edad=Integer.parseInt(txtEdad.getText());

//2
String condicion;
if (chkFamiliar.isSelected())
    condicion="Es un familiar..!!";
    else
    if (edad>=15)
        condicion="Regalo";
        else
        condicion="Entran gratis..!!";

//3
txtR.setText("");
txtR.append("\nLa edad registrada es: "+edad);
txtR.append("\nSu condicion es: "+condicion);
}
```

En el punto uno se declara la variable edad y se le asigna el valor ingresado en la aplicación.

En el punto dos se evalúa el control chkFamiliar, si es seleccionado se le asignará a la variable condición de tipo String el valor Es un Familiar…!! Caso contrario se evaluará si la edad es igual o mayor a 15, en caso suceda así se le asigna a condición el valor de Regalo, caso contrario, es decir, es menor a 15 años entonces el mensaje es Entran gratis. Hay que tener en cuenta que dentro de un else se ha implementado un if y que solo se activarán las instrucciones cuando la persona que ingresa su edad no es un familiar.

En el punto tres se imprimen los valores resultantes de la aplicación.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar los valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
txtEdad.setText("");
txtR.setText("");
chkFamiliar.setSelected(false);
txtEdad.requestFocus();
}
```

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Fiesta de Quince Años",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

5.5. LA CLASE JCOMBOBOX

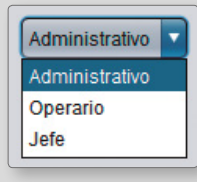
Clase que permite crear un objeto tipo cuadro combinado.

Métodos Constructores:

JCOMBOBOX()	Construye un objeto de tipo JComboBox vacio. Ejm: `JComboBox cboCategoria = new JComboBox();`

Métodos:

addItemListener()	Permite agregar a la lista contenedora de eventos al seleccionar un elemento de la lista contenida en el JComboBox.

addItem(elemento)	Permite agregar un nuevo elemento al contenedor JComboBox.
	
	Ejm: Si tenemos las siguientes categorías de empleados
	
	 
	
	`cboCategoria.addItem("Administrativo");`
	`cboCategoria.addItem("Operario");`
	`cboCategoria.addItem("Jefe");`

getItemCount()	Determina el total de items contenidos dentro del control JComboBox.
	
	Ejm: Mostrar el numero de elementos del control cboCategoria.
	
	`int n=cboCategoria.getItemCount();`
	`JOptionPane.showMessageDialog(null,`
	`"Numero de elementos es: "+n);`

getSelectedIndex()	Permite retornar el numero de indice del elemento seleccionado desde el control JComboBox.
	
	Ejm: Determinar la posicion del elemento seleccionado en el control cboCategoria.
	
	`int pos=cboCategoria.getSelectedIndex();`
	`JOptionPane.showMessageDialog(null,`
	`"La posicion del elemento es: "+pos);`

getSelectedItem()	Permite retornar el elemento seleccionado desde el control JComboBox.
	
	Ejm: Mostrar el elemento seleccionado desde el cuadro combinado cboCategoria.
	
	`String categoria=(String)cboCategoria.getSelectedItem();`
	`JOptionPane.showMessageDialog(null,`
	`"La categoria seleccionada es: "+categoria);`
	
	O también:
	
	`String categoria=String.valueOf(cboCategoria`
	`.getSelectedItem());`
	`JOptionPane.showMessageDialog(null,`
	`"La categoria seleccionada es: "+categoria);`

setBounds(x,y, ancho,alto)	Permite posicionar el objeto JComboBox dentro del control JFrame.
	
	Ejm:
	
	`cboCategoria.setBounds(15, 15, 90, 23);`
	
	`o en su defecto usar:`
	
	`cboCategoria.setLocation(15,15);`
	`cboCategoria.setSize(90,23);`

SETEDITABLE()	Establece la edicion sobre el control JComboBox. Ejm: Habilitar la edicion sobre el control cboCategoria. `cboCategoria.setEditable(true);`
SETENABLED()	Permite establecer la habilitacion o no del control JComboBox. Ejm: Inhabilitar el control cboCategoria. `cboCategoria.setEnabled(false);`

SETVISIBLE()	Permite mostrar u ocultar el control JComboBox.
	
	Ejm:
	
	`Mostrar > cboCategoria.setVisible(True);`
	`Ocultar > cboCategoria.setVisible(False);`

Opciones:

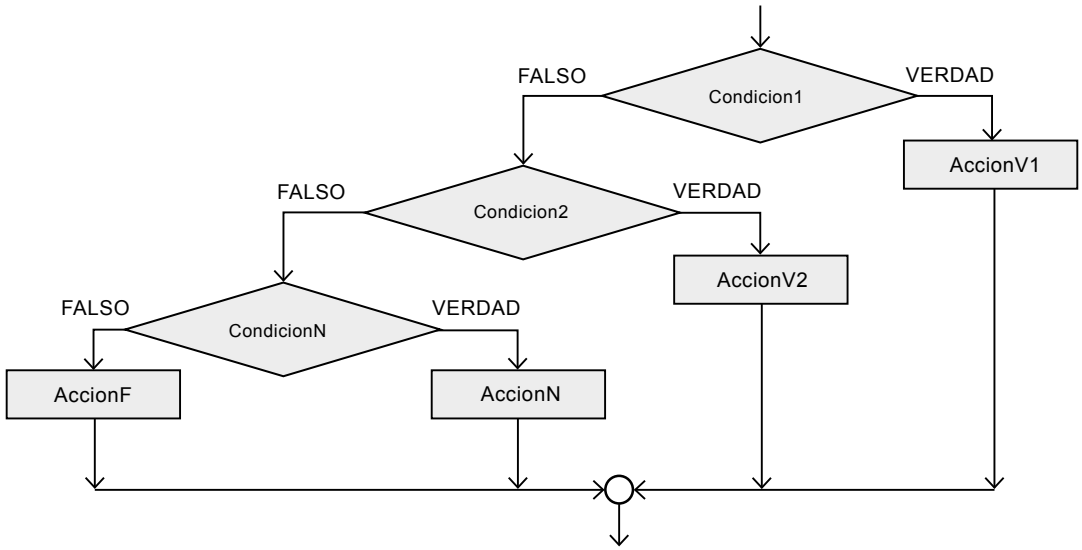
- Declarar un objeto JComboBox > JComboBox cboCategoria;
- Crear un objeto de tipo JComboBox > cboCategoria = new JComboBox();

5.6. SENTENCIA IF DOBLEMENTE ENLAZADA

La sentencia if doblemente enlazada en una aplicación Java permite evaluar un conjunto de condiciones de tal forma que genera un desencadenador, es decir, se evaluará la condición hasta que cumpla con una de las opciones, caso contrario seguirá evaluando.

Formato:

```
if (Condicion1)
   accionV1;
  else if(Condicion2)
  accionV2;
  else if(CondicionN)
  accionVN;
  else
  accionF;
```



Una sentencia if doblemente enlazada se ejecuta de la siguiente forma:

- Se evalúa la condición, cuando sea verdadera se ejecuta la acción correspondiente y se termina la instrucción.
- Si el resultado de la condición es falsa se continúa la evaluación de la condición.
- Si de todas las evaluaciones no encuentra la verdad, entonces se ejecuta la acción por defecto definida en las expresiones else. Veamos el siguiente script:

```
double sueldo;
String categoria="B";
if (categoria.equals("A"))
  double sueldo;
  sueldo=2500;
  else if (categoria.equals("B"))
    sueldo=2100;
  else if (categoria.equals("C"))
    sueldo=1900;
  else
    sueldo=1500;
```

¿Cuál es el valor asignado a la variable Sueldo, después de la evaluación en la condición doblemente enlazada?

- Sueldo > 2100.00

Veamos la evaluación de la condición:

SCRIPT	CATEGORIA	SUELDO
double sueldo;		
String categoria="B";	B	
if (categoria.equals("A"))	B=A (Falso)	
sueldo=2500;		
else if (categoria.equals("B"))	B=B (Verdadero)	
sueldo=2100;		2100.00
else if (categoria.equals("C"))		
sueldo=1900;		
else		
sueldo=1500;		

Si la categoría es B se evalúa hasta que la condición detecte la igualdad a B. Como lo ha encontrado en la segunda evaluación de la condición ya no será necesaria evaluar las demás; por tanto, el sueldo es 2100.

Veamos el caso cuando la categoría sea D, cómo se comportarían las instrucciones:

SCRIPT	CATEGORIA	SUELDO
double sueldo;		
String categoria="D";	D	
if (categoria.equals("A"))	D=A (Falso)	
sueldo=2500;		
else if (categoria.equals("B"))	D=B (Falso)	
sueldo=2100;		
else if (categoria.equals("C"))	D=C (Falso)	
sueldo=1900;		
Else		
sueldo=1500;		1500.00

Si la categoría es D se determinara por cada condición el valor Falso ya que D no es igual A ni B ni C, entonces aquí se activa el valor por defecto en este caso si se tuvo que evaluar; por tanto, el sueldo es 1500.

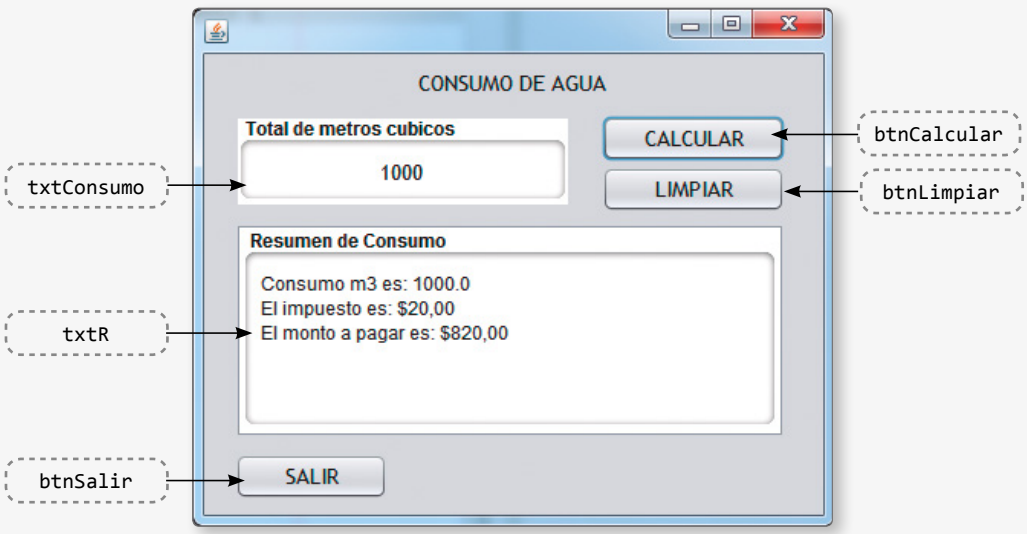
CASO DESARROLLADO 5: CONSUMO DE AGUA

Debido a la escases de agua en una determinada ciudad se pone en práctica un sistema de cobro de agua que penalizará el consumo excesivo de la forma que se indica en la siguiente tabla:

CONSUMO (M3)	DOLARES / M3
Menor a 100	0.15
Entre 100 y 499	0.20
Entre 500 y 1000	0.35
Mas de 1000	0.80

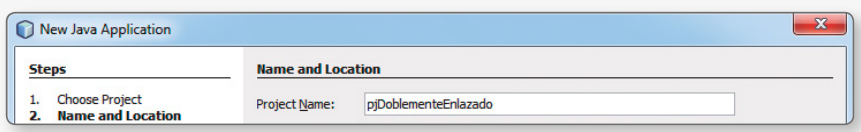
El usuario deberá ingresar un consumo en metros cúbicos para poder determinar el monto a pagar, si sabe que además se le incrementará un impuesto de 2.5% del monto solo si sobrepasa los $600.00.

GUI Propuesto:

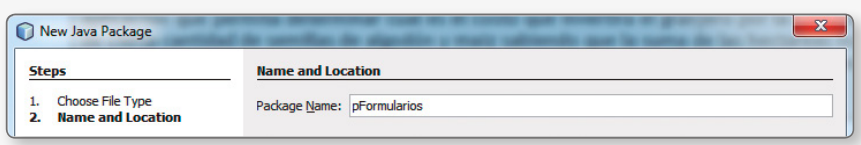


Debe considerar:

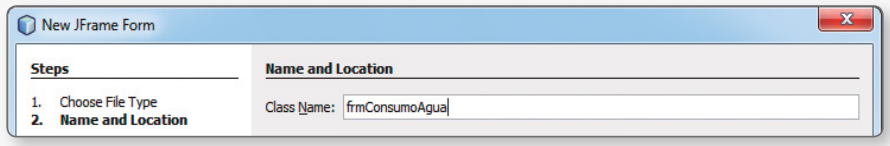
- *Crear un nuevo proyecto en NetBeans llamado pjDoblementeEnlazado.*



- *Agregar un nuevo paquete al proyecto pjDoblementeEnlazado llamado pFormularios.*



- *Agregar un nuevo Formulario al paquete pFormularios llamada frmConsumoAgua.*



- *Asigne nombres a todos los controles, para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…*
- *Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la siguiente Fig. 5.8:*

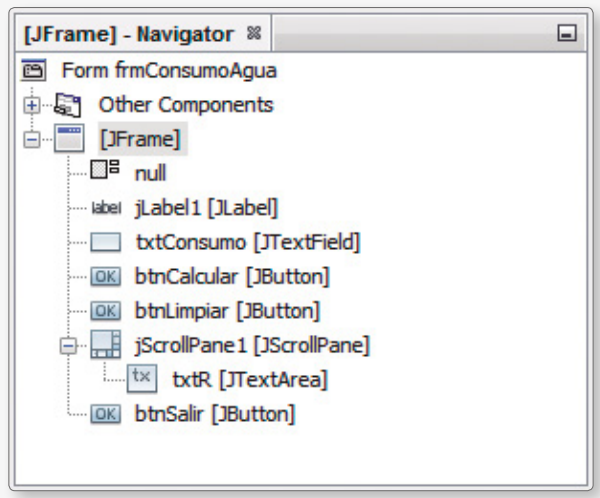


Fig. 5.8

- *Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.*
- *Al Frame active el* **Form Size Police** *con el valor* **Generate Resize Code**.
- *Asigne la siguientes propiedades a los controles:*

txtConsumo	**Border**	TitledBorder=Total de metros cubicos
	Text	Dejar vacio
txtR	**Border**	TitledBorder=Resumen de consumo
	Text	Dejar vacio
btnCalcular	**Text**	CALCULAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- *Vea el script de botones:*

El siguiente script muestra el código que se le asigna al botón btnCalcular como botón de respuesta al ingreso de valores en el Frame.

```
private void btnCalcularActionPerformed(…) {
//1
double consumo=Double.parseDouble(txtConsumo.getText());

//2
double monto=0;
if (consumo<100)
    monto=0.15*consumo;
    else if(consumo>=100 && consumo<500)
    monto=0.2*consumo;
    else if(consumo>=500 && consumo<1000)
    monto=0.35*consumo;
    else
    monto=0.8*consumo;

//3
double impuesto = 0;
if (monto>600) impuesto=2.5/100 * monto;
    monto=monto+impuesto;

//4
txtR.setText("");
txtR.append("Consumo m3 es: "+consumo);
txtR.append("\nEl impuesto es: $"+
                        String.format("%.2f",impuesto));
txtR.append("\nEl monto a pagar es: $"+
                        String.format("%.2f",monto));
}
```

En el punto uno captura el total consumido por metros cúbicos por el usuario y se almacena en la variable consumo de tipo decimal doble.

En el punto dos se evalúa dicho consumo, aquí se hace necesario el uso de las condiciones doblemente enlazadas hasta encontrar el consumo adecuado y así calcular el monto a cancelar por el usuario. Aquí se guía de los costos por metros cúbicos según el caso. Esta expresión también se pudo implementar de la siguiente forma:

```
double monto=0;
if (consumo<100)
    monto=0.15*consumo;
    else if(consumo<500)
    monto=0.2*consumo;
    else if(consumo<1000)
    monto=0.35*consumo;
    else
    monto=0.8*consumo;
```

Si el consumo fuera 400 no cumple en la primera condicional (consumo<100), pero si cumple en la segunda se le asigna el monto y se termina la instrucción, es decir, ya no continuará evaluando el valor.

En el punto tres se calcula el impuesto según el monto, cuando este supere a 600 se aplicará el impuesto, hay que tener en cuenta que al final se debe recalcular el monto aumentando el impuesto, en caso no cumpla con la condición igual se aumenta el impuesto pero con el valor inicial de cero.

En el punto cuatro se imprimen los valores resultantes de las expresiones.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
txtConsumo.setText("");
txtR.setText("");
txtConsumo.requestFocus();
}
```

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Control de Consumo",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

CASO DESARROLLADO 6: VENTA DE PRODUCTOS

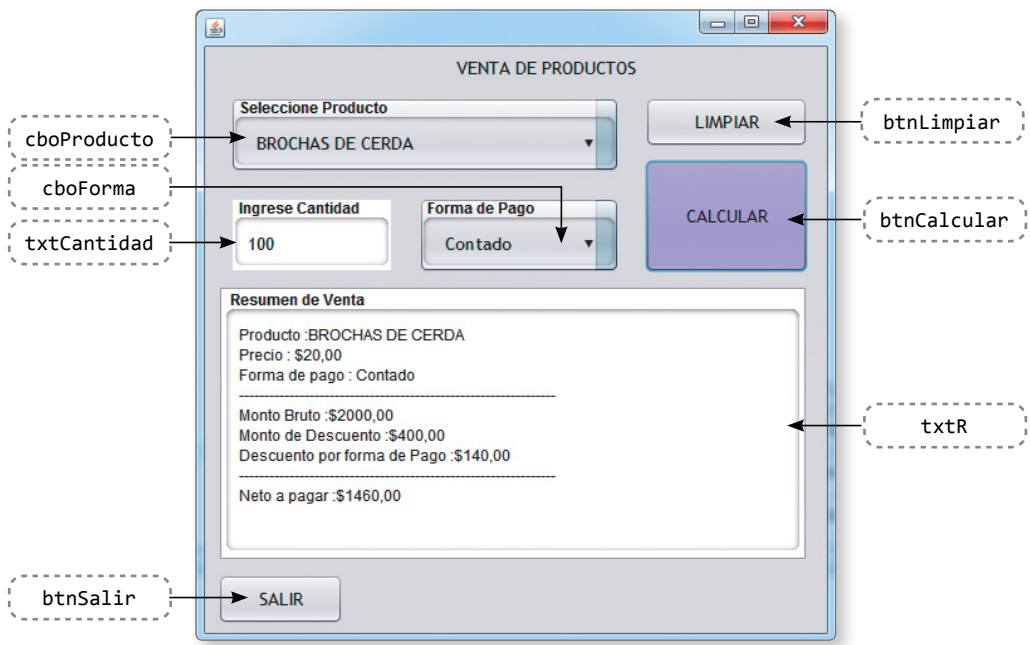
Un cliente solicita el pedido de cierta cantidad de brochas de cerda, rodillos y sellador. Las brochas de cerda tienen un 20% de descuento; los rodillos, un 15% de descuento; y el sellador, 5%.

Los precios de cada artículo son como siguen:

- *Brochas de cerda $ 20.00*
- *Rodillos $ 45.00*
- *Sellador $ 10.00*

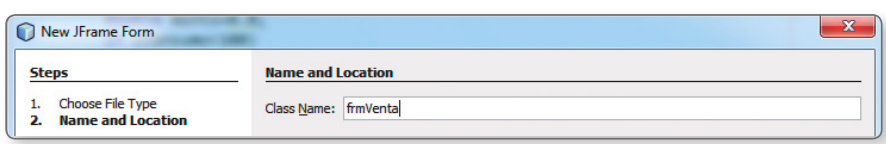
Se desea implementar una aplicación que permita ingresar la cantidad comprada por un determinado cliente. Además, si paga al contado tiene un descuento del 7%. Al final deberá mostrar el costo total de la orden, tanto para el pago de contado como para el caso de pago de crédito.

GUI Propuesto:



Debe considerar:

- Agregar al proyecto pjSeleccion un Frame llamado frmVenta.



- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la Fig. 5.9:

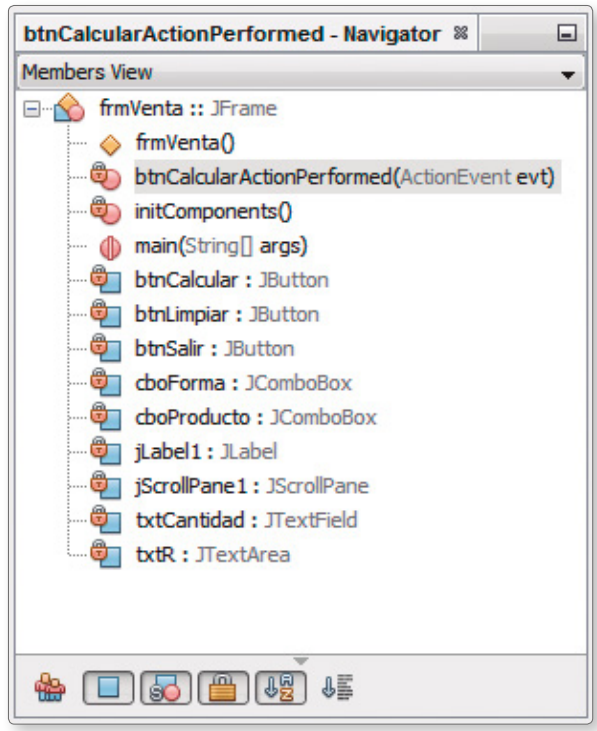


Fig. 5.9

- Asignar **Null Layout** al **setLayout** del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtCantidad	**Border**	TitledBorder=Ingrese Cantidad
	Text	Dejar vacio
cboProducto	**Border**	TitledBorder=Seleccione Producto
	Model	Dejar vacio
cboForma	**Border**	TitledBorder=Forma de Pago
	Model	Dejar vacio
txtR	**Border**	TitledBorder=Resumen de Venta
	Text	Dejar vacio
btnCalcular	**Text**	CALCULAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de botones:

El siguiente script muestra el código que permite inicializar con valores predeterminados los cuadros combinados.

```
public frmVenta() {
initComponents();
//1
cboProducto.addItem("BROCHAS DE CERDA");
cboProducto.addItem("RODILLO");
cboProducto.addItem("SELLADOR");

//2
cboForma.addItem("Contado");
cboForma.addItem("Credito");
    }
```

El siguiente script muestra el código que se le asigna al botón btnCalcular como botón de respuesta al ingreso de valores en el Frame.

```
  private void btnCalcularActionPerformed(…) {
  //1
  int producto=cboProducto.getSelectedIndex();
  int cantidad=Integer.parseInt(txtCantidad.getText());
  int forma=cboForma.getSelectedIndex();

  //2
  double precio,descuento;
  if (producto==0){
       precio=20;
       descuento=0.2;
  }
  else if(producto==1){
       precio=45;
       descuento=0.15;
  }
  else{
       precio=10;
       descuento=0.05;
  }

  //3
  double monto=(precio*cantidad);
  double montoDescuento=monto*descuento;

  //4
  double descuentoxpago;
  if (forma==0)
        descuentoxpago=0.07*monto;
        else
        descuentoxpago=0;

  //5
  double neto=monto-montoDescuento-descuentoxpago;
```

```
//6
txtR.setText("");
txtR.append("Producto :"+cboProducto.getSelectedItem());
txtR.append("\nPrecio : $"+String.format("%.2f",precio));
txtR.append("\nForma de pago : "+cboForma.getSelectedItem());
txtR.append("\n-------------------------------------------");
txtR.append("\nMonto Bruto :$"+String.format("%.2f",monto));
txtR.append("\nMonto de Descuento :$"+
                        String.format("%.2f",montoDescuento));
txtR.append("\nDescuento por forma de Pago :$"+
                        String.format("%.2f",descuentoxpago));
txtR.append("\n-------------------------------------------");
txtR.append("\nNeto a pagar :$"+
                        String.format("%.2f",neto));
    }
```

En el punto uno se declara la variable producto de tipo entero y tendrá por misión guardar el número de índice del producto seleccionado en este caso Brocha de Cerda (0), Rodillo (1) y Sellador (2). También se captura la cantidad ingresada por el usuario y finalmente se obtiene la posición de la forma de pago, es decir, al Contado (0) y al Crédito (1).

Otra forma de evaluar el producto seleccionado sería de la siguiente forma:

```
//1
String producto=String.valueOf(cboProducto.getSelectedItem());
int cantidad=Integer.parseInt(txtCantidad.getText());
int forma=cboForma.getSelectedIndex();

//2
double precio,descuento;
if (producto.equals("BROCHAS DE CERDA")){
     precio=20;
     descuento=0.2;
}
else if(producto.equals("RODILLO")){
     precio=45;
     descuento=0.15;
}
else{
     precio=10;
     descuento=0.05;
}
```

Donde el producto seleccionado por el usuario es enviado directamente a una variable de tipo String, para esto debe convertirse un objeto en String con el método String.valueOf. Luego, cuando se necesite evaluar se usará el método equals.

En el punto dos se determina el precio y el descuento que tiene asignado cada producto, recuerde que 0 es para brocha de cerda, 1 es para rodillo y 2 es para sellador.

En el punto tres se calcula el monto bruto a pagar esto se calcula multiplicando la cantidad por el precio asignado en las instrucciones anteriores, también se calcula el monto de descuento multiplicando el monto bruto por el descuento asignado según el producto seleccionado.

En el punto cuatro se calcula el descuento según la forma de pago, tenga en cuenta que 0 está asignado a la forma de pago al contado.

En el punto cinco se calcula el neto a pagar por el cliente, recuerde que aquí se deben realizar todos los descuentos respectivos.

Finalmente, en el punto seis se imprimen los valores resultantes de la aplicación.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
txtCantidad.setText("");
cboProducto.setSelectedIndex(-1);
cboForma.setSelectedIndex(-1);
txtR.setText("");
cboProducto.requestFocus();
}
```

En el siguiente script se muestra como salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

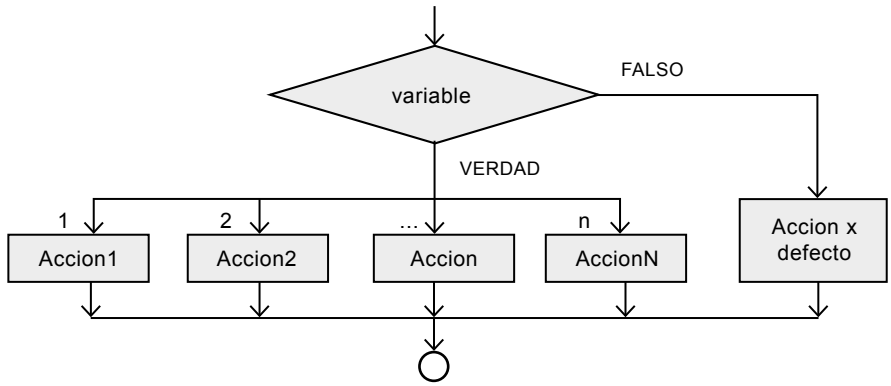
```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Venta de Productos",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

5.7. SENTENCIA SWITCH

La sentencia switch en una aplicación Java es una instrucción de decisión múltiple, es decir, el compilador busca el valor contenido en una variable contra una lista de constantes definidas por el desarrollador que pueden ser de tipo int o char, cuando el compilador encuentra el valor de igualdad entre la variable y la constante se ejecutará un grupo de instrucciones asociadas a dicha constante, si no encuentra el valor de igualdad entonces se ejecutarán instrucciones asociadas a un default.

Formato:

```
switch(variable){
   case valor1: accion1; break;
   case valor2: accion2; break;
   …
   case valorN: accionN; break;
   default: accion x defecto
}
```



Una sentencia switch se ejecuta de la siguiente forma:

- Se evalúa la variable especificada y se compara con cada una de las constantes definidas por el programador.
- Si la variable es igual a la constante definida, entonces se ejecutan las acciones definidas para dicha constante y se termina la instrucción usando la función break.
- No hay necesidad de especificar llaves por cada bloque de instrucciones.

5.8. ¿CÓMO IMPLEMENTAR UN SWITCH?

Al implementar una instrucción switch debe considerar:

- ¿Qué tipo de dato se va a evaluar?
- ¿Cuántas constantes se implementaran?
- ¿Qué acciones se ejecutarán por cada caso?

CASO 1

Una empresa renta autos de tres clases: pequeños, medianos y grandes. La tarifa del alquiler por día es: $ 15 en el auto pequeño, $ 20 en el auto mediano y $ 30 en el auto grande. Además la empresa cobra $ 0.20/km recorrido en auto pequeño, $ 0.30/km recorrido en auto mediano y $ 0.40/km recorrido en auto grande. Si la cantidad de km recorridos por el auto supera los 5km por día se le aumentará un 2.5% sobre el monto a pagar por el cliente.

Solución:

Si tiene en cuenta que las clases de autos son pequeño, mediano y grande y que podría colocarlo dentro de un cuadro combinado. La asignación de índice para cada uno sería:

CLASES DE AUTO	INDICE
Pequeño	0
Mediano	1
Grande	2

Entonces la implementación con la instrucción switch sería:

```
double tarifa=0,costo=0;
switch(clases){
    case 0: tarifa=15; costo=0.2; break;
    case 1: tarifa=20; costo=0.3; break;
    case 2: tarifa=30; costo=0.4; break;
    default: JoptionPane.showMessageDialog(null,"Clase no valida");
}
```

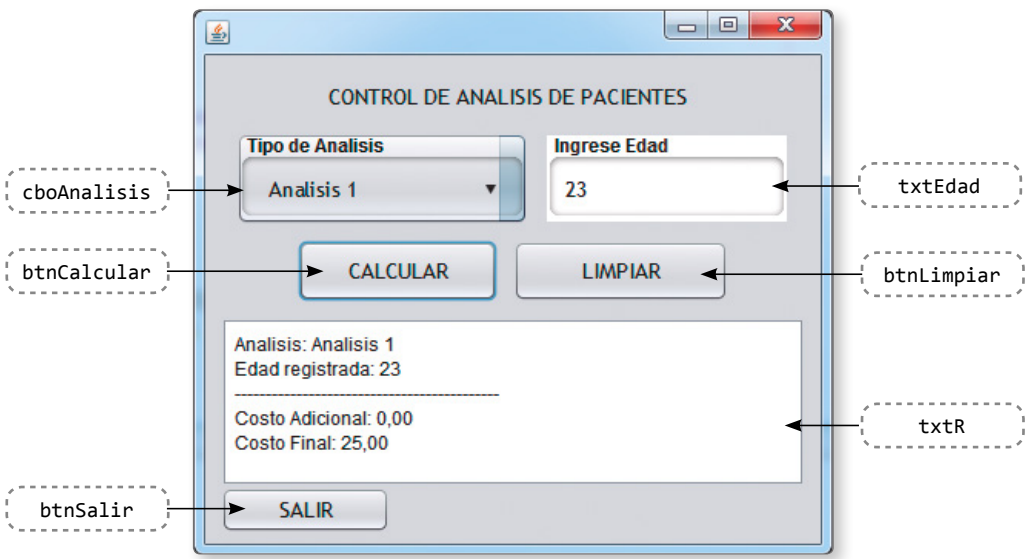
CASO DESARROLLADO 7: HOSPITAL

En un hospital de la ciudad de Lima se necesita controlar los montos recaudados por análisis de los pacientes durante un día completo de atención, los precios de los análisis son como siguen:

Tipo de análisis 1=$ 25.00
Tipo de análisis 2=$ 36.00
Tipo de análisis 3=$ 50.00

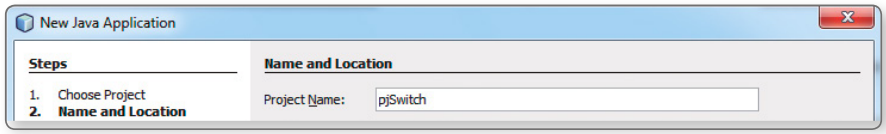
Además, se determina que los pacientes con edad de entre 14 y 22 años implican un costo adicional del 10%. Implementar una aplicación que permita calcular y mostrar el costo total que representan los análisis durante el día en el hospital.

GUI Propuesto:

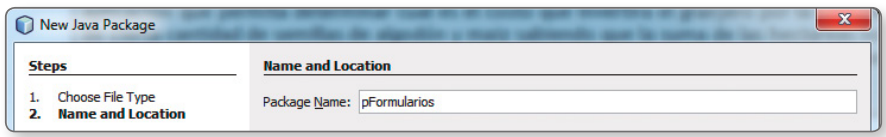


Debe considerar:

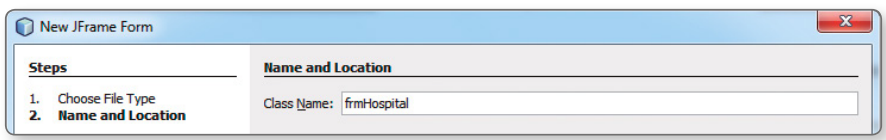
- Crear un nuevo proyecto en NetBeans llamado pjSwitch.



- Agregar un nuevo paquete al proyecto pjSwitch llamado pFormularios.



- Agregar un nuevo Formulario al paquete pFormularios llamada frmHospital.



- Asigne nombres a todos los controles, para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…
- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la siguiente Fig. 5.10.

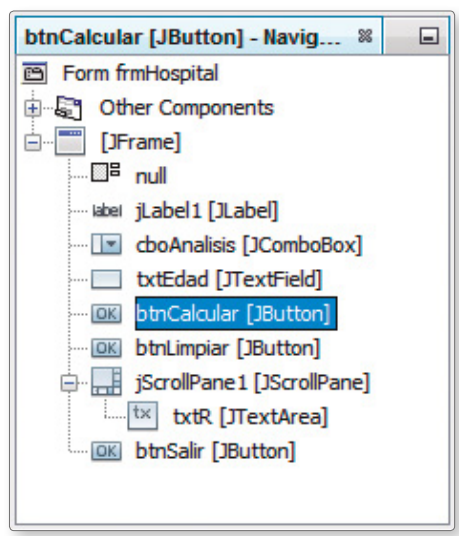


Fig. 5.10

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.

- Asigne la siguientes propiedades a los controles:

cboAnalisis	**Border**	TitledBorder=Tipo de Analisis
	Model	Dejar vacio
txtEdad	**Border**	TitledBorder=Ingrese Edad
	Text	Dejar vacio
txtR	**Text**	Dejar vacio
btnCalcular	**Text**	CALCULAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Veamos el script de botones

El siguiente script muestra el código que se le asigna al botón btnCalcular como botón de respuesta al ingreso de valores en el Frame.

```
private void btnCalcularActionPerformed(…) {
//1
int analisis=cboAnalisis.getSelectedIndex();
int edad=Integer.parseInt(txtEdad.getText());

//2
double costo;
switch(analisis){
     case 0: costo=25;break;
     case 1: costo=36;break;
     default: costo=50;
}

//3
double adicional=0;
if (edad>=14 && edad<=22)
      adicional=costo*0.1;

costo=costo+adicional;

//4
txtR.setText("");
txtR.append("Analisis: "+cboAnalisis.getSelectedItem());
txtR.append("\nEdad registrada: "+edad);
txtR.append("\n-----------------------------------------");
txtR.append("\nCosto Adicional: "+
                          String.format("%.2f",adicional));
txtR.append("\nCosto Final: "+String.format("%.2f",costo));
    }
```

En el punto uno se declara la variable análisis que tiene por misión almacenar el análisis seleccionado por el usuario, esta variable es de tipo entero ya que al seleccionar un análisis se obtentrá el índice de dicho análisis. En la variable edad se almacena la edad registrada por el usuario.

En el punto dos se calcula el costo según el tipo de análisis seleccionado, aquí use la instrucción switch ya que los índices de los análisis inician en cero y terminan en la cantidad total de elementos menos uno. Por esta razón se habilita el case 0 para el primer análisis, el case 1 para el segundo análisis y 2 para el tercer análisis.

Como verá en el código no se ha especificado el índice 2 ya que por ser el último índice se le asigna el valor por defecto.

En el punto tres se calcula el monto adicional, para esto se evalúa la edad ingresada y finalmente, se recalcula el costo con el formulario costo=costo+adicional.

En el punto cuatro se imprimen todos los valores resultantes de la aplicación dentro del control txtR.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar los valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
   cboAnalisis.setSelectedIndex(-1);
   txtEdad.setText("");
   txtR.setText("");
   cboAnalisis.requestFocus();
}
```

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Switch",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

EJERCICIOS PROPUESTOS

1. *El ejército nacional ha decidido hacer una jornada de ventas de libretas militares para muchos hombres que no han definido su situación militar u otros que no son aptos para prestar el servicio. Además de la edad de joven, se tendrá en cuenta el nivel del sisben de la persona. Para todos los hombres mayores de 18 años la libreta tendrá un costo de $350000, pero para aquellos que tengan nivel 1 se les hará un descuento del 40%; para los de nivel 2, el descuento será del 30%; para nivel 3 del 15%; y para los demás estratos o niveles no habrá descuento. Para los jóvenes con los 18 años la libreta tiene un costo de $200000 y los jóvenes con nivel del sisben 1, tendrán un descuento del 60%; para los de nivel 2, descuento del 40%; para los del 3, un descuento del 20% y para los demás estratos no habrá descuento. Hacer un algoritmo que tome la edad y el nivel del sisben de un hombre y nos muestre descuento que le hacen y su valor final a pagar.*

2. *Implemente una aplicación que permita calcular el pago que tiene que realizar un familia por el consumo mensual del agua potable, sabiendo que existe un pago fijo de $10.00 y que los 50 primeros litros de agua no tienen costo, entre 51 y 200 litros se cobra $0.5 por litro y más de 200 litros a $ 1.5 por litro.*

3. *Una empresa quiere hacer una compra de varias piezas de la misma clase a una fábrica de refacciones. La empresa, dependiendo del monto total de la compra, decidirá qué hacer para pagar al fabricante.*

   - *Si el monto total de la compra excede de $500 000 la empresa tendrá la capacidad de invertir de su propio dinero un 55% del monto de la compra, pedir prestado al banco un 30% y el resto lo pagara solicitando un crédito al fabricante.*
   - *Si el monto total de la compra no excede de $500 000 la empresa tendrá capacidad de invertir de su propio dinero un 70% y el restante 30% lo pagará solicitando crédito al fabricante.*

   El fabricante cobra por concepto de intereses un 20% sobre la cantidad que se le pague a crédito.

4. *Una empresa usa equipos de construcción alquilados y paga $450000 hasta un máximo de 320 horas de uso. Para más de 320 horas y hasta 1200 horas de uso, se pagan los $450000 más un monto adicional de $1000 por cada hora adicional. Cuando se superan las 1200 horas se pagan $200000 más un monto adicional de $ 400 por cada hora que supere las 1200 horas de uso. Dadas las horas de uso, generar los cálculos correspondientes según las reglas dadas.*

5. *Para un crucero, un viajero puede pagar con 17 días de anticipación y recibirá un descuento del 11.5% solo en el evento que su edad esté entre 28 a 30 años o solo del 7.8% si está fuera de ese rango. Si paga con 9 días de anticipación recibirá un descuento del 2% solo si antes ha tomado otro crucero. Ingresar los precios del crucero y los datos adicionales necesarios para determinar cuánto es el descuento.*

6. *Implemente una aplicación que tome el peso en kilos de una cantidad de ropa a lavar en una lavadora industrial y que le devuelva el nivel dependiendo del peso; además le informe la cantidad de litros de agua que necesita.*

Se sabe que con más de 30 kilos la lavadora no funcionará ya que es DEMASIADO PESO. Si la ropa pesa 22 o más kilos, el nivel será de MÁXIMO; si pesa 15 o más nivel será de ALTO; si pesa 8 o más será un nivel MEDIO o de lo contrario el nivel será MINIMO.

7. *Aplicación que permita emitir una factura correspondiente a la venta de un artículo del cual se adquieren varias unidades. El IVA a aplicar es del 12% y si el monto total (precio de ventas + IVA) es menor o igual a $30.00 se aplicará un 10% de descuento, si el monto total es mayor a $30.00 pero menor a $70.00 se le aplicará un 25% y si el monto total es superior a $70.00 entonces se le aplicará un descuento del 35%.*

8. *Implementar una aplicación que permita agregar una letra para representar la calificación de los estudiantes, las calificaciones son notas entre 1 y 10; use los siguientes parámetros: A para calificaciones mayores o iguales a 9, B para calificaciones mayores o iguales a 8, C para calificaciones mayores o iguales a 7, D para calificaciones mayores o iguales a 6, F para todas las demás calificaciones.*

CAP.

6

Programación modular

CAPACIDAD:

- Reconocer el uso de las variables locales y globales.
- Implementar las aplicaciones usando métodos sin valor de retorno.
- Implementar aplicaciones usando métodos con valor de retorno.

CONTENIDO:

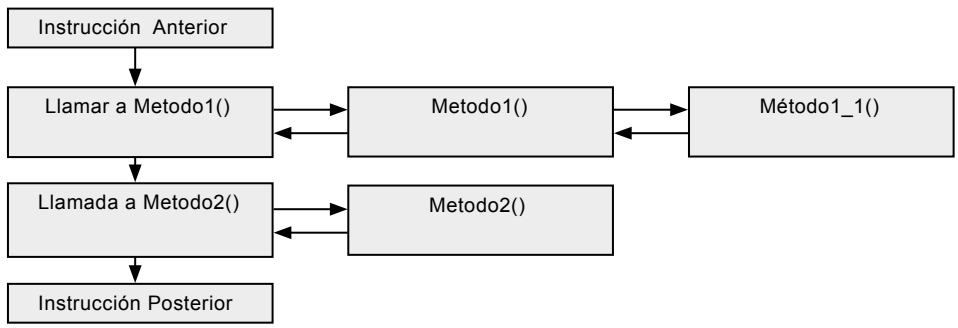
6.1. INTRODUCCIÓN

La programación modular es una metodología que adoptan los desarrolladores por la necesidad de implementar casos complejos y generar muchas líneas de código. Consiste en dividir un programa en módulos o subprogramas controlados desde un modulo principal, con el fin de hacerlo más legible y manejable.

En realidad es una evolución de la programación estructurada para solucionar problemas de programación más grandes y complejos, ya que sería demasiado complicado tener controlada una aplicación de 1000 líneas a más.

Al aplicar la programación modular, una aplicación que resulta compleja debe ser dividida en varias subaplicaciones más simples, y estas a su vez en otras subaplicaciones más simples. Esto debe hacerse hasta obtener códigos suficientemente simples como para poder ser resueltos fácilmente con Java, esto no quiere decir que no sea manejable tener mucho código sino que el desarrollador tendrá más comodidad al programar. A esta técnica se le llama refinamiento sucesivo, es decir, divide y vencerás.

En Java a los módulos o subaplicaciones se les denomina métodos, estos serán implementados netamente con el criterio del programador, pues no hay un patrón o regla que se deba seguir.



Finalmente, se podría decir que un método tiene las siguientes características:

- Es un bloque de códigos que tiene un nombre especificado por el programador.
- En algunas ocasiones necesita de parámetros o argumentos, esto dependerá de la tarea que se necesite realizar.
- Un método contiene sentencias o instrucciones para realizar un proceso.
- Un método devuelve un valor de algún tipo conocido solo si así lo implementa el programador.

6.2. VARIABLES LOCALES Y GLOBALES

Se determina variable local y global al lugar donde esta sea declarada; una variable afectará el uso que el programa quiera hacer de esa variable. Si se considera lo mencionado en el tema de las variables en los capítulos anteriores, verá que en todas las aplicaciones realizadas hasta el momento son variables de ámbito local. Vea el siguiente script:

```
private void btnCalcularActionPerformed(…) {
double consumo=Double.parseDouble(txtConsumo.getText());

//2
double monto=0;
if (consumo<100)
    monto=0.15*consumo;
    else if(consumo>=100 && consumo<500)
    monto=0.2*consumo;
    else if(consumo>=500 && consumo<1000)
    monto=0.35*consumo;
    else
    monto=0.8*consumo;

//3
double impuesto = 0;
if (monto>600) impuesto=2.5/100 * monto;
    monto=monto+impuesto;

//4
txtR.setText("");
txtR.append("Consumo m3 es: "+consumo);
txtR.append("\nEl impuesto es: $"+
                          String.format("%.2f",impuesto));
txtR.append("\nEl monto a pagar es: $"+
                          String.format("%.2f",monto));
}
```

Todas las variables definidas en el método **btnCalcularActionPerformed** son declaradas como variables locales como por ejemplo: consumo, monto e impuesto.

6.3. VARIABLES LOCALES

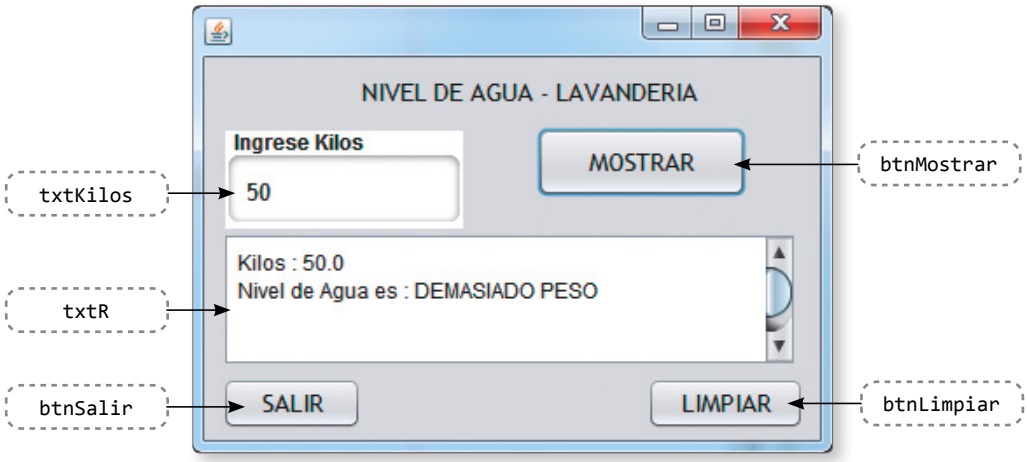
Se dice que una variable es local cuando su ámbito de trabajo se restringe al método que la ha declarado entonces se dice que dicha variable es local a ese metodo. Esto quiere decir que dicha variable sólo va a poder ser manipulada única y exclusivamente dentro del método, evitando ser referencia fuera. La caracteristica principal de las variables locales es que al finalizar la ejecución del método estas variables se destruyen; además cuando son declaradas dentro de un método no son inicializadas con ningun valor ya que se crea vacia.

CASO DESARROLLADO 1: LAVANDERÍA CON VARIABLES LOCALES

Implemente una aplicación que tome el peso en kilos de una cantidad de ropa a lavar en una lavadora industrial y nos devuelva el nivel de agua dependiendo del peso.

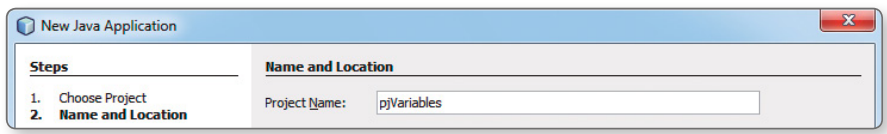
Se sabe que con más de 30 kilos la lavadora no funcionará ya que es DEMASIADO PESO. Si la ropa pesa 22 o más kilos, el nivel será de MÁXIMO; si pesa 15 o más será de ALTO; si pesa 8 o más, será un nivel MEDIO; o de lo contrario el nivel será MÍNIMO.

GUI Propuesto:

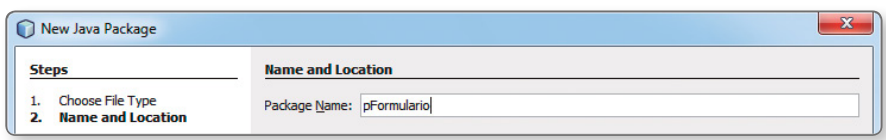


Debe considerar:

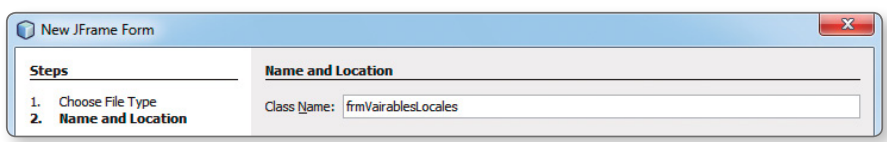
- Crear un nuevo proyecto en NetBeans llamado pjVariables.



- Agregar un nuevo paquete al proyecto pjVariables llamado pFormulario.



- Agregar un nuevo Formulario al paquete pFormulario llamado frmVariablesLocales.



- Asigne nombres a todos los controles, para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…
- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la Fig. 6.1.

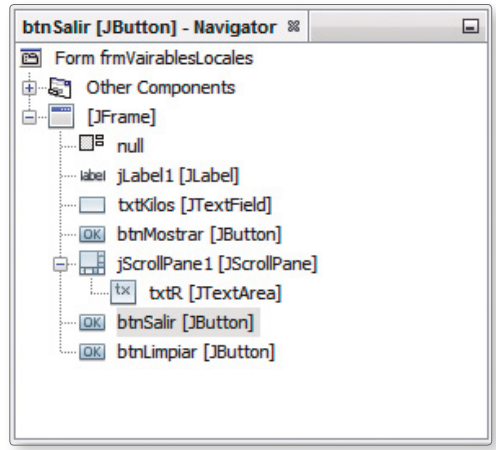


Fig. 6.1

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtKilos	**Border**	TitledBorder=Ingrese Kilos
	Text	Dejar vacio
txtR	**Text**	Dejar vacio
btnMostrar	**Text**	MOSTRAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de botones:
- El siguiente script muestra el código que se le asigna al botón btnMostrar como botón de respuesta al ingreso de valores en el Frame.

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
//1
double kilos=Double.parseDouble(txtKilos.getText());

//2
String mensaje="";
if (kilos>30)
    mensaje="DEMASIADO PESO";
    else if(kilos>22)
    mensaje="MAXIMO";
    else if(kilos>15)
    mensaje="MEDIO";
    else
    mensaje="MINIMO";

//3
txtR.setText("");
txtR.append("Kilos : "+kilos);
txtR.append("\nNivel de Agua es : "+mensaje);
}
```

En el punto uno se declara la variable kilos que almacena el valor ingresado en el control txtKilos.

En el punto dos se evalúan los kilos, dependiendo del valor se asignará un mensaje.

En el punto tres, se imprimen los valores resultantes de las expresiones, considerando la declaración de las variables Kilos y mensaje como locales.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
txtKilos.setText("");
txtR.setText("");
txtKilos.requestFocus();
}
```

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Variables Locales",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

6.4. VARIABLES GLOBALES

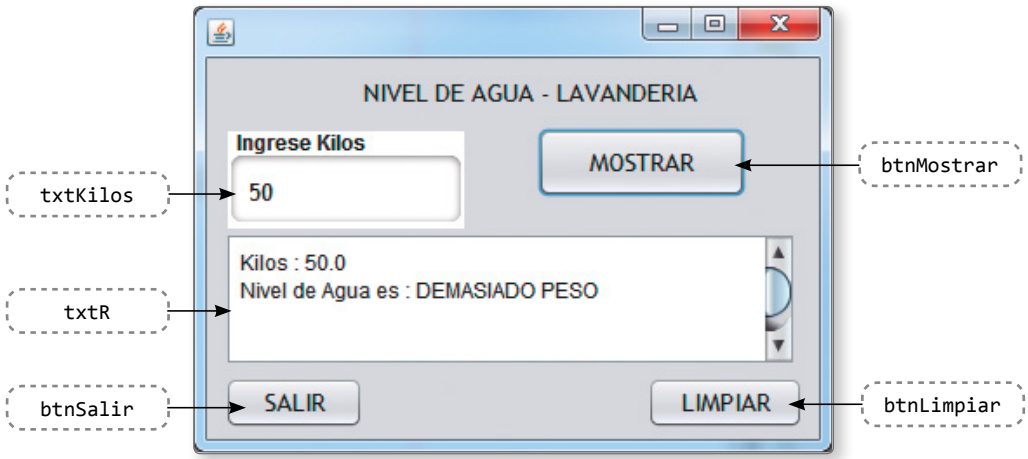
Una variable global se define fuera del cuerpo de un método, normalmente se declara al inicio de la clase principal. El ámbito de una variable global en Java es sobre todos los métodos que componen la aplicación, cualquier método puede acceder a dichas variables para obtener o asignar un valor. Es decir, se puede hacer referencia a su dirección de memoria en cualquier parte del programa. Su característica principal es que toda variable global se inicializa automáticamente.

CASO DESARROLLADO 2: LAVANDERÍA CON VARIABLES GLOBALES

Implemente una aplicación que tome el peso en kilos de una cantidad de ropa a lavar en una lavadora industrial y nos devuelva el nivel de agua dependiendo del peso.

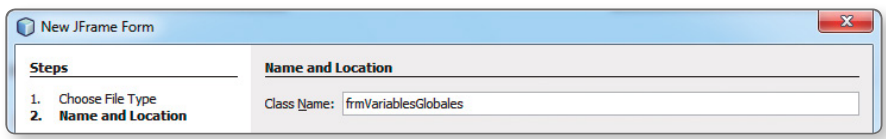
Se sabe que con más de 30 kilos la lavadora no funcionará ya que es DEMASIADO PESO. Si la ropa pesa 22 o más kilos, el nivel será de MÁXIMO; si pesa 15 o más, el nivel será de ALTO; si pesa 8 o más será un nivel MEDIO; o de lo contrario el nivel será MÍNIMO.

GUI Propuesto:



Debe considerar:

- Agregar un nuevo Formulario al paquete pFormulario llamado frmVariablesGlobales.



- Asigne nombres a todos los controles, para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…
- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la Fig. 6.2.

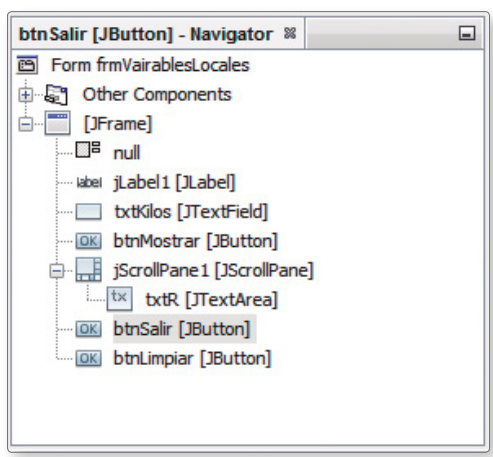


Fig. 6.2

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne las siguientes propiedades a los controles:

txtKilos	**Border**	TitledBorder=Ingrese Kilos
	Text	Dejar vacio
txtR	**Text**	Dejar vacio
btnMostrar	**Text**	MOSTRAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de botones:

El siguiente script muestra la declaración de las variables globales de la aplicación, como notará ya no es necesario inicializar las variables pues Java lo hace automáticamente.

```
public class frmVariablesGlobales extends javax.swing.JFrame {

    double kilos;
    String mensaje;

    public frmVariablesGlobales() {
        initComponents();
    }
```

El siguiente script muestra el código que se le asigna al botón btnMostrar como botón de respuesta al ingreso de valores en el Frame.

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
//1
kilos=Double.parseDouble(txtKilos.getText());

//2
if (kilos>30)
    mensaje="DEMASIADO PESO";
    else if(kilos>22)
    mensaje="MAXIMO";
    else if(kilos>15)
    mensaje="MEDIO";
    else
    mensaje="MINIMO";

//3
txtR.setText("");
txtR.append("Kilos : "+kilos);
txtR.append("\nNivel de Agua es : "+mensaje);
}
```

Como kilos fue declarado como variable global en el punto uno se le asigna un valor y el punto tres lo imprime.

En el punto dos se evalúa el valor de kilos que se le asignó en el punto uno.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
txtKilos.setText("");
txtR.setText("");
txtKilos.requestFocus();
}
```

En el siguiente script se muestra como salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Variables Globales",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

6.5. MÉTODOS VOID

El término *void* significa vacío, este método se caracteriza por no devolver valor alguno a quien lo invoque, pero sigue la tendencia de la programación modular, es decir, realiza un proceso determinado. Ahora se preguntará en qué casos se puede usar los métodos void, veamos algunos casos:

- Asignar un valor a una variable global.
- Calcular algún valor e imprimirlo en el mismo método.
- Limpiar los controles como las cajas de texto.
- Emitir mensajes al usuario.

Formato: implementación de un método void sin parámetros.

```
void nombreMetodo(){
   //Declaracion de variables locales
   //Cuerpo del metodo
}
```

Formato: implementación de un método void con parámetros.

```
void nombreMetodo(tipo parametro1, tipo parametro2…){
   //Declaracion de variables locales
   //Cuerpo del metodo
}
```

Formato: llamada a un método void sin parámetros.

```
nombreMetodo();
```

Formato: llamada a un método void con parámetros.

```
nombreMetodo(parametro1, parametro2);
```

6.5.1. Métodos void sin parámetros

Veamos un ejemplo de cómo se implementa un método void sin parámetros con su respectiva llamada, para esto usará valores directos sobre las variables:

Primero debe declarar como variables globales a n1,n2,n3 y promedio:

```
int n1;
int n2;
int n3;
double promedio;
```

Luego implementamos el método que permite calcular el promedio de las 3 notas:

```
void calculaPromedio(){
    promedio=(n1+n2+n3)/3.0;
}
```

La llamada al método sería de la siguiente forma:

```
n1=10;
n2=15;
n3=20;

calculaPromedio();

JOptionPane.showMessageDialog("El Promedio es: "+promedio);
```

Cuando se le asigna a n1 el valor 10 este será accesible por cualquier método, es decir, al momento de llamar a calculaPromedio y usar n1 en el cálculo del promedio este ya tiene valor, lo mismo le sucede a n2 y n3, en caso se declarará como variable local a n1, n2 y n3 la aplicación generaría un error.

6.5.2. Métodos void con parámetros

Veamos el caso anterior con la diferencia que se usarán parámetros:

Primero debe declarar como variable global al promedio:

```
double promedio;
```

Luego implemente el método que permite calcular el promedio de las 3 notas, con los parámetros necesarios para el promedio:

```
void calculaPromedio(int n1, int n2, int n3){
     promedio=(n1+n2+n3)/3.0;
}
```

La llamada al método sería de la siguiente forma:

```
int n1=10;
int n2=15;
int n3=20;

calculaPromedio(n1,n2,n3);

JOptionPane.showMessageDialog("El Promedio es: "+promedio);
```

Al no declarar las notas como variables globales las notas deberán ser pasadas al método por medio de parámetros, observe que las variables n1, n2 y n3 fueron declaradas como locales y que al llamar al método calculaPromedio se le envía las notas sabiendo que tiene un valor asignado previamente, los valores enviados serán recibidos por los parámetros en el mismo orden que se le envían. Hay que resaltar que no necesariamente las variables deben llamarse igual que los parámetros, ya que como se mencionó anteriormente, lo que importa es el orden en que se envían los valores al método. Entonces, el último script que invoca al método calculaPromedio podría ser de la siguiente forma:

```
int a=10;
int b=15;
int c=20;

calculaPromedio(a,b,c);

JOptionPane.showMessageDialog("El Promedio es: "+promedio);
```

Como se observa, a, b y c son variables locales que tienen asignadas un valor numérico de tipo entero así como lo hacia n1, n2 y n3. Estas variables son enviadas al método calculaPromedio de tal forma que el valor de a es asignado a n1 dentro del método, b es asignado a n2 y c es asignado a n3 respetivamente. Como se mencionó anteriormente los parámetros del método reciben los valores que se les envía desde la llamada; lo que sí es importante es la cantidad de parámetros implementados ya que se deben enviar la misma cantidad de variables hacia los parámetros.

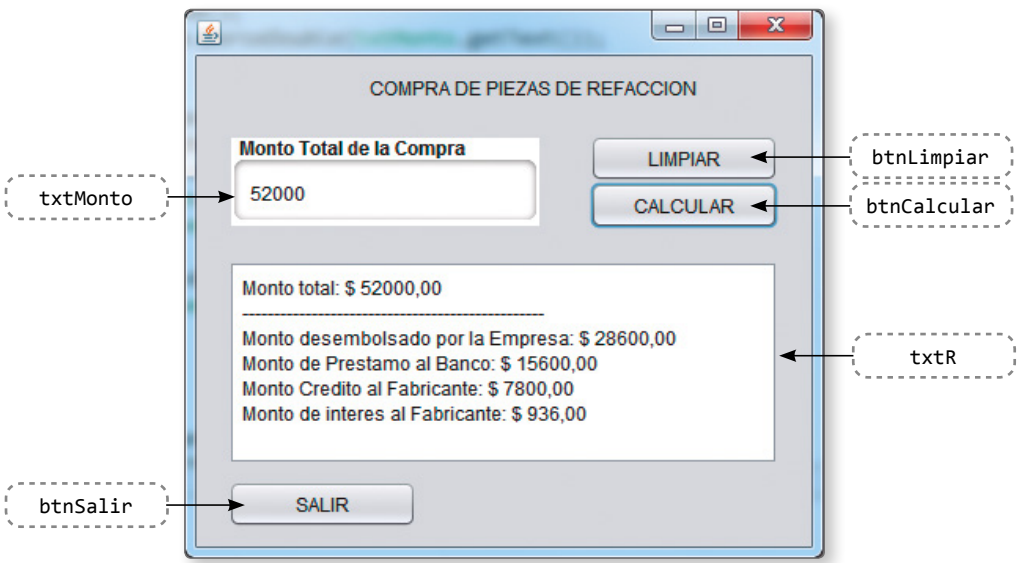
CASO DESARROLLADO 3: COMPRA DE PIEZAS DE REFACCIÓN SIN PARÁMETROS

Una empresa quiere hacer una compra de varias piezas de la misma clase a una fábrica de refacciones. La empresa, dependiendo del monto total de la compra,, decidirá qué hacer para pagar al fabricante:

- *Si el monto total de la compra excede de $ 50,000.00 la empresa tendrá la capacidad de invertir de su propio dinero un 55% del monto de la compra, pedir prestado al banco un 30% y el resto lo pagará solicitando un crédito al fabricante.*
- *Si el monto total de la compra no excede de $ 50,000.00 la empresa tendrá la capacidad de invertir de su propio dinero un 70% y el restante 30% lo pagará solicitando crédito al fabricante.*

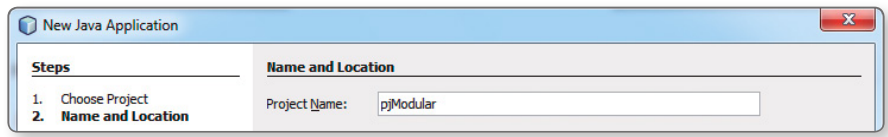
El fabricante cobra por concepto de intereses un 20% sobre la cantidad que se le pague a crédito. Implemente una aplicación que permita ingresar el monto total de la compra y determine el monto a invertir por parte de la empresa, el monto prestado por el banco y el monto solicitado al fabricante.

GUI Propuesto:

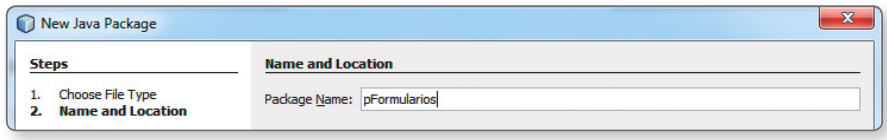


Debe considerar:

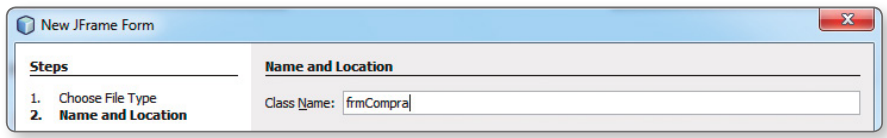
- Crear un nuevo proyecto en NetBeans llamado pjModular.



- Agregar un nuevo paquete al proyecto pjModular llamado pFormularios.



- Agregar un nuevo Formulario al paquete pFormularios llamada frmCompra.



- Asigne nombres a todos los controles para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…
- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la Fig. 6.3.

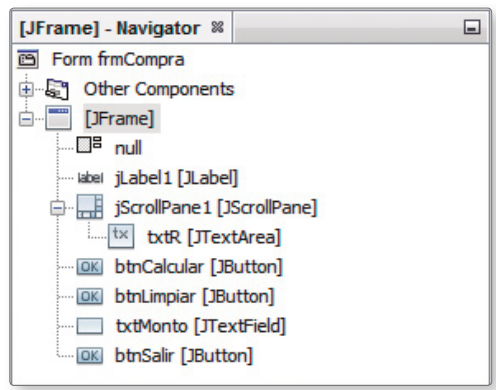


Fig. 6.3

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtMonto	**Border** **Text**	TitledBorder=Monto total de la compra Dejar vacio
txtR	**Text**	Dejar vacio
btnCalcular	**Text**	CALCULAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de botones:

El siguiente script muestra la declaración de las variables globales de la aplicación:

```
package pFormularios;
public class frmCompra extends javax.swing.JFrame {
//1
double monto;
double empresa;
double prestamo;
double credito;
double interes;

public frmCompra() {
   initComponents();
}
```

En el punto uno se declaran las variables globales monto para obtener el monto total ingresado a la aplicación. La variable empresa que almacenará el monto asumido por la empresa, la variable préstamo que almacenará el monto que se solicitará al banco solo en caso el monto sea superior a 50000 dólares, la variable crédito que permitirá almacenar el monto que se solicitará al fabricante e interés que almacenará el monto de interés que el fabricante solicita por pedir un crédito.

El siguiente script muestra el código que se le asigna al botón btnCalcular como botón de respuesta al ingreso de valores en el Frame.

```
private void btnCalcularActionPerformed(…) {
capturaMonto();
calculaInversion();
calculaInteres();
imprimir();
}
```

Como verá dentro del botón calcular solo hay llamadas a métodos y estos se encargarán de todo el proceso necesario, lo que si hay que tener en cuenta es el orden de las llamadas a los métodos, puesto que no se puede imprimir sin antes llamar al método calculaInversion o calculaInteres.

El siguiente script muestra la implementación del método capturaMonto.

```
void capturaMonto(){
    monto=Double.parseDouble(txtMonto.getText());
}
```

El método capturaMonto tiene por misión capturar el monto ingresado en el control txtMonto y asignado a la variable monto de ámbito global, con este dicho valor podrá ser accedido por los demás métodos de la misma aplicación.

El siguiente script muestra la implementación del método calculaInversion.

```
void calculaInversion(){
   if (monto>50000){
       empresa=0.55*monto;
       prestamo=0.3*monto;
       credito=monto-(empresa+prestamo);
   }else{
       empresa=0.7*monto;
       credito=monto-empresa;
   }
}
```

El método calculaInversion permite calcular los montos que asume la empresa, el monto de préstamo y el monto de crédito condicionando, para esto el monto capturado por el método capturaMonto, a la vez que estos valores calculados también son accesibles por otros métodos por haber sido declarados como globales.

El siguiente script muestra la implementación del método calculaInteres.

```
void calculaInteres(){
        interes=0.12*credito;
}
```

El método calculaInteres tiene por misión calcular el monto que le asigna el fabricante por concepto de pago al crédito del monto de la compra, este valor también podrá ser accedido por otros métodos por haber sido declarado como variable global.

El siguiente script muestra la implementación del método imprimir:

```
void imprimir(){
   txtR.setText("");
   txtR.append("Monto total: $ "+String.format("%.2f",monto));
   txtR.append("\n-----------------------------------------------");
   txtR.append("\nMonto desembolsado por la Empresa: $ "+
                              String.format("%.2f",empresa));
   txtR.append("\nMonto de Prestamo al Banco: $ "+
                              String.format("%.2f",prestamo));
   txtR.append("\nMonto Credito al Fabricante: $ "+
                              String.format("%.2f",credito));
   txtR.append("\nMonto de interes al Fabricante: $ "+
                              String.format("%.2f",interes));
}
```

El método imprimir tiene por misión imprimir todos los valores resultantes de la aplicación, como todos fueron declarados como globales es permitida su impresión.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
        limpiaControles();
}

void limpiaControles(){
        txtMonto.setText("");
        txtR.setText("");
        txtMonto.requestFocus();
}
```

Como verá dentro del botón btnLimpiar se invoca al método limpiaControles que permitirá limpiar todos los controles usados en la aplicación.

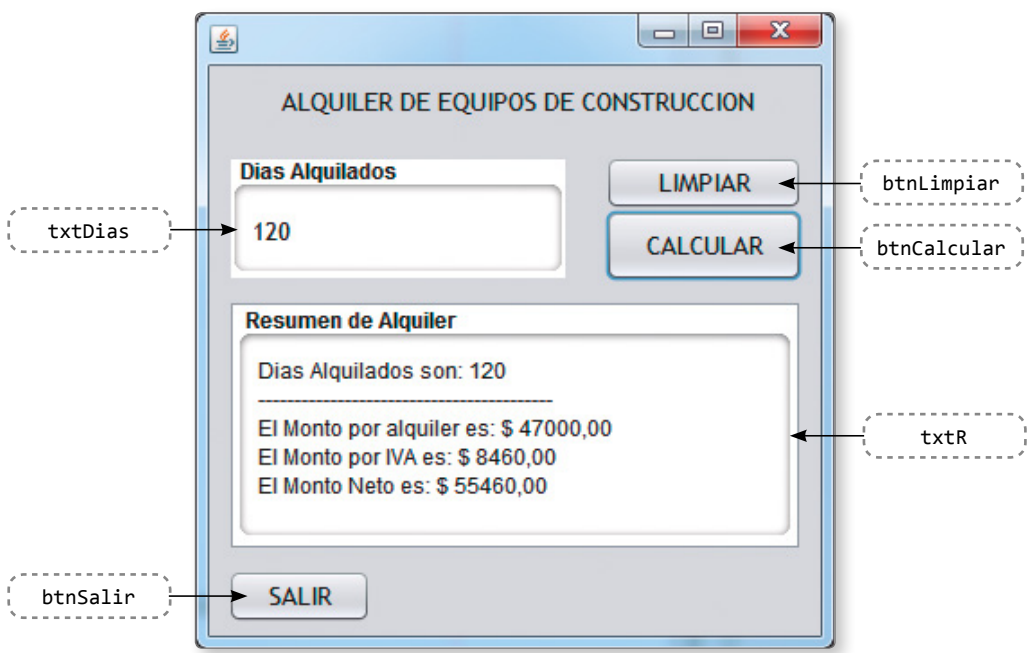
En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Métodos Void",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

CASO DESARROLLADO 4: ALQUILER DE EQUIPOS DE CONSTRUCCIÓN CON PARÁMETROS

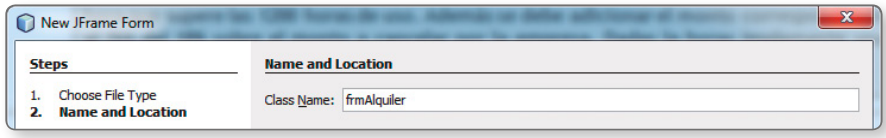
Una empresa usa equipos de construcción alquilados y paga $14,500.00 hasta un máximo de 100 días de uso. Para más de 100 días y hasta 150 días de uso, se pagan los $17,000.00 más un monto adicional de $1000 por cada día que supere los 150 días. Cuando se superan los 200 días se pagan $20,000.00 más un monto adicional de $400.00 por cada día que supere los 200 días de uso. Además, se debe adicionar el monto correspondiente al IVA del 18% sobre el monto a cancelar por la empresa. Dadas las horas implemente una aplicación que permita mostrar los gastos que tiene que realizar la empresa.

GUI Propuesto:



Debe considerar:

- Agregar un nuevo Formulario al paquete pFormularios llamada frmAlquiler.



- Asigne nombres a todos los controles para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…
- Despues de asignar los nombres de los objetos el panel Navigator debe mostrarse como la Fig. 6.4.

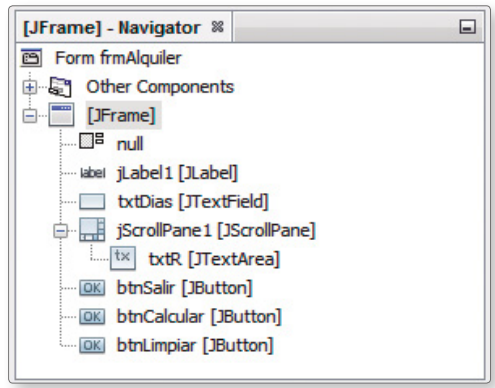


Fig. 6.4

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtDias	**Border**	TitledBorder=Dias Alquilados
	Text	Dejar vacio
txtR	**Text**	Dejar vacio
btnCalcular	**Text**	CALCULAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de botones:

El siguiente script muestra el código que se le asigna al botón btnCalcular como botón de respuesta al ingreso de valores en el Frame.

```
private void btnCalcularActionPerformed(…) {
    capturarDatos();
}
```

El siguiente script muestra el código que se le asigna al botón btnCalcular como botón de respuesta al ingreso de valores en el Frame.

```
//1
void capturarDatos(){
    int dias=Integer.parseInt(txtDias.getText());
    determinaMonto(dias);
}

//2
void determinaMonto(int dias){
    double m;
    if (dias<=100)
        m=14500;
        else if(dias<=150)
            m=17000+(150-dias)*1000;
        else
            m=20000+(200-dias)*400;
        determinaIVA(dias,m);
}

//3
void determinaIVA(int dias, double monto){
    double I=monto*0.18;
    determinaNeto(dias, monto,I);
}

//4
void determinaNeto(int d,double m, double I){
    double n=m+I;
    imprimir(d,m,I,n);
}

//5
void imprimir(int dias,double monto,double IVA,double neto){
    txtR.setText("");
    txtR.append("Dias Alquilados son: "+dias);
    txtR.append("\n----------------------------------------");
    txtR.append("\nEl Monto por alquiler es: $ "+
                                String.format("%.2f",monto));
    txtR.append("\nEl Monto por IVA es: $ "+
                                String.format("%.2f",IVA));
    txtR.append("\nEl Monto Neto es: $ "+String.format("%.2f",neto));
}
```

En el punto uno se implementa el método capturarDatos que permite asignar el valor ingresado en los días alquilados por el usuario en la variable local días. Luego, allí mismo se invoca al método determinaMonto enviando como parámetro los días capturados.

En el punto dos se implementa el método determinaMonto, el cual recibe como parámetro los días alquilados, luego se declara la variable local m que tendrá por misión almacenar el valor del monto. Después se evalúan los dias alquilados y finalmente, se invoca al método determinaIVA con los parámetros días y el monto por medio de la variable local m.

En el punto tres se implementa el método determinaIVA que tiene como parámetro los días y el monto, como verá aquí el parámetro días acumula el valor de la variable días enviada desde el método determinaMonto, así también el parámetro monto recibe como valor el contenido de la variable m enviada desde el método determinaMonto. Luego se determina el IVA por medio de la variable local

I, que luego será enviada como parámetro al método determinaNeto, como notará aquí el parámetro días no fue usado en el método, pero se envía como parámetro al método determinaNeto para que este a su vez lo envíe al método imprimir, recuerde que este caso se trata de métodos void con parámetros y; por lo tanto, se está evitando usar variables globales.

En el punto cuatro se implementa el método determinaNeto que recibe como parámetros d que representa a los días alquilados, m que representa el monto calculado y la i que representa el IVA. Luego se calcula el neto y se almacena dentro de la variable local n y finalmente, se invoca al método imprimir con los parámetros necesarios en la impresión como los días(d), monto(m), IVA(I) y el neto(n).

En el punto cinco se implementa el método imprimir que recibe como parámetros los días, monto, IVA y el neto que serán impresos en el control txtR.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
    limpiaControles();
}

void limpiaControles(){
    txtDias.setText("");
    txtR.setText("");
    txtDias.requestFocus();
}
```

En el siguiente script se muestra como salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Void con parametros",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

6.6. Métodos con valor de retorno

Un método en Java se caracteriza por indicar el tipo de dato que devolverá, ya se ha visto que si se implementa con void no devuelve valor alguno, pero si se indica algún tipo de datos se vuelve un método con valor de retorno.

Estos métodos pueden devolver una variable u objeto, bien sea por valor devolviendo una copia o por referencia para los objetos. Todos los tipos de datos primitivos en Java se devuelven por valor y todos los objetos se devuelven por referencia.

Para devolver un valor se utiliza la palabra clave return. La palabra clave return va seguida de una expresión que será evaluada para saber el valor de retorno. Esta expresión puede ser compleja o puede ser simplemente el nombre de un objeto, una variable de tipo primitivo o una constante.

Formato: implementación de un método con valor de retorno sin parámetros.

```
tipoDatos nombreMetodo(){
   //Declaracion de variables locales
   //Cuerpo del método
   //Valor de retorno
}
```

Formato: implementación de un método con valor de retorno con parámetros.

```
tipoDatos nombreMetodo(tipo parametro1, tipo parametro2…){
   //Declaracion de variables locales
   //Cuerpo del método
   //Valor de Retorno
}
```

Formato: llamada a un método con valor de retorno sin parámetros.

```
variable = nombreMetodo();
```

Formato: llamada a un método con valor de retorno con parámetros.

```
Variable = nombreMetodo(parametro1, parametro2);
```

6.6.1. Métodos con valor de retorno sin parámetros

Veamos un ejemplo de cómo se implementa un método con valor de retorno sin parámetros con su respectiva llamada, para esto se usará valores directos sobre las variables:

En el siguiente caso verá cómo se captura un valor desde el control txtEdad a la variable local edad.

Primero implemente el método capturaEdad que permitirá obtener la edad ingresada por el usuario en el control txtEdad:

```
int capturaEdad(){
   Return Integer.parseInt(txtEdad.getText());
}
```

Luego invoque al método capturaEdad y le asigna a una variable llamada edad del mismo tipo:

```
int edad = capturaEdad();
```

También podría imprimir directamente la edad de la siguiente forma:

```
JOptionPane.showMessageDialog(null, "La Edad es: "+capturaEdad());
```

O podría usar la edad para el cálculo de la fecha de nacimiento:

```
int año=2013-capturaEdad();
```

6.6.2. Métodos con valor de retorno con parámetros

Veamos el caso del promedio de 3 notas de un alumno.

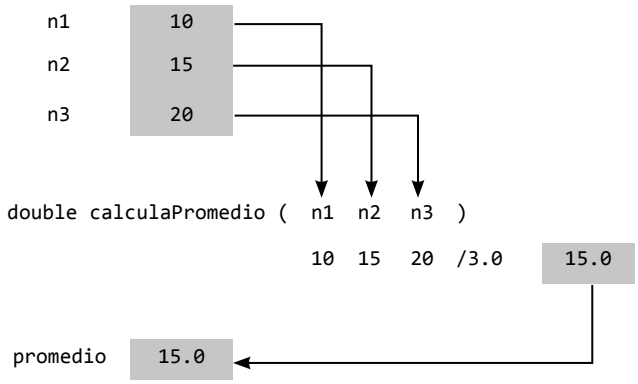
Primero implementará el método que permite calcular el promedio de las 3 notas, con los parámetros necesarios para el promedio:

```
double calculaPromedio(int n1, int n2, int n3){
     return (n1+n2+n3)/3.0;
}
```

La llamada al método sería de la siguiente forma:

```
int n1=10;
int n2=15;
int n3=20;

double promedio = calculaPromedio(n1,n2,n3);

JOptionPane.showMessageDialog("El Promedio es: "+promedio);
```

Vea de forma gráfica que valores se le asigna a todas las variables de la aplicación:



CASO DESARROLLADO 5: TELAS Y MODA DE OTOÑO SIN PARÁMETROS

Una tienda de venta de ropa para damas necesita una aplicación que permita controlar las ventas realizadas por temporada, para lo cual cuenta con los siguientes precios para sus productos:

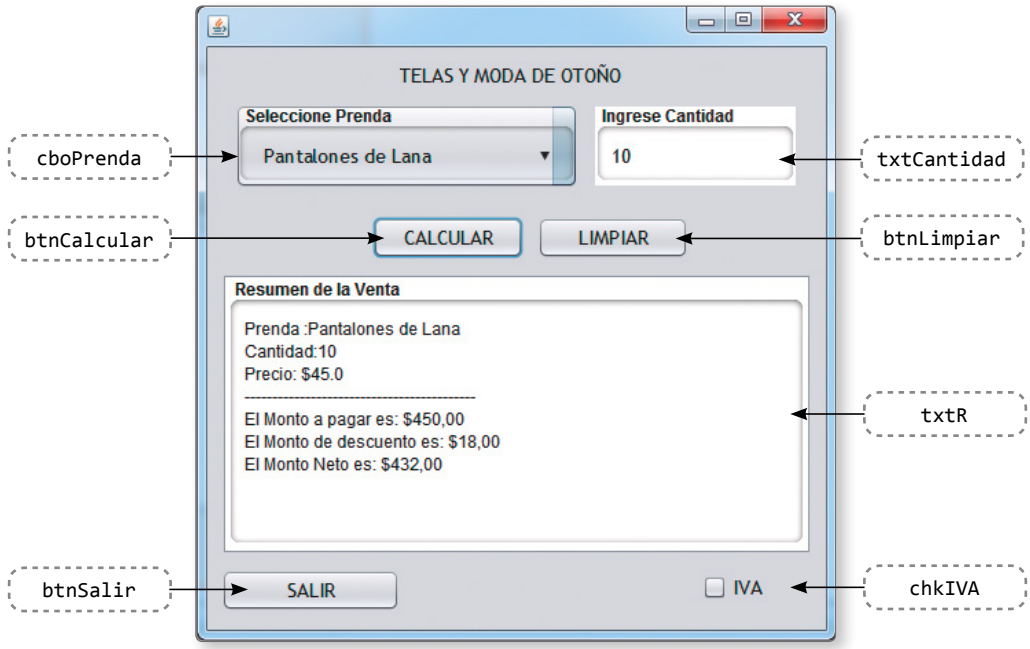
PRENDAS	PRECIO UNITARIO
Pantalones de lana	$ 45.00
Sueter de casimir	$ 100.00
Blusa de seda	$ 14.00
Camisola de seda	$ 10.00
Falda recta	$ 40.00
Saco de lana	$ 120.00

Al monto de la venta se le aplicará un descuento por temporada de acuerdo a la siguiente tabla:

MONTO DE LA VENTA	% DESCUENTO
Menor a $ 100.00	2%
Entre $ 100.00 y $ 500.00	4%
Entre $ 501.00 y $ 1000.00	6%
Entre $ 1001.00 y $ 1500.00	8%
Mayor a $ 1500.00	20%

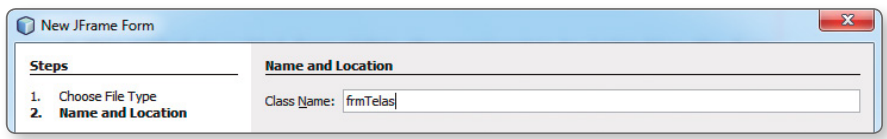
Implemente una aplicación que permita mostrar el monto de la venta, monto de descuento y el monto neto a pagar por la compra de ciertas prendas de un mismo tipo, adicionalmente mostrar el valor del IVA al 18% del monto neto a pagar.

GUI Propuesto:



Debe considerar:

- Agregar un nuevo Formulario al paquete pFormularios llamada frmTelas.



- Asigne nombres a todos los controles, para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…
- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la Fig. 6.5

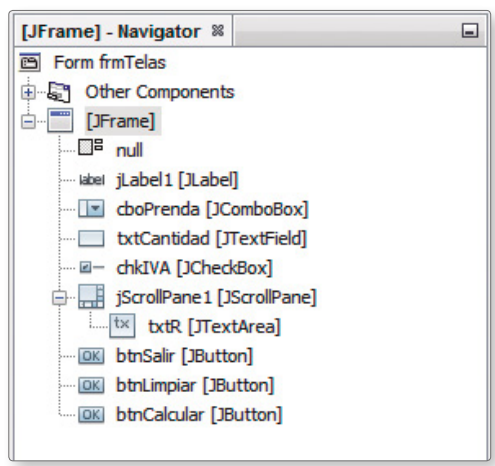


Fig. 6.5

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne las siguientes propiedades a los controles:

cboPrenda	**Border**	TitledBorder=Seleccione Prenda
	Text	Dejar vacio
txtCantidad	**Border**	TitledBorder=Ingrese Cantidad
	Text	Dejar vacio
txtR	**Border**	TitledBorder=Resumen de la Venta
	Text	Dejar vacio
btnCalcular	**Text**	CALCULAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de botones:

El siguiente script muestra la invocación al método llenaPrendas que tiene por misión llenar el cuadro combinado cboPrenda con los productos que ofrece la tienda. La llamada debe realizarse siempre después del initComponents:

```
public frmTelas() {
initComponents();
llenaPrendas();
}
```

El siguiente script muestra el código que se le asigna al botón btnCalcular como botón de respuesta al ingreso de valores en el Frame.

```
private void btnCalcularActionPerformed(…) {
    imprimir();
}
```

Al presionar el botón calcular se invoca al método imprimir ya que este imprime todos los valores de respuesta de la aplicación que se están delegando a los métodos con valor de retorno.

El siguiente script muestra el proceso que realizará el control chkIVA al ser activado o desactivado, hay que tener en cuenta que para llegar a dicho método se tiene que realizar los siguientes pasos:

- Clic derecho sobre el control chkIVA
- Seleccione Events > Item > itemStateChanged

```
private void chkIVAItemStateChanged(java.awt.event.ItemEvent evt) {
    if (chkIVA.isSelected()){
       imprimir();
       txtR.append("\nEl monto IVA es: $"+
                                String.format("%.2f",calculaIVA()));
       }else{
        imprimir();
    }
}
```

El siguiente script muestra la implementación de los métodos a usar en la aplicación:

```
//1
void llenaPrendas(){
    cboPrenda.addItem("Pantalones de Lana");
    cboPrenda.addItem("Sueter de casimir");
    cboPrenda.addItem("Blusa de seda");
    cboPrenda.addItem("Camisola de seda");
    cboPrenda.addItem("Falda recta");
    cboPrenda.addItem("Saco de lana");
}

//2
int getPrenda(){
    return cboPrenda.getSelectedIndex();
}

//3
int getCantidad(){
    return Integer.parseInt(txtCantidad.getText());
}

//4
double asignaPrecio(){
    switch(getPrenda()){
       case 0: return 45;
       case 1: return 100;
       case 2: return 14;
       case 3: return 10;
       case 4: return 40;
       default: return 120;
    }
}

//5
double calculaMonto(){
    return getCantidad()*asignaPrecio();
}

//6
double calculaDescuento(){
    if (calculaMonto()<=100)
        return 0.02*calculaMonto();
    else if (calculaMonto()<=500)
        return 0.04*calculaMonto();
    else if (calculaMonto()<=1000)
        return 0.06*calculaMonto();
    else if (calculaMonto()<=1500)
        return 0.08*calculaMonto();
    else
        return 0.20*calculaMonto();
}

//7
double calculaNeto(){
    return calculaMonto()-calculaDescuento();
}

//8
double calculaIVA(){
    return calculaNeto()*0.18;
}
```

```
//9
void imprimir(){
    txtR.setText("");
    txtR.append("Prenda :"+cboPrenda.getSelectedItem());
    txtR.append("\nCantidad:"+getCantidad());
    txtR.append("\nPrecio: $"+asignaPrecio());
    txtR.append("\n----------------------------------------");
    txtR.append("\nEl Monto a pagar es: $"+
                    String.format("%.2f",calculaMonto()));
    txtR.append("\nEl Monto de descuento es: $"+
                    String.format("%.2f",calculaDescuento()));
    txtR.append("\nEl Monto Neto es: $"+
                    String.format("%.2f",calculaNeto()));
}
```

En el punto uno se muestra la implementación del método llenaPrendas que tiene por misión enviar las prendas de la tienda al control cboPrenda.

En el punto dos se implementa el método getPrenda que tiene por misión obtener la prenda seleccionada por el usuario desde el control cboPrenda, el valor capturado desde el cuadro combinado es el índice asignado a cada prenda. Es decir:

(0) Pantalones de Lana
(1) Sueter de casimir
(2) Blusa de seda
(3) Camisola de seda
(4) Falda recta
(5) Saco de lana

En el punto tres se implementa el método getCantidad, el cual captura la cantidad ingresada por el usuario al momento de comprar un producto.

En el punto cuatro se implementa el método asignaPrecio que permite devolver el precio del producto seleccionado por el usuario, para esto se usa la instrucción switch y la variable a comparar es el método getPrenda que captura la prenda seleccionada por el usuario. Cuando se aplica un retorno directamente no es necesario colocar la instrucción break.

En el punto cinco se implementa el método calculaMonto, el cual permite determinar el monto a cancelar por el usuario estos datos se obtienen de los métodos getCantidad y asignaPrecio.

En el punto seis se implementa el método calculaDescuento, que permite determinar el monto de descuento según el monto obtenido para esto invocamos al método calculaMonto. La instrucción return se colocó dentro de todos los casos pero para no generar errores necesitamos especificar else dentro de la instrucción if.

En el punto siete se implementa el método calculaNeto que permite calcular el neto a cancelar por el cliente, en este caso se invoca al método calculaMonto y calculaDescuento para realizar una substracción entre ellos.

En el punto ocho se implementa el método calculaIVA que permite calcular el monto IVA aplicado al monto neto, esto se logra invocando al método calculaNeto.

En el punto nueve se implementa el método imprimir que permitirá mostrar los resultados de la aplicación, hay que tener en cuenta que este es el único método que se invoca en el botón btnCalcular; por lo tanto, este método tendrá la llamada a todos los métodos para un resultado adecuado.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
txtCantidad.setText("");
txtR.setText("");
txtCantidad.requestFocus();
}
```

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Telas y moda de Otoño",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

Caso desarrollado 6: Medios de publicidad con parámetros

La empresa Claire ha contratado a una compañía de publicidad para que le ayude a diseñar una campaña promocional a nivel nacional, para lo cual ha identificado tres medios de publicidad más efectivos para este producto:

Medios de Publicidad	Costo $
Comerciales de television	$ 30,000.00
Anuncios en revistas	$ 15,000.00
Anuncios en suplementos dominicales	$ 25,000.00

Como oferta la compañía ofrece un descuento de acuerdo a la siguiente tabla:

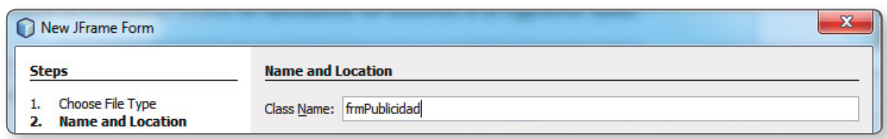
Total de Contratos	Descuento %
Hasta 5	10%
De 6 a 10	25%
Mas de 10	30%

Implemente una aplicación que permita seleccionar los medios de publicidad y la cantidad a contratar para mostrar el monto bruto, descuento y el monto neto a pagar por la empresa.

GUI Propuesto:

Debe considerar:

- Agregar un nuevo Formulario al paquete pFormularios llamada frmPublicidad.



- Asigne nombres a todos los controles para esto debe presionar clic derecho sobre el objeto y seleccionar > Change Variable Name…
- Después de asignar los nombres de los objetos el panel Navigator debe mostrarse como la Fig. 6.6.

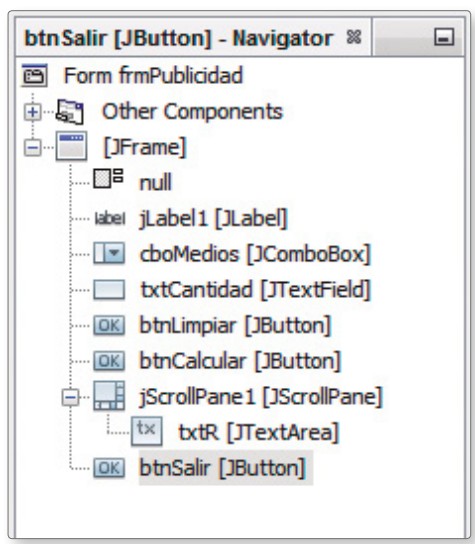


Fig. 6.6

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

cboMedios	**Border**	TitledBorder=Medios de Publicidad
	Model	Dejar vacio
txtCantidad	**Border**	TitledBorder=Cantidad de Contratos
	Text	Dejar vacio
txtR	**Text**	Dejar vacio
btnCalcular	**Text**	CALCULAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de botones:

El siguiente script muestra la invocación al método llenaMedios que se encargará de llenar el cuadro combinado de los medios de publicidad, este llamado se ha realizado dentro del método constructor de la clase principal.

```
public frmPublicidad() {
initComponents();
llenaMedios();
}
```

El siguiente script muestra el código que se le asigna al botón btnCalcular como botón de respuesta al ingreso de valores en el Frame.

```
//1
void llenaMedios(){
cboMedios.addItem("Comerciales de Television");
cboMedios.addItem("Anuncios en Revistas");
cboMedios.addItem("Anuncios en suplementos dominicales");
}

//2
int getMedio(){
return cboMedios.getSelectedIndex();
}

//3
int getCantidad(){
return Integer.parseInt(txtCantidad.getText());
}

//4
double asignaPrecio(int medio){
switch(medio){
    case 0: return 30000;
    case 1: return 15000;
    default: return 25000;
}
}
```

```
//5
double calculaBruto(int cantidad,double precio){
return cantidad*precio;
}

//6
double calculaDescuento(double bruto){
if (bruto<=5)
   return bruto*0.1;
   else if (bruto<=10)
   return bruto*0.25;
   else
   return bruto*0.3;
}

//7
double calculaNeto(double bruto,double descuento){
return bruto-descuento;
}

//8
void imprimir(double precio,double bruto,
              double descuento,double neto){
txtR.setText("");
txtR.append("Medio de Publicidad: "+
                              cboMedios.getSelectedItem());
txtR.append("\nPrecio: $ "+String.format("%.2f",precio));
txtR.append("\n----------------------------------------");
txtR.append("\nMonto Bruto: $ "+String.format("%.2f",bruto));
txtR.append("\nMonto de Descuento: $ "+
                              String.format("%.2f",descuento));
txtR.append("\nMonto Neto: $ "+String.format("%.2f",neto));
}
```

En el punto uno se implementa el método llenaMedios que es el encargado de llenar el control cboMedios con los medios de publicidad proporcionada por el caso.

En el punto dos se implementa el método que captura el medio de publicidad seleccionado por el usuario, aquí también se está usando el índice de cada medio por tal motivo el método tiene como retorno un valor entero.

En el punto tres se implementa el método que captura la cantidad ingresada por el usuario.

En el punto cuatro se implementa el método que permite asignar un precio a la publicidad seleccionada, dicha publicidad pasa como parámetro de tipo entero al método y usando la instrucción switch determina el precio según la tabla propuesta por el caso. Hay que tener en cuenta que al asignar un return al case no será necesario especificar la instrucción break.

En el punto cinco se implementa el método que permite determinar el monto bruto de la venta, para esto se envía como parámetros la cantidad y el precio de la publicidad seleccionada.

En el punto seis se implementa el método que permite determinar el monto de descuento según la tabla proporcionada por el caso. Se tiene que considerar que habiendo un return por cada expresión obligatoriamente debe colocar la cláusula else ya que al no colocarla generaría un error de retorno, pues todo método siempre devuelve un valor.

En el punto siete se implementa el método que permite calcular el neto a cancelar por el cliente para esto necesita como parámetro el monto bruto y el monto de descuento.

En el punto ocho se implementa el método imprimir que tendrá por misión imprimir los valores resultantes de la aplicación, hay que tener en cuenta que los valores a imprimir se deben enviar como parámetros a dicho método.

En el siguiente script se muestra el código asignado en el botón calcular:

```
private void btnCalcularActionPerformed(…) {
        //1
        int medio=getMedio();
        int cantidad=getCantidad();

        //2
        double precio=asignaPrecio(medio);

        //3
        double bruto=calculaBruto(cantidad, precio);
        double descuento=calculaDescuento(bruto);
        double neto=calculaNeto(bruto,descuento);

        //4
        imprimir(precio,bruto,descuento,neto);
}
```

En el punto uno se declara la variable medio que almacenará el medio de publicidad obtenido desde el método getMedio, recuerde que este método devuelve un valor entero es por esa razón que la variable medio también es entera. La variable cantidad almacena la cantidad obtenida desde el método getCantidad.

En el punto dos se declara la variable precio que tendrá por misión almacenar el precio obtenido desde el método asignaPrecio aquí es el momento de enviarle el valor del medio de publicidad por el parámetro.

En el punto tres se almacena el monto bruto obtenido desde el método calculaBruto, para esto se le envía como parámetros a la cantidad y el precio obtenidos en los pasos anteriores. De la misma forma para la variable descuento y la variable neto.

En el punto cuatro se invoca al método imprimir enviando los valores obtenidos desde las variables.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
cboMedios.setSelectedIndex(-1);
txtCantidad.setText("");
cboMedios.requestFocus();
}
```

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Publicidad",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

6.7. VALIDACIÓN DE DATOS

Un proceso de validación es una de las acciones más importantes que tiene que implementar un desarrollador, la razón seguro que usted lo ha notado a lo largo de las aplicaciones vertidas en este material, hasta este punto no se ha comprobado, ni verificado o controlado algún valor que el usuario ha ingresado; normalmente esto se le conoce como descuido del desarrollador pero en este caso no lo podíamos implementar ya que serian muchas líneas de código, con el tema de programación modular si se implementa mejor las validaciones de los valores. Vea algunos casos no validados hasta el momento por usted:

- Valores vacíos en los controles, imagine usted no ingresar la cantidad de horas a un trabajador ¿Cómo podría calcular su sueldo?
- Ingresar valores fuera del rango establecido por la aplicación; por ejemplo, si necesita ingresar las notas de un alumno (si sabe que son entre 0 y 20) no estaría permitido el ingreso de un valor negativo o un valor superior a 20.
- No seleccionar un elemento del cuadro combinado (JComboBox), cuando una aplicación depende de la selección realizada sobre un JComboBox definitivamente se mostrarán errores.

Ahora se verán algunos ejemplos de cómo validar los valores:

Ejemplo :

Controlar un objeto JTextField vacío; por ejemplo, txtDescripcion.

```
if (txtDescripcion.equals("")){
   JOptionPane.showMessageDialog(null,
               "Ingrese una descripcion…!!!");
}
```

Ejemplo :

Controlar el valor numérico de un objeto JTextField; por ejemplo, txtSueldo solo permite ingresar montos entre 1000 y 2000.

```
if (Double.parseDouble(txtSueldo.getText())>=1000
        && Double.parseDouble(txtSueldo.getText())<=2000){
   JOptionPane.showMessageDialog(null,
                "Sueldo valido…!!!");
}
```

Ejemplo :

Controlar si el usuario ha seleccionado un elemento del JComboBox; por ejemplo, cboProducto.

```
if (cboProducto.getSelectedIndex()==-1){
   JOptionPane.showMessageDialog(null,
            "Debe seleccionar un producto…!!!");
}
```

CASO DESARROLLADO 7: PROMEDIO DE NOTAS

Aplicación que permita determinar el promedio de 4 notas de un alumno para lo cual se solicita lo siguiente:

- *Determinar la más alta nota registrada por el alumno.*
- *Determinar la nota más baja registrada por el alumno.*
- *Mostrar un mensaje de acuerdo a su promedio; si está aprobado determinar por cuántos puntos aprobó, caso contrario determinar por cuántos puntos desaprobó.*

GUI Propuesto:

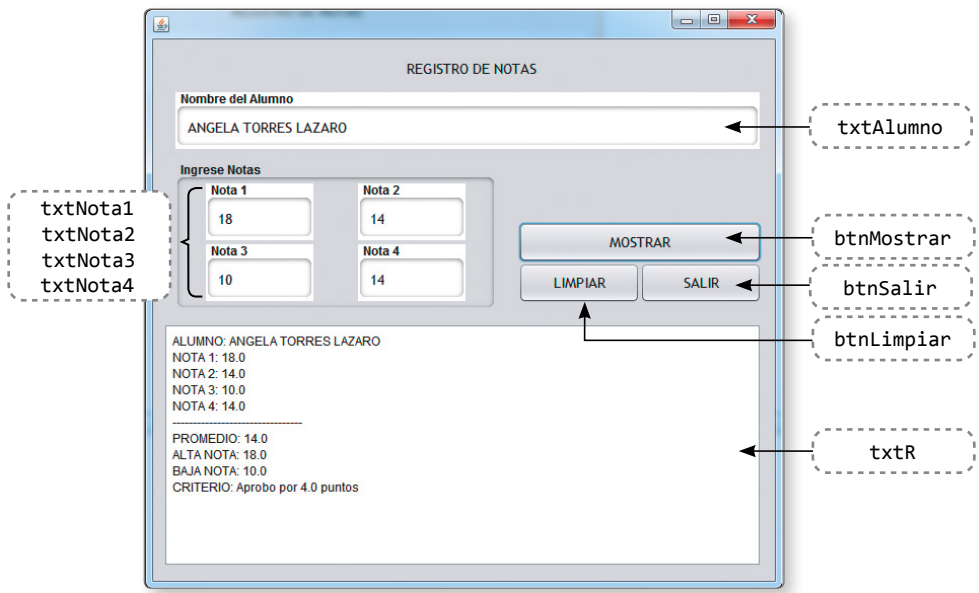


Fig. 6.7

Antes de dar solución al problema, primero se analizará el caso y se listará los métodos necesarios para un buen proceso de datos de la aplicación:

PROCESOS	METODOS	TIPO DEVUELTO
Obtener el nombre del alumno	getAlumno()	string
Obtener la nota 1	getNota1()	double
Obtener la nota 2	getNota2()	double
Obtener la nota 3	getNota3()	double
Obtener la nota 4	getNota4()	double
Calcular el promedio de notas	calculaPromedio(nota1,nota2,nota3,nota4)	double
Determinar la Nota mas alta	determinaAltaNota(nota1,nota2,nota3,nota4)	double
Determinar la Nota mas baja	determinaBajaNota(nota1,nota2,nota3,nota4)	double

PROCESOS	METODOS	TIPO DEVUELTO
Mostrar un mensaje de acuerdo a su promedio	determinaCriterio(promedio)	string
Validar los datos ingresados	valida()	void
Limpiar los controles	limpiar()	void
Imprimir los resultados	imprimir(promedio,altaNota,bajaNota,criterio)	void

La lista de procesos son las actividades que se deben realizar en la aplicación, mientras que la columna métodos determina como se llamarán los métodos y finalmente tipo de devuelto determina qué tipo de datos es devuelto por el método. Estos puntos los debe tener bien claros antes de comenzar a desarrollar la aplicación.

Luego en la aplicación debe considerar:

- Agregar un nuevo Formulario al paquete pFormularios y asígnele el nombre frmPromedio.

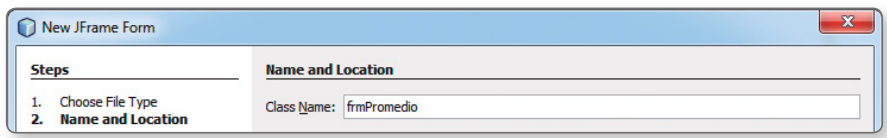


Fig. 6.8

- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 6.8, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos, para eso visualice el panel Navigator; debe mostrarse como la Fig. 6.9.

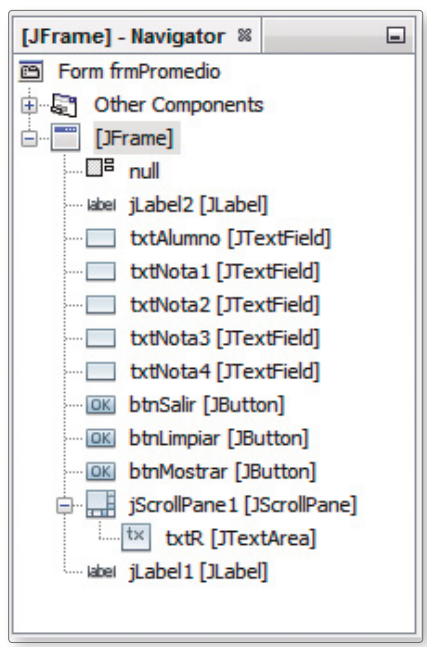


Fig. 6.9

- Asignar Null Layout a la opción setLayout del Frame; para esto presione clic derecho sobre el Frame>setLayout>Null Layout.
- A la opción **Form Size Police** del objeto Frame asígnele el valor **Generate Resize Code**.
- Asigne las siguientes propiedades a los controles:

txtAlumno	**Border**	TitledBorder=Nombre del Alumno
	Text	Dejar vacío
txtNota1	**Border**	TitledBorder=Nota 1
	Text	Dejar vacío
txtNota2	**Border**	TitledBorder=Nota 2
	Text	Dejar vacío
txtNota3	**Border**	TitledBorder=Nota 3
	Text	Dejar vacío
txtNota4	**Border**	TitledBorder=Nota 4
	Text	Dejar vacío
txtR	**Text**	Dejar vacío
btnMostrar	**Text**	MOSTRAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- A continuación se explicará el script completo de la aplicación.

El siguiente script muestra todos los métodos implementados para la aplicación:

```
//1
String getAlumno(){
return txtAlumno.getText();
}

//2
double getNota1(){
return Double.parseDouble(txtNota1.getText());
}

//3
double getNota2(){
return Double.parseDouble(txtNota2.getText());
}

//4
double getNota3(){
return Double.parseDouble(txtNota3.getText());
}

//5
double getNota4(){
      return Double.parseDouble(txtNota4.getText());
}
```

```
//6
String valida(){
      if (getAlumno().equals("")){
            txtAlumno.setText("");
            txtAlumno.requestFocus();
            return "Nombre del Alumno";
      } else if (txtNota1.getText().equals("") || getNota1()<0 ||
                  getNota1()>20){
            txtNota1.setText("");
            txtNota1.requestFocus();
            return "Nota 1";
       } else if (txtNota2.getText().equals("") || getNota2()<0 ||
                  getNota2()>20){
            txtNota2.setText("");
            txtNota2.requestFocus();
            return "Nota 2";
       } else if (txtNota3.getText().equals("") || getNota3()<0 ||
                  getNota3()>20){
            txtNota3.setText("");
            txtNota3.requestFocus();
            return "Nota 3";
       } else if (txtNota4.getText().equals("") || getNota4()<0 ||
                  getNota4()>20){
            txtNota4.setText("");
            txtNota4.requestFocus();
            return "Nota 4";
       } else
           return "";
}

//7
double calculaPromedio(double n1,double n2,double n3,double n4){
     return (n1+n2+n3+n4)/4.0;
}

//8
double determinaAltaNota(double n1,double n2,double n3,double n4){
     double mayor=0;
     if (n1>n2) mayor=n1; else mayor=n2;
     if (n3>mayor) mayor=n3;
     if (n4>mayor) mayor=n4;
     return mayor;
}

//9
double determinaBajaNota(double n1,double n2,double n3,double n4){
     double menor=0;
     if (n1<n2) menor=n1; else menor=n2;
     if (n3<menor) menor=n3;
     if (n4<menor) menor=n4;
     return menor;
}

//10
String determinaCriterio(double promedio){
     if (promedio>10)
        return "Aprobo por "+String.valueOf(promedio-10)+" puntos";
     else
        return "Desaprobo por "+String.valueOf(11-promedio)+" puntos";
}
```

```
//11
void imprimir(  double promedio,double altaNota,
                double bajaNota, String criterio){
        txtR.setText("");
        txtR.append("ALUMNO: "+getAlumno());
        txtR.append("\nNOTA 1: "+getNota1());
        txtR.append("\nNOTA 2: "+getNota2());
        txtR.append("\nNOTA 3: "+getNota3());
        txtR.append("\nNOTA 4: "+getNota4());
        txtR.append("\n-------------------------------");
        txtR.append("\nPROMEDIO: "+promedio);
        txtR.append("\nALTA NOTA: "+altaNota);
        txtR.append("\nBAJA NOTA: "+bajaNota);
        txtR.append("\nCRITERIO: "+criterio);
}

//12
void limpiar(){
        txtAlumno.setText("");
        txtNota1.setText("");
        txtNota2.setText("");
        txtNota3.setText("");
        txtNota4.setText("");
        txtAlumno.requestFocus();
}
```

En el punto uno se implementa el método getAlumno que permite capturar el nombre ingresado en el control txtAlumno.

En el punto dos se implementa el método getNota1, que permite obtener la nota registrada por el alumno en el control txtNota1. En el punto tres se implementa el método getNota2, que permite obtener la nota registrada por el alumno en el control txtNota2. En el punto cuatro se implementa el método getNota3, que permite obtener la nota registrada por el alumno en el control txtNota3. En el punto cinco se implementa el método getNota4, que permite obtener la nota registrada por el alumno en el control txtNota4.

En el punto seis se implementa el método valida, que permite controlar el ingreso de los valores por medio de los controles, la idea principal es: en el caso de los valores tipo cadena solo verificar si el control no está vacío, en caso sean valores tipo número verificar si el control no está vacío, además verificar que el valor sea el correcto; por ejemplo, un valor negativo o superior a 20 no sería una nota válida. Además, por cada evaluación se limpiará la caja y se enviará el foco a dicho control. Finalmente, si todo es correcto el método valida devuelve un cadena vacía, cuando esto ocurra la aplicación determinará que todos los datos fueron ingresados correctamente.

En el punto siete se implementa el método calculaPromedio, que permite calcular el promedio dependiendo de las notas enviadas como parámetros.

En el punto ocho se implementa el método determinaAltaNota, que permite determinar cuál de las 4 notas ingresadas como parámetros es la mayor y esta será devuelta por el método.

En el punto nueve ocurre algo parecido al punto ocho, pero esta vez el método devolverá la menor nota de las 4 ingresadas como parámetros.

En el punto diez se implementa el método determinaCriterio que permitirá determinar un mensaje de acuerdo el promedio ingresado como parámetro. Hay que considerar que para concatenar el texto “Aprobo por" con un valor numérico; este último deberá ser convertido a un valor de tipo texto, para esto se usa el método valueOf de la clase String.

En el punto once se implementa el método imprimir que permitirá enviar los valores de respuesta de la aplicación al control txtR.

En el punto doce se implementa el método limpiar que permite limpiar los controles usados en la aplicación.

El siguiente script muestra el código que se le asigna al botón btnMostrar como botón de respuesta al ingreso de valores en la aplicación.

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
    //1
    if (valida().equals("")){
        //2
        String alumno=getAlumno();
        double nota1=getNota1();
        double nota2=getNota2();
        double nota3=getNota3();
        double nota4=getNota4();

        //3
        double promedio=calculaPromedio(nota1, nota2, nota3, nota4);

        //4
        Double altaNota=determinaAltaNota(nota1, nota2, nota3, nota4);
        double bajaNota=determinaBajaNota(nota1, nota2, nota3, nota4);
        String criterio=determinaCriterio(promedio);

        //5
        imprimir(promedio, altaNota, bajaNota, criterio);
      }else {
        JOptionPane.showMessageDialog(null,
                    "Verificar los datos en el campo "+valida());
      }
}
```

En el punto uno se verifica el valor devuelto por el método valida, recuerde que este método tiene la capacidad de devolver un texto indicando donde ocurrió un error, en caso devuelva vacío será un indicador que los valores han sido ingresados correctamente.

En el punto dos se declaran las variables locales alumno, nota1, nota2, nota3 y nota4 que permitirán almacenar los valores ingresados por el usuario. Estos valores capturados están representados por métodos ya implementados como getAlumno, getNota1, getNota2, getNota3 y getNota4 respectivamente.

En el punto tres se declara la variable promedio que permitirá almacenar el promedio obtenido desde el método calculaPromedio, si recordamos su implementación este método necesita de 4 parámetros que son enviados desde el botón Mostrar; por ese motivo se declararon variables locales para las notas ya que en algún momento serían enviados al método calculaPromedio.

En el punto cuatro se declaran las variables altaNota que tienen por misión almacenar la mayor nota registrada para el alumno; esto será gracias al método determinaAltaNota para lo cual se deben enviar las 4 notas para que sean evaluadas. De la misma forma se realizará para la variable bajaNota con el método determinaBajaNota. Y finalmente, la variable criterio almacenara el mensaje de los puntos obtenidos por el alumno, para esto se necesita enviar como parámetro el promedio.

En el punto cinco se invoca al método imprimir para poder mostrar los valores obtenidos como respuesta de la aplicación, este método tiene como parámetros el promedio, la más alta nota (altaNota), la más baja nota (bajaNota) y el criterio obtenido por el alumno.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
  private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
  limpiar();
  }
```

Como verá solo es cuestión de invocar al método que limpia los controles.

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Validacion…!!!",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

CAP.

7

Estructura de repetición

CAPACIDAD:

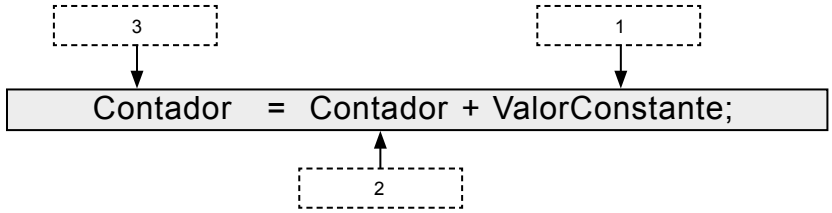
- Implementar aplicaciones usando los contadores y acumuladores de programación.
- Implementar aplicaciones usando estructuras de repetición.

CONTENIDO:

7.1. CONTADORES

Es una variable de tipo entero cuyo valor acumula las veces que se pasa por ella. Se suele denominar contador cuando la variable incrementa o decrementa de 1 en 1.

Formato:



En el punto uno se define el valor constante del contador; por ejemplo, para un contador de incremento se usaría +1, en el caso de decremento sería -1.

En el punto dos se especifica la variable contadora con el último valor asignado, este valor será parte de la nueva expresión.

En el punto tres se asigna un nuevo valor a la variable contadora, dicho valor es el resultado de la expresión especificada en el punto uno y dos.

Hay que tener en cuenta que cuando vimos el tema de operadores en Java se listó los operadores de incremento y decremento, veamos algunas analogías sobre los contadores:

CONTAR A LOS EMPLEADOS DE UNA EMPRESA		
cEmpleados = cEmpleados + 1;	cEmpleados++;	cEmpleados+=1;

CONTAR EL NUMERO DE NOTAS REGISTRADAS, PARA ESTE CASO APLICAREMOS UN CONTEO EN FORMA DE DECREMENTO		
cNotas = cNotas - 1;	cNotas--;	cNotas-=1;

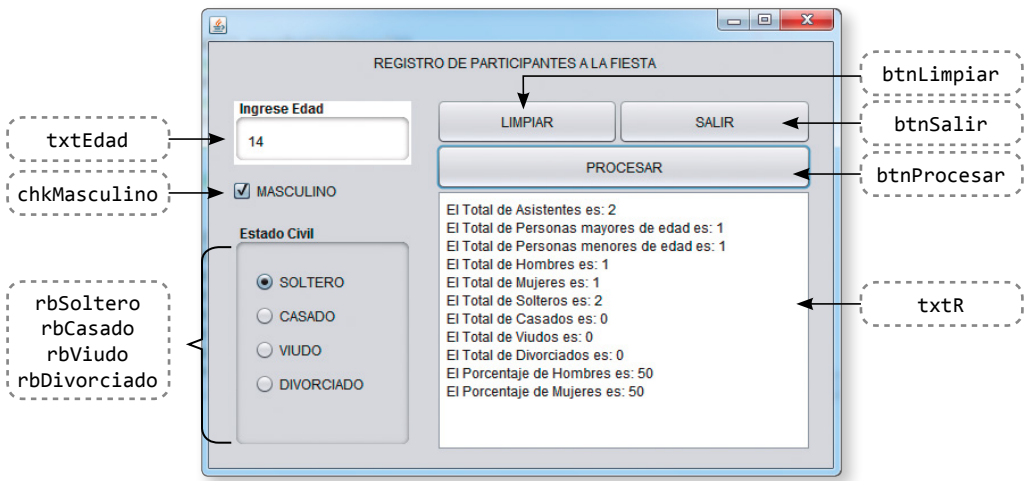
CONTAR EL NUMERO DE VECES QUE EL USUARIO REALIZA UN CLIC SOBRE UN BOTÓN, PARA ESTE CASO USAREMOS UNA CONSTANTE 2		
cBoton = cBoton + 2;	cBoton++; cBoton++;	cBoton+=2;

CASO DESARROLLADO 1: CONTADOR DE ASISTENCIA A UNA FIESTA

Implemente una aplicación que permita controlar los asistentes a una fiesta; para esto deberá ingresar la edad, sexo(masculino y femenino) y estado civil del asistente (soltero, casado, viudo o divorciado), luego mostrar las siguientes estadísticas:

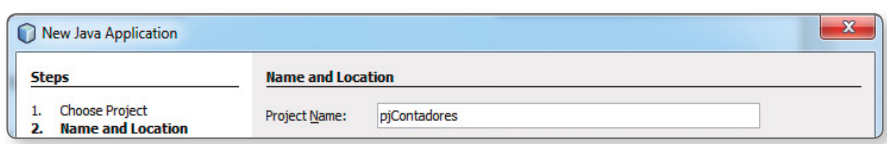
- *Total de Asistentes*
- *Total de Personas Mayores de Edad*
- *Total de Personas Menores de Edad*
- *Total de Hombres*
- *Total de Mujeres*
- *Total de Solteros*
- *Total de Casados*
- *Total de Viudos*
- *Total de Divorciados*
- *Porcentaje de Hombres*
- *Porcentaje de Mujeres*

GUI Propuesto:

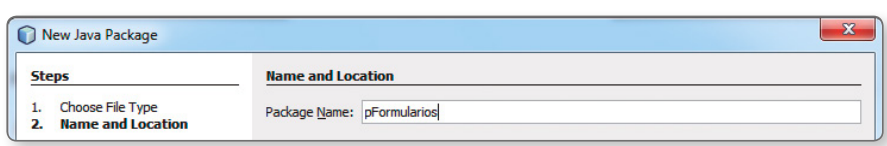


Debe considerar:

- Crear un nuevo proyecto en NetBeans llamado pjModular.



- Agregar un nuevo paquete al proyecto pjModular llamado pFormularios.



- Agregar un nuevo Formulario al paquete pFormularios llamada frmFiesta.

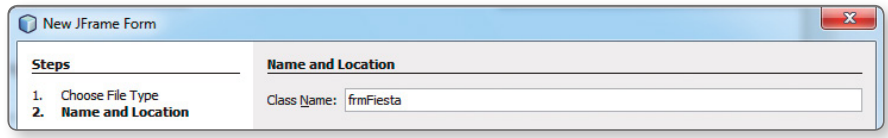


Fig. 7.1

- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 7.1, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos, para eso visualice el panel Navigator; debe mostrarse como la Fig. 7.2

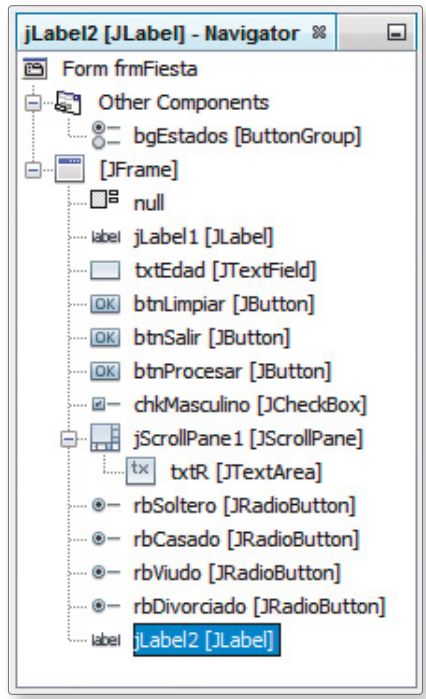


Fig. 7.2

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtEdad	**Border** **Text**	TitledBorder=Monto total de la compra Dejar vacio
chkMasculino	**Text**	MASCULINO
bgEstados	**Variable Name**	bgEstados
rbSoltero	**ButtonGroup** **Text**	bgEstados SOLTERO
rbCasado	**ButtonGroup** **Text**	bgEstados CASADO
rbViudo	**ButtonGroup** **Text**	bgEstados VIUDO
rbDivorciado	**ButtonGroup** **Text**	bgEstados DIVORCIADO
txtR	**Text**	Dejar vacio
btnProcesar	**Text**	PROCESAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Vea el script de la aplicación:

  El siguiente script muestra la declaración de las variables globales de la aplicación.

```
public class frmFiesta extends javax.swing.JFrame {

//Declaracion de variables Globales
int total,tHombres,tMujeres;
int tMayores,tMenores;
int tSolteros,tCasados,tViudos,tDivorciados;

    public frmFiesta() {
initComponents();
    }
```

Java propone que solo deben ser declaradas como variables globales aquellas que permiten contar o acumular algún valor, también debe considerar que toda variable global es inicializada de forma automática.

El siguiente script muestra todos los métodos implementados en la aplicación.

```
//1
int getEdad(){
    return Integer.parseInt(txtEdad.getText());
}

//2
boolean getMasculino(){
    if (chkMasculino.isSelected())
       return true;
    else
       return false;
}
```

```
//3
String getEstado(){
    if (rbSoltero.isSelected())
       return "Soltero";
    else if (rbCasado.isSelected())
       return "Casado";
    else if (rbViudo.isSelected())
       return "Viudo";
    else
       return "Divorciado";
}

//4
String valida(){
    if (txtEdad.getText().equals("") ||
                      Integer.parseInt(txtEdad.getText())<0)
       return "Edad del Asistente..!!!";
    else if (!(rbSoltero.isSelected() || rbCasado.isSelected() ||
                rbViudo.isSelected() || rbDivorciado.isSelected()))
       return "Estado Civil del Asistente..!!!";
    else
       return "";
    }

//5
void estadisticas(){
    //
    total++;

    //
    if (getEdad()>=18)
        tMayores++;
    else
        tMenores++;

    //
    if (getMasculino()==true)
        tHombres++;
    else
        tMujeres++;

    //
    if (getEstado().equals("Soltero"))
        tSolteros++;
    else if (getEstado().equals("Casado"))
        tCasados++;
    else if (getEstado().equals("Viudo"))
        tViudos++;
    else
        tDivorciados++;
}
```

```
//6
void imprimir(){
     txtR.setText("");
     txtR.append("El Total de Asistentes es: "+total);
     txtR.append("\nEl Total de Personas mayores es: "+tMayores);
     txtR.append("\nEl Total de Personas menores es: "+tMenores);
     txtR.append("\nEl Total de Hombres es: "+tHombres);
     txtR.append("\nEl Total de Mujeres es: "+tMujeres);
     txtR.append("\nEl Total de Solteros es: "+tSolteros);
     txtR.append("\nEl Total de Casados es: "+tCasados);
     txtR.append("\nEl Total de Viudos es: "+tViudos);
     txtR.append("\nEl Total de Divorciados es: "+tDivorciados);
     txtR.append("\n % de Hombres es: "+(tHombres*100)/total);
     txtR.append("\n % de Mujeres es: "+(tMujeres*100)/total);
}

//7
void limpiar(){
     txtEdad.setText("");
     chkMasculino.setSelected(false);
     rbSoltero.setSelected(false);
     rbCasado.setSelected(false);
     rbViudo.setSelected(false);
     rbDivorciado.setSelected(false);
     txtEdad.requestFocus();
}

//8
void reinicializaVariables(){
     total=0;
     tHombres=0;
     tMujeres=0;
     tMayores=0;
     tMenores=0;
     tSolteros=0;
     tCasados=0;
     tViudos=0;
     tDivorciados=0;
}
```

En el punto uno se implementa el método getEdad, que permite capturar la edad registrada por el usuario desde la GUI.

En el punto dos se implementa el método getMasculino, que permite determinar si el usuario seleccionó el control chkMasculino; en caso sea así el retorno será el valor true caso contrario false. Hay que tener en cuenta que para determinar la selección se usa el método isSelected.

En el punto tres se implementa el método getEstado, que permite obtener una cadena que represente la opción seleccionada por el usuario, al igual que el control chkMasculino el control JRadioButton cuenta con el método isSelected para determinar si fue o no seleccionado.

En el punto cuatro se implementa el método valida que permite comprobar y verificar los valores ingresados o seleccionados por el usuario, al control txtEdad se le comprobará que no sea un valor vacío y que su valor no sea menor a 0 ya que nadie puede tener una edad inferior a cero. Luego se evalúa si el usuario ha seleccionado por lo menos un estado civil, en vista que el método isSelected

determina que el usuario seleccione una opción. En este caso negamos esa selección para eso usamos el operador (!). Finalmente, si todo es correcto el método devuelve un valor textual vacío.

En el punto cinco se implementa el método estadísticas que determina todos los cálculos que solicita la aplicación, en vista que este método tiene diferentes cálculos su valor de devolución es vacío (void). La instrucción total++; determina el total de asistentes a la fiesta, tMayores y tMenores; determina el total de personas mayores y menores de edad evaluados desde la condicion getEdad()>=18, considere que getEdad devuelve la edad ingresada por el usuario desde la GUI. tHombres y tMujeres determina el total de hombres y mujeres asistentes a la fiesta, evaluados desde la condición getMasculino()==true, considere que el método getMasculino devuelve solo true o false. tSolteros, tCasados, tViudos y tDivorciados determinan el total personas con dichos estados civiles, hay que considerar que el método getEstado devuelve ese valor en forma de cadena, por tal motivo se usa el método equals al momento de evaluarlos.

En el punto seis se implementa el método imprimir que permite devolver todos los valores calculados al control txtR.

En el punto siete se implementa el método limpiar que tiene la misión de limpiar todos los controles del Frame y devolver el foco al control txtEdad.

En el punto ocho se implementa el método reinicializaVariables que tiene por misión devolver el valor inicial a cero a todas las variables globales, este método será usado cuando el usuario seleccione el botón Limpiar.

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnProcesarActionPerformed(…) {
        if (valida().equals("")){
            estadisticas();
            limpiar();
            imprimir();
        } else
            JOptionPane.showMessageDialog(null,
                                    "Ocurrio un error en "+valida());
    }
```

Lo primero es invocar al método valida para determinar si todos los valores ingresados por el usuario son los correctos; caso contrario enviar un mensaje con el nombre del objeto errado. En caso no genere error el método valida se procede a invocar al método estadísticas, el cual contabilizara los valores y los registrará en las variables declaradas como global, se limpiarán los controles invocando al método limpiar y finalmente se imprimirán los resultados.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
        limpiar();
        txtR.setText("");
        reinicializaVariables();
}
```

Como verá dentro del botón btnLimpiar se invoca al método limpia que permitirá limpiar todos los controles usados en la aplicación y además reiniciará el valor inicial de todas las variables globales mediante el método reinicializaVariables.

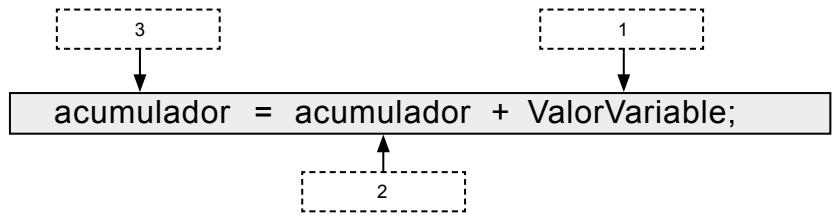
En el siguiente script se muestra como salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Acumuladores y Contadores",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

7.2. Acumuladores

Un acumulador o totalizador suele utilizarse para acumular resultados producidos en las iteraciones de un bucle o de llamadas sucesivas. Se parece en estructura a los contadores con la diferencia que no tienen un valor constante sino más bien variable.

Formato:



En el punto uno se define el valor que se acumulará, normalmente está representado por una variable.

En el punto dos se hace referencia a la variable acumulador con el último valor asignado, este valor será parte de la nueva expresión.

En el punto tres se asigna un nuevo valor a la variable acumulador, dicho valor es el resultado de la expresión especificada en el punto uno y dos.

Veamos algunos ejemplos de acumuladores:

Acumular las edades de los empleados	
aEdad = aEdad + edad;	aEdad += Edad;

Acumular los sueldos de los empleados.	
aSueldo = aSueldo + sueldo;	aSueldo += sueldo;

CASO DESARROLLADO 2: APOYO EN RECEPCIÓN DE HOTEL

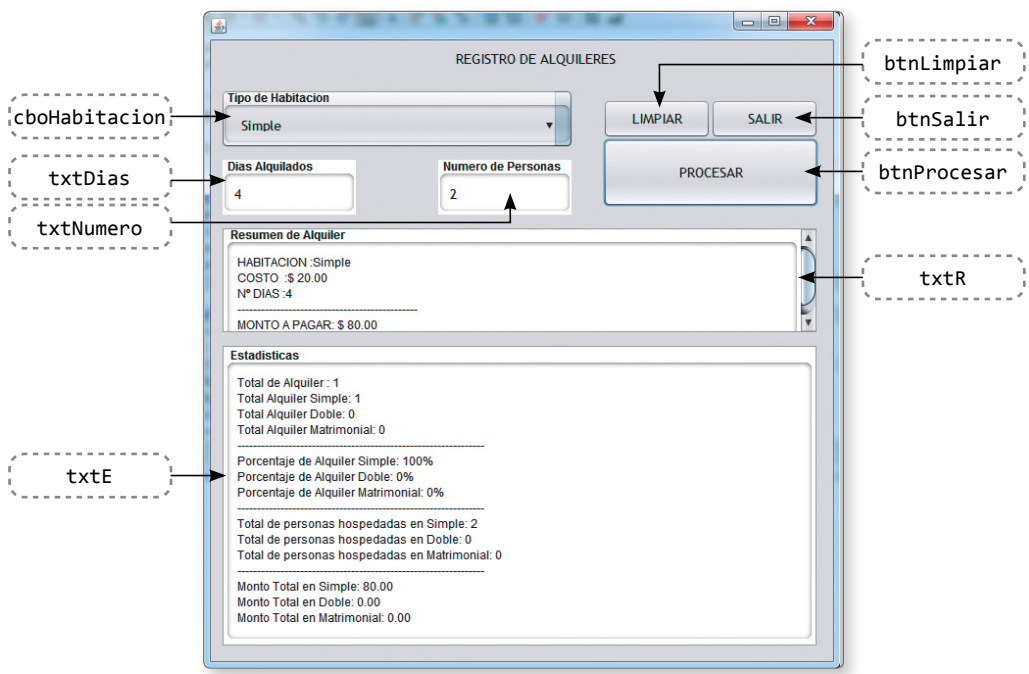
Implemente una aplicación que permita apoyar al área de recepción de un hotel. La aplicación debe registrar los alquileres que se realizan y el número de personas que se hospedan. El hotel cuenta con los siguientes tipos de habitación:

TIPO DE HABITACION	COSTO $
Simple	$ 20.00
Doble	$ 40.00
Matrimonial	$ 100.00

Al final determinar:

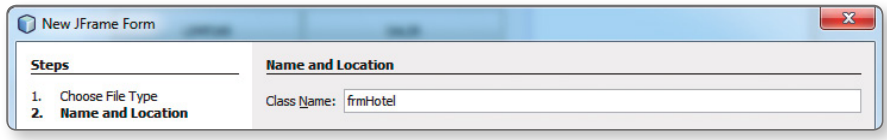
- *Costo por alquiler.*
- *Total de Alquileres por tipo de habitación.*
- *Reporte porcentual de alquileres por tipo de habitación.*
- *Reporte de personas hospedadas por tipo de habitación.*
- *Reporte de montos acumulados por tipo de habitación.*

GUI Propuesto:



Debe considerar:

- Agregar un nuevo Formulario al paquete pFormularios dentro del proyecto pjContadores llamada frmHotel.



- Asigne un nombre a cada uno de los controles; como se muestra en la Fig., para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Despues de asignar nombres a los objetos asegurese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig.

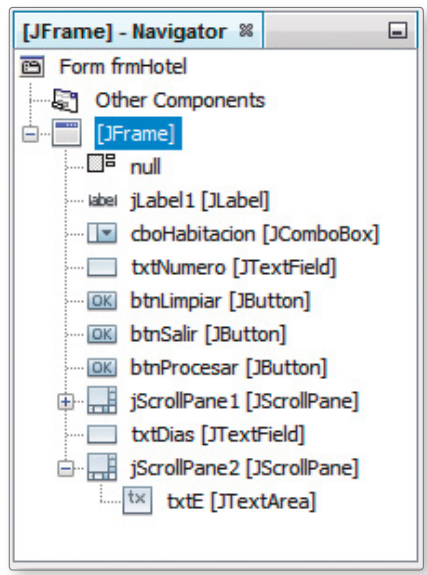


Fig. 7.3

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne las siguientes propiedades a los controles:

cboHabitacion	**Border**	TitledBorder=Tipo de Habitacion
	Model	Dejar vacio
txtDias	**Border**	TitledBorder=Dias Alquilados
	Text	Dejar vacio
txtNumero	**Border**	TitledBorder=Numero de Personas
	Text	Dejar vacio
txtR	**Border**	TitledBorder=Resumen de Alquiler
	Text	Dejar vacio
txtE	**Border**	TitledBorder=Estadisticas
	Text	Dejar vacio

btnProcesar	**Text**	PROCESAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR

- Veamos el script de la aplicación:

  El siguiente script muestra la declaración de las variables globales de la aplicación:

```
public class frmHotel extends javax.swing.JFrame {

    int cSimple,cDoble,cMatrimonial;
    int pSimple,pDoble,pMatrimonial;
    double aSimple,aDoble,aMatrimonial;

    public frmHotel() {
        initComponents();
        llenaHabitacion();
    }
```

Las variables que inician con el carácter .c" son aquellas que tienen la misión de contar cuantas veces se alquila un tipo de habitación, en el caso de las variables con carácter inicial .p" tiene por misión acumular el total de personas por cada tipo de habitación y las variables con carácter inicial .a" tienen por misión acumular los montos obtenidos en un determinado alquiler según el tipo de habitación seleccionada.

En el método constructor de la clase frmHotel; se invoca al método llenaHabitacion que permitirá llenar los tipos de habitaciones en el control cboHabitacion.

El siguiente script muestra todos los métodos implementados en la aplicación.

```
//1
void llenaHabitacion(){
    cboHabitacion.addItem("Simple");
    cboHabitacion.addItem("Doble");
    cboHabitacion.addItem("Matrimonial");
}

//2
int getHabitacion(){
    return cboHabitacion.getSelectedIndex();
}

//3
int getDias(){
    return Integer.parseInt(txtDias.getText());
}

//4
int getNumero(){
    return Integer.parseInt(txtNumero.getText());
}
```

```
//5
double determinaCosto(int habitacion){
     switch(habitacion){
        case 0: return 20;
        case 1: return 40;
        default: return 100;
     }
}

//6
double calculaMonto(int dias, double costo){
    return dias*costo;
}

//7
void resumen(double costo,int dias,double monto){
    txtR.setText("");
    txtR.append("HABITACION :"+cboHabitacion.getSelectedItem());
    txtR.append("\nCOSTO  :$ "+String.format("%.2f",costo));
    txtR.append("\nNº DIAS :"+dias);
    txtR.append("\n-----------------------------------------------");
    txtR.append("\nMONTO A PAGAR: $ "+String.format("%.2f",monto));
}

//8
void estadisticas(int numero, double monto){
    switch(getHabitacion()){
        case 0:
          cSimple++;
          pSimple+=numero;
          aSimple+=monto;
          break;
        case 1:
          cDoble++;
          pDoble+=numero;
          aDoble+=monto;
          break;
        default:
          cMatrimonial++;
          pMatrimonial+=numero;
          aMatrimonial+=monto;
     }
}

//9
void imprimir(){
    int total = cSimple+cDoble+cMatrimonial;
    txtE.setText("");
    txtE.append("Total de Alquiler : "+total);
    txtE.append("\nTotal Alquiler Simple: "+cSimple);
    txtE.append("\nTotal Alquiler Doble: "+cDoble);
    txtE.append("\nTotal Alquiler Matrimonial: "+cMatrimonial);
    txtE.append("\n-----------------------------------------------");
    txtE.append("\nPorcentaje de Alquiler Simple: "+
                                (cSimple*100)/total+"%");
    txtE.append("\nPorcentaje de Alquiler Doble: "+
                                (cDoble*100)/total+"%");
    txtE.append("\nPorcentaje de Alquiler Matrimonial: "+
                                (cMatrimonial*100)/total+"%");
    txtE.append("\n-----------------------------------------------");
    txtE.append("\nTotal de personas hospedadas en Simple: "+pSimple);
```

```
    txtE.append("\nTotal de personas hospedadas en Doble: "+pDoble);
    txtE.append("\nTotal de personas hospedadas en Matrimonial: "+
                                                    pMatrimonial);
    txtE.append("\n-----------------------------------------------");
    txtE.append("\nMonto Total en Simple: "+
                               String.format("%.2f",aSimple));
    txtE.append("\nMonto Total en Doble: "+
                               String.format("%.2f",aDoble));
    txtE.append("\nMonto Total en Matrimonial: "+
                               String.format("%.2f",aMatrimonial));
}

//10
void limpiar(){
    cboHabitacion.setSelectedIndex(-1);
    txtDias.setText("");
    txtNumero.setText("");
    cboHabitacion.requestFocus();
}

//11
void reiniciarGlobales(){
    cSimple=0;
    pSimple=0;
    aSimple=0;
    cDoble=0;
    pDoble=0;
    aDoble=0;
    cMatrimonial=0;
    pMatrimonial=0;
    aMatrimonial=0;
}

//12
String valida(){
    if (cboHabitacion.getSelectedIndex()==-1){
       cboHabitacion.requestFocus();
       return "Tipo de Habitacion";
    }
    else if (txtDias.getText().equals("") ||
                        Integer.parseInt(txtDias.getText())<1){
       txtDias.requestFocus();
       return "Numero de dias";
    }
    else if (txtNumero.getText().equals("") ||
                        Integer.parseInt(txtNumero.getText())<1){
       txtDias.requestFocus();
       return "Numero de dias";
    }
    else
       return "";
}
```

En el punto uno se implemente el método llenaHabitacion, que permite llenar el control cboHabitacion con los tipos de habitación que ofrece el hotel.

En el punto dos se implementa el método getHabitacion, que permite capturar el tipo de habitación seleccionada por el usuario.

En el punto tres se implementa el método getDias, que permite capturar el número de días hospedado por el cliente.

En el punto cuatro se implementa el método getNumero, que permite capturar el número de personas que ingresan y que alquilan un determinado tipo de habitación en el hotel.

En el punto cinco se implementa el método determinaCosto, que permite asignar un costo según el tipo de habitación seleccionada por el usuario, para esto el método debe tener un parámetro llamado habitación, el cual contiene el tipo de habitación seleccionada.

En el punto seis se implementa el método calculaMonto, que permite calcular el monto por alquiler, para esto se necesita como parámetros los días de hospedaje y el costo por tipo de habitación seleccionada.

En el punto siete se implementa el método resumen que tiene por misión imprimir los valores hasta ahora calculados dentro del control txtR.

En el punto ocho se implementa el método estadísticas que permita determinar los conteos y acumulaciones de la aplicación, para dicha actividad se le envía como parámetros el número de personas y el monto calculado. Desde aquí se podrá obtener los contadores por tipo de habitación, el número de personas que ingresaron y el acumulado de montos por tipo de habitación.

En el punto nueve se implementa el método imprimir que permite mostrar el resultado de todos los contadores y acumuladores de la aplicación. Todos los resultados serán enviados al control txtE.

En el punto diez se implementa el método limpiar, que tiene por misión limpiar los controles usados por el usuario y devolver el foco al control cboHabitacion.

En el punto once se implementa el método reiniciarGlobales, que tiene por misión inicializar a cero todas las variables que permiten contar y acumular valores en la aplicación. Este método será usado por el botón Limpiar.

En el punto doce se implementa el método valida, que permite controlar y verificar que los datos seleccionados e ingresados por el usuario sean los correctos. Este método identifica donde se ocasionó el error, además de devolver el foco en el control errado. Si todos los valores fueron ingresados correctamente el método devolverá un espacio vacío.

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnProcesarActionPerformed(…) {
    if (valida().equals("")){
        //1
        int habitacion=getHabitacion();
        int dias=getDias();
        int numero=getNumero();

        //2
        double costo=determinaCosto(habitacion);

        //3
        double monto=calculaMonto(dias, costo);
```

```
            //4
            resumen(costo, dias, monto);
            estadisticas(numero, monto);

            //5
            imprimir();
            limpiar();
        }else
            JOptionPane.showMessageDialog(null, "Error en "+valida());
    }
```

Antes de comenzar con las actividades que realiza el evento ActionPerformed del botón btnProcesar se tiene que evaluar si todos los valores ingresados son correctos, por tal motivo se evalúa el valor retornado desde el método valida, si este devuelve un espacio vacío entonces los datos fueron ingresados correctamente y se procederá a capturar los valores.

En el punto uno se captura el contenido de los valores ingresados por el usuario, previamente evaluados por el método valida.

En el punto dos se determina el costo según la habitación seleccionada y es enviada a la variable local costo.

En el punto tres se determina el monto a pagar por el cliente según los días y costo asignado por la habitación seleccionada, esto será enviado a la variable local monto.

En el punto cuatro se invoca a los métodos resumen y estadísticas respectivamente, para poder determinar los costos del alquiler, además de contar y acumular los valores.

En el punto cinco se invoca al método imprimir que tiene por misión imprimir los conteos y acumulaciones realizadas en la aplicación.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
limpiar();
reiniciarGlobales();
txtR.setText("");
txtE.setText("");
}
```

Como verá dentro del botón btnLimpiar se invoca al método limpiar que permitirá limpiar todos los controles usados en la aplicación y además reiniciará el valor inicial de todas las variables globales mediante el método reiniciarVariables.

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "HOTEL",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

Casos Propuestos

1. *Una tienda de venta de juegos didácticos se ha dado cuenta que no tiene un control exacto de sus productos, lo cual le ocasiona mucha pérdida de dinero anualmente, también se debe considerar que dicha tienda cuenta con un departamento de producción en la cual producen los juegos que son enviados a la tienda. La nueva encargada de la administración necesita un inventario rápido de estos productos, para lo cual se implementara una aplicación que permita ingresar la descripción, el tipo de juego (Acuático, Resorte y Giratorios) y el total de productos encontrados de un mismo tipo. Al final la aplicación deberá mostrar:*

   - Total de juegos registrados.
   - Acumulado Total de Productos por tipo de juego.
   - Total de juegos por tipo.

2. *Un hospital de la ciudad ha decidido hacer una colecta a nivel nacional para recaudar fondos y así ayudar con los gastos que ocasione un operación de aquellas personas que llegan sin recursos económicos a dicho hospital, para lo cual se necesita una aplicación que permita seleccionar la ciudad de donde proviene la colaboración (Lima, Cuzco, Arequipa, Tacna y Chiclayo) y el monto donado en dólares. Al final deberá mostrar:*

   - Monto total recaudado.
   - Monto recaudado por cada ciudad.
   - Reporte porcentual de los montos según la ciudad.
   - Número total de donaciones por cada ciudad.

7.3. ESTRUCTURAS REPETITIVAS

Son aquellas que permiten repetir una secuencia de instrucciones o expresiones un número determinado de veces, esta cantidad será implementada por el programador. También son llamados bucles o ciclos que permiten repetir las instrucciones de un determinado número de veces.

Vea algunos casos donde las estructuras de tipo repetición pueden encajar:

- Imprimir un número determinado de veces un mensaje.
- Establecer un límite de ingreso de valor a un usuario; por ejemplo, en un cajero automático solo está permitido errar 3 veces el ingreso de su clave.

Un ejemplo práctico podría ser una carrera de autos. Se sabe que todos los corredores tienen que realizar varios recorridos sobre un mismo circuito.

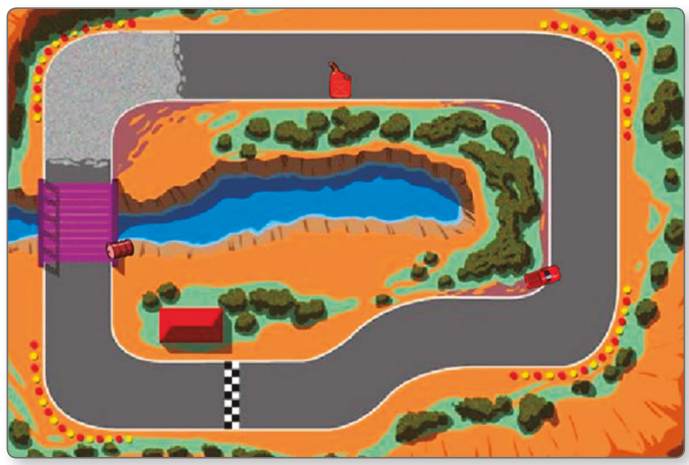


Fig. 7.4

Pero si necesitamos hacer algunas estadísticas sobre la carrera podríamos proponer:

- Número de vueltas realizadas por un determinado auto.
- Total acumulado Km/h por cada vuelta.
- Máxima aceleración realizada por toda la carrera.

Hay que tener en cuenta que las estructuras de repetición están presentes en varias actividades que usted realiza, mejor dicho los bucles están involucrados en nuestra vida diaria.

7.4. Clase DefaultListModel

Un DefaultListModel es un tipo de contenedor para listas que se encarga de administrar los valores de una lista, así como su incorporación, eliminación y actualización.

Métodos Constructores:

DeafultListModel ()	Construye un modelo vacío. Ejm: `moCategoria = new DefaultListModel();`

Métodos:

addElement()	Permite añadir un valor al modelo.
	
	Ejm. Añadirá las categorías de un empleado al modelo moCategoria.
	
	`moCategoria.addElement("Jefe");`
	`moCategoria.addElement("Operario");`
	`moCategoria.addElement("Administrador");`
	`moCategoria.addElement("Tesorero");`

getSize()	Permite devolver el número de elementos contenidos en un modelo. Ejm. Capturar la cantidad de elementos del modelo moCategoria. `int elementos = moCategoria.getSize();`

getElementAt()	Permite obtener un elemento del modelo desde un índice específico.
	
	Ejm. Obtener el primer elemento del modelo moCategoria.
	
	`int indice=0;`
	`String categoria = moCategoria.getElementAt(indice);`
	
	Ejm. Sumar todos los elementos contenidos en el modelo moSubtotal.
	
	`int n=moNumero.size();`
	`double subtotal=0;`
	`for(int i=0;i<n;i++){`
	`subtotal+=Double.parseDouble(moSubtotal.`
	`getElementAt(i).toString());`
	`}`

remove()	Permite eliminar un elemento ubicado en una posición del índice del modelo.
	
	Ejm. Eliminar el primer elemento contenido en el modelo moCategoria.
	
	`int posicion=0;`
	`moCategoria.removeITem(posicion);`

size()	Permite determinar el tamaño total de los elementos contenidos en el modelo. Ejm. Capturar la cantidad de elementos del modelo moCategoria. `int total = moCategoria.size();`

Opciones:

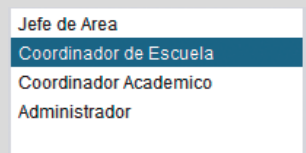
- Declarar un objeto de la clase DefaultListModelo > DefaultListModel moCategoria;
- Crear un objeto de la clase DefaultListModelo > moCategoria = new DefaultListModel();

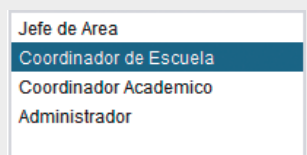
7.5. CLASE JLIST

Clase que permite crear un objeto tipo lista, en la cual se puede almacenar valores que serán visibles y accesibles por el usuario.

Métodos Constructores:

JLIST()	Construye un objeto JList vacío. Ejm: `lstProductos = new JList();`

JLIST (LISTMODEL MODELO)	Construye un objeto JList con un conjunto de valores por medio de un modelo.
	
	Ejm:
	
	`DefaultListModel moCategorias=new DefaultListModel();`
	`moCategorias.addElement("Jefe de Area");`
	`moCategorias.addElement("Coordinador de Escuela");`
	`moCategorias.addElement("Coordinador Academico");`
	`moCategorias.addElement("Administrador");`
	
	`JList lstCategoria = new JList(moCategorias);`
	`lstCategoria.setBounds(10,10,200,100);`
	`add(lstCategoria);`
	
	 

JLIST (OBJECT[] LISTA)	`Object objCategorias[] = new Object[4];`
	`objCategorias[0]="Jefe de Area";`
	`objCategorias[1]="Coordinador de Escuela";`
	`objCategorias[2]="Coordinador Academico";`
	`objCategorias[3]="Administrador";`
	
	`JList lstCategoria=new JList(objCategorias);`
	`lstCategoria.setBounds(10,10,200,100);`
	`add(lstCategoria);`
	
	 

Métodos:

addListSelectionListener()	Permite añadir un receptor de selección al control JList.
clearSelection()	Permite borrar la selección asignada al control JList.
getSelectedIndex()	Permite devolver el índice del primer elemento seleccionado en el control JList.
getSelectedIndex()	Permite devolver una matriz de tipo entero con todos los elementos formados por los índices seleccionados en el control JList.
getSelectedValue()	Permite devolver el primer valor seleccionado desde el control JList, devolverá nulo si no ha seleccionado nada en el control.
getSelectionMode()	Permite definir las listas simples y múltiples en un control JList.
setListData(Object[] lista)	Permite construir un modelo de lista a partir de una matriz de objetos y después aplica el setModel al modelo.
setModel(ListModel modelo)	Permite asignar al modelo que representa el contenido de la lista y limpia la selección.

Opciones:

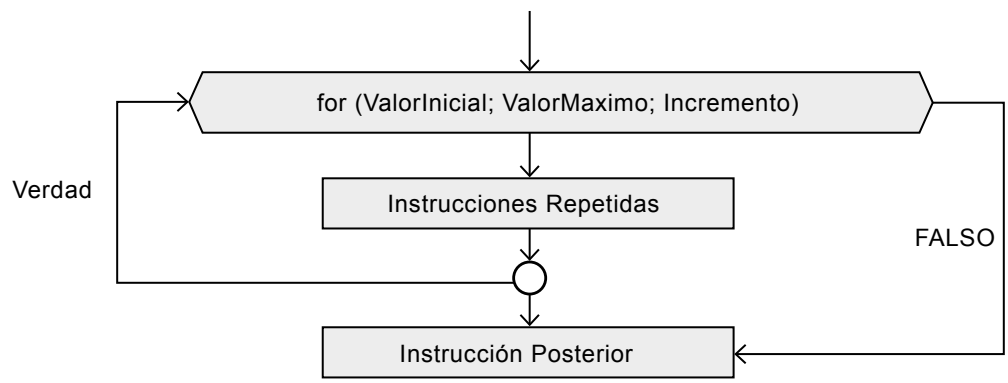
- Declarar un objeto JList > JList lstCategoria;
- Crear un objeto de tipo JList > lstCategoria = new JList();

7.6. Estructura de repetición for

La estructura For ejecuta un número determinado de veces un conjunto de sentencias o instrucciones y controla de forma automática el número de interacciones que esta realiza.

Formato:

```
for(ValorInicial;ValorMaximo;Incremento){
   Intrucciones Repetidas;
}
```



CASO DESARROLLADO 1: VENTA DE CUADERNOS ESCOLARES CON FOR

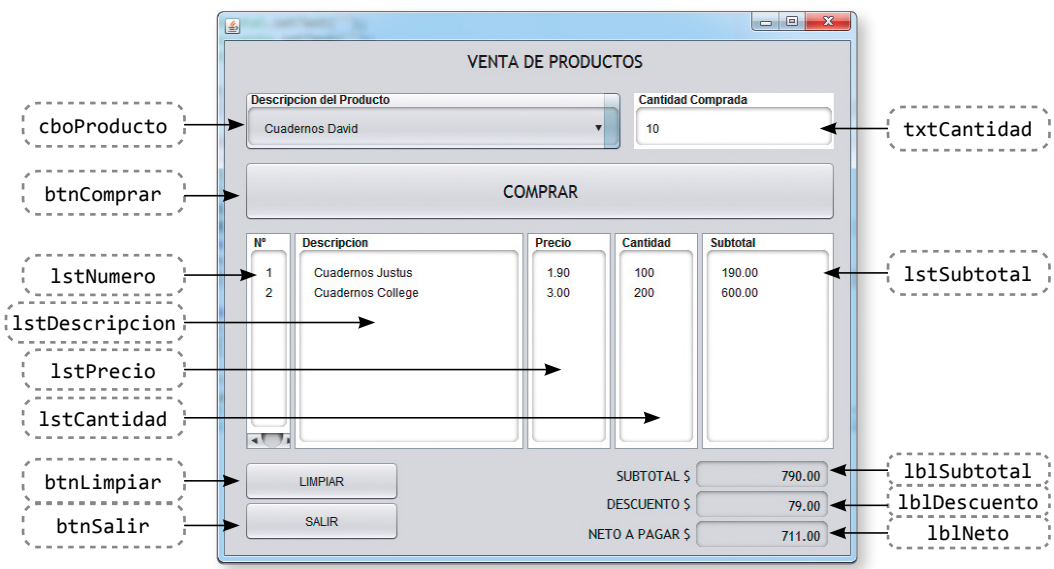
Implemente una aplicación que permita controlar la venta de cuadernos escolares para la campaña de este año, la venta se realiza mediante la selección de un producto y una cantidad determinada. Los productos son como siguen:

DESCRIPCION DEL PRODUCTO	PRECIO $
Cuadernos LayConsa	1.50
Cuadernos Justus	1.90
Cuadernos StanFord	3.50
Cuadernos David	2.50
Cuadernos College	3.00
Cuadernos Alpha	4.50

Tener en cuenta:

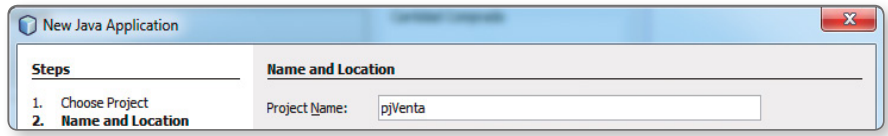
- *Validar todos los valores ingresados mostrando un mensaje de error si fuera necesario.*
- *Debe usar la estructura repetitiva for.*
- *Se deben imprimir todos los valores obtenidos en los controles JList.*
- *Se debe calcular el subtotal acumulado, el descuento (10% del subtotal acumulado) y el neto a pagar por el cliente.*

GUI Propuesto:

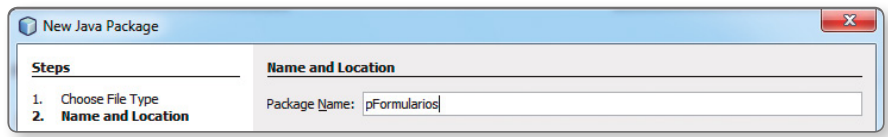


Debe considerar:

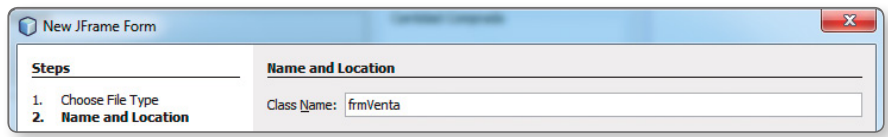
- Crear un nuevo proyecto en NetBeans llamado pjVenta.



- Agregar un nuevo paquete al proyecto pjVenta llamado pFormularios



- Agregar un nuevo Formulario al paquete pFormularios llamada frmVenta.



- Asigne un nombre a cada uno de los controles; como se muestra en la Fig., para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Despues de asignar nombres a los objetos asegurese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 7.5.

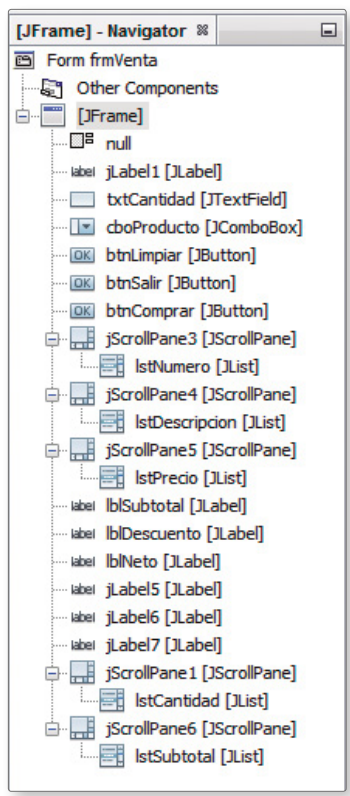


Fig. 7.5

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

cboProducto	**Border**	TitledBorder=Descripcion del Producto
	Text	Dejar vacio
txtCantidad	**Border**	TitledBorder=Cantidad Comprada
	Text	Dejar vacio
btnComprar	**Text**	COMPRAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR
lstNumero	**Border**	TitledBorder=Nº
	Model	Dejar vacio
lstDescripcion	**Border**	TitledBorder=Descripcion
	Model	Dejar vacio
lstPrecio	**Border**	TitledBorder=Precio
	Model	Dejar vacio
lstCantidad	**Border**	TitledBorder=Cantidad
	Model	Dejar vacio
lstSubtotal	**Border**	TitledBorder=Subtotal
	Model	Dejar vacio
lblSubtotal	**Border**	TitledBorder=Dejar vacio
	Text	Dejar vacio
lblDescuento	**Border**	TitledBorder=Dejar vacio
	Text	Dejar vacio
lblNeto	**Border**	TitledBorder=Dejar vacio
	Text	Dejar vacio

- Vea el script de la aplicación:

El siguiente script muestra la declaración de las variables globales de la aplicación:

```
package pFormularios;

import java.text.DecimalFormat;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;

public class frmVenta extends javax.swing.JFrame {
    //Modelos
    DefaultListModel moNumero,moDescripcion,moPrecio;
    DefaultListModel moCantidad,moSubtotal;

    //Formato
    DecimalFormat df;

    //Numero de Producto
    int num;

    public frmVenta() {
        initComponents();

        cargaProductos();
        cargaModelos();

        df=new DecimalFormat("###0.00");
    }
```

Hay que tener en cuenta las librerías mostradas en el código, pues de ellas dependerán muchas clases implementadas, por ejemplo:

- java.text.DecimalFormat: usado para objeto de redondeo de valores numéricos decimales.
- javax.swing.DefaultListModel: usado para los objetos de tipo DefaultListModel que permitirán llenar de elementos los controles JList.
- javax.swing.JOptionPane: usado para los objetos JOptionPane que permitirá mostrar cuadros de mensaje al usuario.

Luego se declara como globales a los modelos que se usarán en la aplicación, entonces tendría lo siguiente:

- moNumero: modelo que representa el número de orden de los productos comprados.
- moDescripcion: modelo que representa la descripción del producto seleccionado por el usuario.
- moPrecio: modelo que representa a los precio de los productos comprados.
- moCantidad: modelo que representa la cantidad de productos comprados.
- moSubtotal: modelo que representa el subtotal de la compra.

También se declara la variable global df de la clase DecimalFormat que permitirá definir el formato para monedas que se usará en la aplicación.

Luego se define la variable global num que tendrá por misión determinar el número de elementos comprados por el cliente.

En el método constructor de la clase frmVenta se invoca a los métodos cargaProductos que tienen por misión llenar el control cboProductos con los productos especificados, en el caso. cargaModelos que permitirá invocar al método para crear los objetos de la clase DefaultListModel y los asigna a sus respectivos controles JList. Hay que tener en cuenta que por cada control JList se debe implementar un modelo. Finalmente, se define el formato de decimales mediante el objeto df; donde # representa a cualquier valor numérico entero entre 0 y 9 y el .00 es para definir la cantidad de decimales de los valores.

El siguiente script muestra todos los métodos implementados en la aplicación.

```
//1.
void cargaModelos(){
   //
   moNumero=new DefaultListModel();
   moDescripcion=new DefaultListModel();
   moPrecio=new DefaultListModel();
   moCantidad=new DefaultListModel();
   moSubtotal=new DefaultListModel();

   //
   lstNumero.setModel(moNumero);
   lstDescripcion.setModel(moDescripcion);
   lstPrecio.setModel(moPrecio);
   lstCantidad.setModel(moCantidad);
   lstSubtotal.setModel(moSubtotal);
}
```

```
//2.
void cargaProductos(){
   cboProducto.addItem("Cuadernos LayConsa");
   cboProducto.addItem("Cuadernos Justus");
   cboProducto.addItem("Cuadernos StanFord");
   cboProducto.addItem("Cuadernos David");
   cboProducto.addItem("Cuadernos College");
   cboProducto.addItem("Cuadernos Alpha");
}

//3.
int getProducto(){
    return cboProducto.getSelectedIndex();
}

//4.
String getDescripcionProducto(){
    return String.valueOf(cboProducto.getSelectedItem());
}

//5.
int getCantidad(){
    return Integer.parseInt(txtCantidad.getText());
}

//6.
double asignaPrecio(int producto){
    switch(producto){
       case 0: return 1.5;
       case 1: return 1.9;
       case 2: return 3.5;
       case 3: return 2.5;
       case 4: return 3.0;
       default: return 4.5;
    }
}

//7.
double calculaSubtotal(double precio,int cantidad){
    return precio*cantidad;
}

//8.
String valida(){
    if (cboProducto.getSelectedIndex()==-1){
       cboProducto.requestFocus();
       return "Descripcion del Producto";
    } else if (txtCantidad.getText().equals("")||
                      Integer.parseInt(txtCantidad.getText())<0){
       txtCantidad.setText("");
       txtCantidad.requestFocus();
       return "Cantidad comprada";
    } else
       return "";
}

//9.
void limpiarControles(){
    cboProducto.setSelectedIndex(-1);
    txtCantidad.setText("");
    cboProducto.requestFocus();
}
```

```
//10.
void imprimirListas(double precio,double subtotal){
     num++;
     moNumero.addElement(num);
     moDescripcion.addElement(getDescripcionProducto());
     moPrecio.addElement(df.format(precio));
     moCantidad.addElement(getCantidad());
     moSubtotal.addElement(df.format(subtotal));
     calculaTotales();
}

//11.
void calculaTotales(){
     int n=moNumero.size();
     double subtotal=0;
     for(int i=0;i<n;i++){
  subtotal+=Double.parseDouble(moSubtotal.getElementAt(i).toString());
     }

     double descuento=subtotal*0.1;
     double neto=subtotal-descuento;

     lblSubtotal.setText(df.format(subtotal));
     lblDescuento.setText(df.format(descuento));
     lblNeto.setText(df.format(neto));
}
```

En el punto uno se implementa el método cargaModelos, que permite crear los objetos de la clase DefaultListModel y asignar a cada JList creado un modelo.

En el punto dos se implementa el método cargaProductos, que tiene por misión llenar el control cboProducto con los productos proporcionados por el caso.

En el punto tres se implementa el método getProducto, que tiene por misión obtener el producto seleccionado por el usuario mediante su número de índice.

En el punto cuatro se implementa el método getDescripcionProducto, que tiene por misión devolver el valor del control cboProducto seleccionado por el usuario, a diferencia del método getProducto que devuelve el índice del producto seleccionado, getDescripcionProducto devuelve que producto fue seleccionado; esto nos servirá para enviarlo al modelo perteneciente a la descripción de los productos seleccionados.

En el punto cinco se implementa el método getCantidad, que permite obtener la cantidad ingresada por el usuario debido a la compra de un determinado producto.

En el punto seis se implementa el método asignaPrecio, que tiene por misión asignar con un precio al producto seleccionado por el usuario, este ultimo será enviado al método por medio de un parámetro, al final devolverá un precio de producto.

En el punto siete se implementa el método calculaSubtotal, que tiene por misión devolver el producto entre el precio de un determinado producto y la cantidad seleccionada por el usuario. Hay que tener en cuenta que estos dos últimos deben ser enviados como parámetros hacia el método.

En el punto ocho se implementa el método valida que tiene por misión verificar si los valores ingresados o seleccionados en la aplicación son los correctos, para esto el método devolverá un texto indicando en que control ocurrió un error, tenga en cuenta que si el método devuelve un texto vacío será indicador de que todos los valores ingresados son correctos.

En el punto nueve se implementa el método limpiaControles que tiene por misión limpiar los valores ingresados por el usuario y devolver el foco al control cboProducto.

En el punto diez se implementa el método imprimirListas que se encargará de enviar los elementos que serán visibles en todos los controles JList; hay que tener en cuenta que dichos valores deben ser enviados a los modelos definidos. Al final se invoca al método calculaTotales que permitirá mostrar el acumulado del subtotal, el descuento y el neto a pagar por el cliente. La variable global num permite aumentar en uno por cada compra realizada por el cliente y luego es impreso en el modelo respectivo.

En el punto once se implementa el método calculaTotales que tiene por misión recorrer todos los elementos contenidos en el modelo moSubtotal y así acumularlos para obtener el acumulado de los subtotales. Finalmente, estos valores son calculados y enviados a los controles lblSubtotal, lblDescuento y lblNeto.

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnComprarActionPerformed(java.awt.event.ActionEvent evt) {
       if (valida().equals(""))
        {
            int producto = getProducto();
            int cantidad = getCantidad();

            double precio = asignaPrecio(producto);
            double subtotal = calculaSubtotal(precio, cantidad);

            imprimirListas(precio, subtotal);

            UIResource posicion = new UIResource();
            posicion.setHorizontalAlignment(SwingConstants.RIGHT);
            lstSubtotal.setCellRenderer(posicion);

            limpiarControles();
        } else
            JOptionPane.showMessageDialog(null,"Error en "+valida());
}
```

Lo primero es invocar al método valida para determinar si todos los valores ingresados por el usuario son los correctos; caso contrario enviar un mensaje con el nombre del objeto errado. En caso no genere error el método valida se procede a capturar los valores seleccionados e ingresados por el usuario, es decir, la variable producto obtendrá el índice del producto seleccionado y cantidad almacenará la cantidad comprada.

Luego se declara la variable precio que obtendrá el precio del producto mediante el método asignaPrecio. La variable subtotal almacenara el valor del subtotal obtenido mediante el método calculaSubtotal.

Luego se imprimen los valores obtenidos en las listas respectivas mediante el método imprimirListas. Cuando se imprimen los valores en la lista lstSubtotal hay que considerar que se debe imprimir los

subtotales en el lado derecho del control, para esto se implementó la clase UIResource que permitirá definir la posición del elemento dentro del control lstSubtotal; esto se realiza mediante el método setHorizontalAlignment. Finalmente se invoca al método limpiaControles para realizar una nueva compra.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
limpiarControles();
moNumero.removeAllElements();
moDescripcion.removeAllElements();
moCantidad.removeAllElements();
moPrecio.removeAllElements();
moSubtotal.removeAllElements();

lblSubtotal.setText("");
lblDescuento.setText("");
lblNeto.setText("");

num=0;
}
```

Como verá dentro del botón btnLimpiar se invoca al método limpiaControles que permitirá limpiar todos los controles usados en la aplicación y además se debe remover todos los elementos contenidos dentro los JList mediante sus modelos. Finalmente, se debe reinicializar el valor num ya que se generaría una nueva compra de productos.

En el siguiente script se muestra como salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

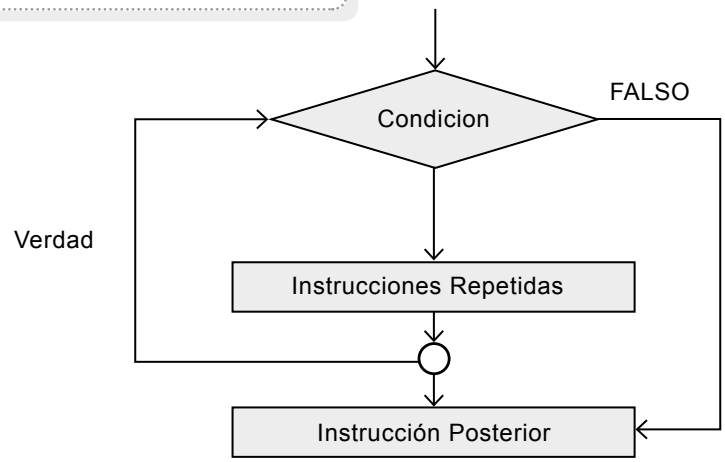
```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Sistema de Ventas",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

7.7. ESTRUCTURA DE REPETICIÓN WHILE

La estructura While ejecuta un número determinado de veces un conjunto de sentencias o instrucciones y controla el número de interacciones que esta realiza mediante una condición lógica.

Formato:

```
while (condicion){
   Intrucciones Repetidas;
   Valor Incremento;
}
```



CASO DESARROLLADO 2: VENTA DE CUADERNOS ESCOLARES CON WHILE

Implemente una aplicación que permita controlar la venta de cuadernos escolares para la campaña de este año, la venta se realiza mediante la selección de un producto y una cantidad determinada. Los productos son como siguen:

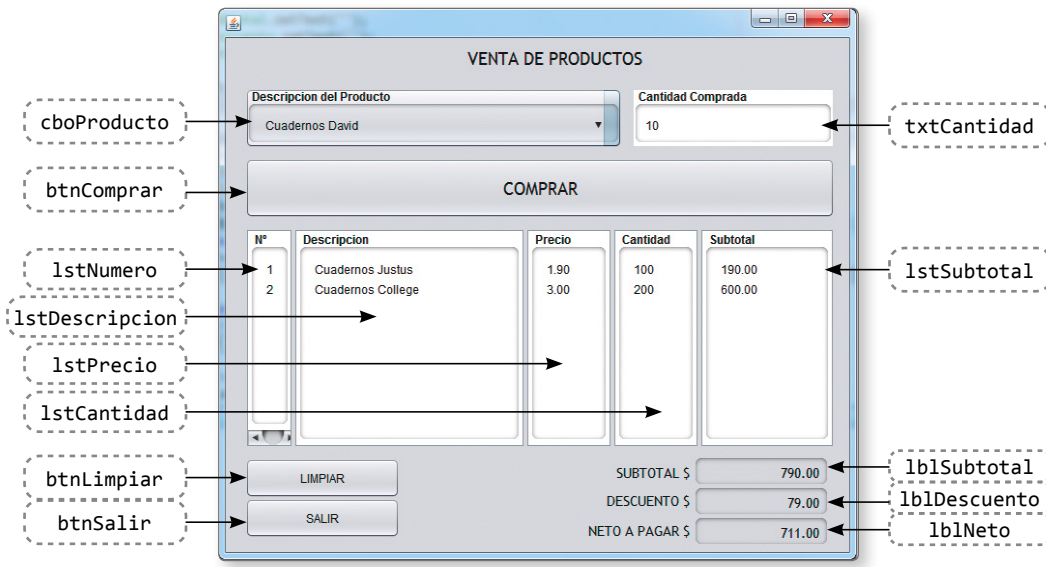
DESCRIPCION DEL PRODUCTO	PRECIO $
Cuadernos LayConsa	1.50
Cuadernos Justus	1.90
Cuadernos StanFord	3.50
Cuadernos David	2.50
Cuadernos College	3.00
Cuadernos Alpha	4.50

Tener en cuenta:

- *Validar todos los valores ingresados mostrando un mensaje de error si fuera necesario.*

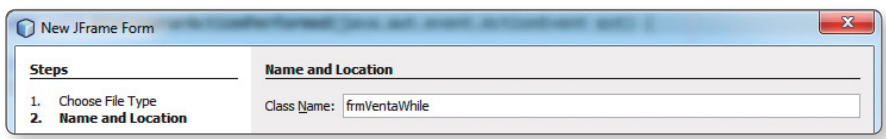
- *Debe usar la estructura repetitiva while.*
- *Se deben imprimir todos los valores obtenidos en controls JList*
- *Se debe calcular el subtotal acumulado, el descuento (10% del subtotal acumulado) y el neto a pagar por el cliente.*

GUI Propuesto:



Debe considerar:

- Agregar un nuevo Formulario al paquete pFormularios llamado frmVentaWhile.



- Asigne un nombre a cada uno de los controles; como se muestra en la Fig., para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Despues de asignar nombres a los objetos asegurese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 7.6.

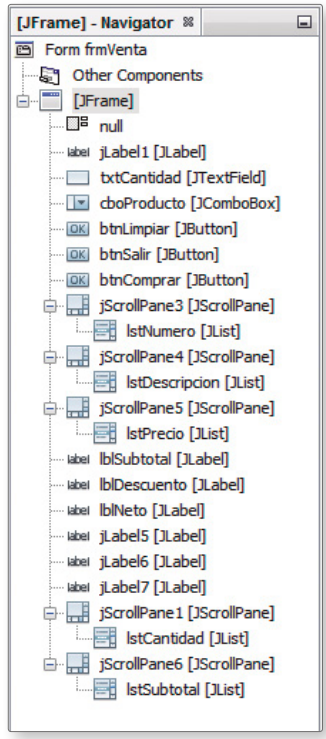


Fig. 7.6

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

cboProducto	**Border** **Text**	TitledBorder=Descripcion del Producto Dejar vacio
txtCantidad	**Border** **Text**	TitledBorder=Cantidad Comprada Dejar vacio
btnComprar	**Text**	COMPRAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR
lstNumero	**Border** **Model**	TitledBorder=N° Dejar vacio
lstDescripcion	**Border** **Model**	TitledBorder=Descripcion Dejar vacio
lstPrecio	**Border** **Model**	TitledBorder=Precio Dejar vacio
lstCantidad	**Border** **Model**	TitledBorder=Cantidad Dejar vacio
lstSubtotal	**Border** **Model**	TitledBorder=Subtotal Dejar vacio
lblSubtotal	**Border** **Text**	TitledBorder=Dejar vacio Dejar vacio
lblDescuento	**Border** **Text**	TitledBorder=Dejar vacio Dejar vacio
lblNeto	**Border** **Text**	TitledBorder=Dejar vacio Dejar vacio

- Vea el script de la aplicación:

  El siguiente script muestra la declaración de las variables globales de la aplicación:

```
package pFormularios;

import java.text.DecimalFormat;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;

public class frmVenta extends javax.swing.JFrame {
    //Modelos
    DefaultListModel moNumero,moDescripcion,moPrecio;
    DefaultListModel moCantidad,moSubtotal;

    //Formato
    DecimalFormat df;

    //Numero de Producto
    int num;

    public frmVenta() {
        initComponents();

        cargaProductos();
        cargaModelos();

        df=new DecimalFormat("###0.00");
    }
```

Los puntos ya han sido definidos en el caso desarrollado 1.

El siguiente script es de todos los métodos implementados en la aplicación.

```
//1.
void cargaModelos(){
   //
   moNumero=new DefaultListModel();
   moDescripcion=new DefaultListModel();
   moPrecio=new DefaultListModel();
   moCantidad=new DefaultListModel();
   moSubtotal=new DefaultListModel();

   //
   lstNumero.setModel(moNumero);
   lstDescripcion.setModel(moDescripcion);
   lstPrecio.setModel(moPrecio);
   lstCantidad.setModel(moCantidad);
   lstSubtotal.setModel(moSubtotal);
}

//2.
void cargaProductos(){
   cboProducto.addItem("Cuadernos LayConsa");
   cboProducto.addItem("Cuadernos Justus");
   cboProducto.addItem("Cuadernos StanFord");
   cboProducto.addItem("Cuadernos David");
   cboProducto.addItem("Cuadernos College");
   cboProducto.addItem("Cuadernos Alpha");
}
```

```
//3.
int getProducto(){
    return cboProducto.getSelectedIndex();
}

//4.
String getDescripcionProducto(){
    return String.valueOf(cboProducto.getSelectedItem());
}

//5.
int getCantidad(){
    return Integer.parseInt(txtCantidad.getText());
}

//6.
double asignaPrecio(int producto){
    switch(producto){
       case 0: return 1.5;
       case 1: return 1.9;
       case 2: return 3.5;
       case 3: return 2.5;
       case 4: return 3.0;
       default: return 4.5;
    }
}

//7.
double calculaSubtotal(double precio,int cantidad){
    return precio*cantidad;
}

//8.
String valida(){
    if (cboProducto.getSelectedIndex()==-1){
       cboProducto.requestFocus();
       return "Descripcion del Producto";
    } else if (txtCantidad.getText().equals("")||
                     Integer.parseInt(txtCantidad.getText())<0){
       txtCantidad.setText("");
       txtCantidad.requestFocus();
       return "Cantidad comprada";
    } else
       return "";
}

//9.
void limpiarControles(){
    cboProducto.setSelectedIndex(-1);
    txtCantidad.setText("");
    cboProducto.requestFocus();
}

//10.
void imprimirListas(double precio,double subtotal){
     num++;
     moNumero.addElement(num);
     moDescripcion.addElement(getDescripcionProducto());
     moPrecio.addElement(df.format(precio));
     moCantidad.addElement(getCantidad());
     moSubtotal.addElement(df.format(subtotal));
     calculaTotales();
}
```

```
//11.
void calculaTotales(){
     int n=moNumero.size();
     double subtotal=0;

     int i=0;
     while(i<n){
          subtotal+=Double.parseDouble(moSubtotal.
                                        getElementAt(i).toString());
          i++;
     }

     double descuento=subtotal*0.1;
     double neto=subtotal-descuento;

     lblSubtotal.setText(df.format(subtotal));
     lblDescuento.setText(df.format(descuento));
     lblNeto.setText(df.format(neto));
}
```

Los puntos del 1 al 10 han sido definidos en el caso desarrollado 1.

En el punto once se implementa el método calculaTotales que tiene por misión recorrer todos los elementos contenidos en el modelo moSubtotal mediante la instrucción while a diferencia de la instrucción for; while tiene que definir un punto de inicio fuera del mismo, es decir, int i=0; luego se condiciona dicha variable hasta que llega al total elementos encontrados. Es obligatorio que en todo while se especifica el valor incremento (i++).

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnCompraActionPerformed(java.awt.event.ActionEvent evt) {
       if (valida().equals(""))
        {
            int producto = getProducto();
            int cantidad = getCantidad();

            double precio = asignaPrecio(producto);
            double subtotal = calculaSubtotal(precio, cantidad);

            imprimirListas(precio, subtotal);

            UIResource posicion = new UIResource();
            posicion.setHorizontalAlignment(SwingConstants.RIGHT);
            lstSubtotal.setCellRenderer(posicion);

            limpiarControles();
        } else
            JOptionPane.showMessageDialog(null,"Error en "+valida());
}
```

Los puntos ya han sido definidos en el caso desarrollado 1.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
limpiarControles();
moNumero.removeAllElements();
moDescripcion.removeAllElements();
moCantidad.removeAllElements();
moPrecio.removeAllElements();
moSubtotal.removeAllElements();

lblSubtotal.setText("");
lblDescuento.setText("");
lblNeto.setText("");

num=0;
}
```

Los puntos ya han sido definidos en el caso desarrollado 1.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Sistema de Ventas",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

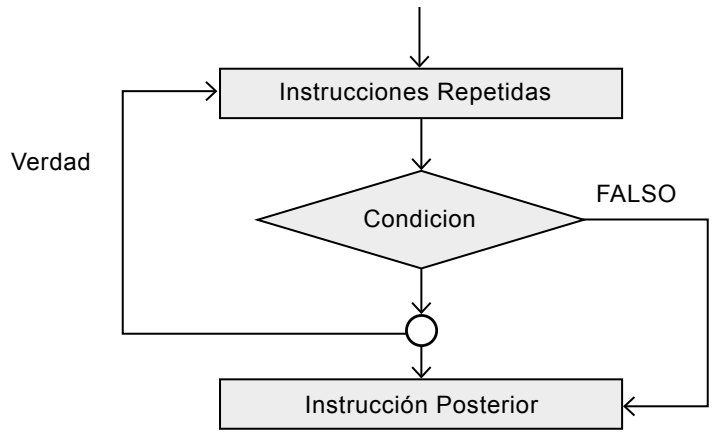
Los puntos ya han sido definidos en el caso desarrollado 1.

7.8. Estructura de repetición Do... While

La estructura Do… While ejecuta un número determinado de veces un conjunto de sentencias o instrucciones y controla el número de iteracciones que esta realiza mediante una condición lógica.

Formato:

```
do {
   Intrucciones Repetidas;
   Valor Incremento;
} while (condicion);
```



Caso desarrollado 3: Venta de cuadernos escolares con Do While

Implemente una aplicación que permita controlar la venta de cuadernos escolares para la campaña de este año, la venta se realiza mediante la selección de un producto y una cantidad determinada. Los productos son como siguen:

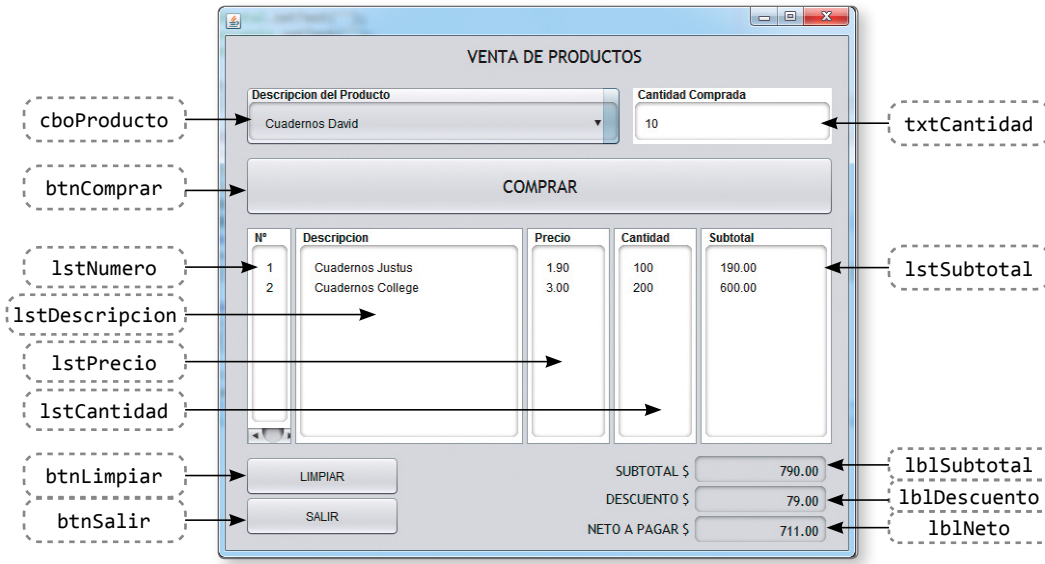
Descripcion del Producto	Precio $
Cuadernos LayConsa	1.50
Cuadernos Justus	1.90
Cuadernos StanFord	3.50
Cuadernos David	2.50
Cuadernos College	3.00
Cuadernos Alpha	4.50

Tener en cuenta:

- *Validar todos los valores ingresados mostrando un mensaje de error si fuera necesario.*

- *Debe usar la estructura repetitiva do…while.*
- *Se deben imprimir todos los valores obtenidos en controls JList.*
- *Se debe calcular el subtotal acumulado, el descuento (10% del subtotal acumulado) y el neto a pagar por el cliente.*

GUI Propuesto:



Debe considerar:

- Agregar un nuevo Formulario al paquete pFormularios llamado frmVentaDoWhile.

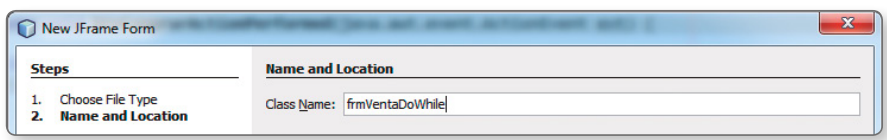


Fig. 7.7

- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 7.7, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**…
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos, para eso sualice el panel Navigator; debe mostrarse como la Fig. 7.8

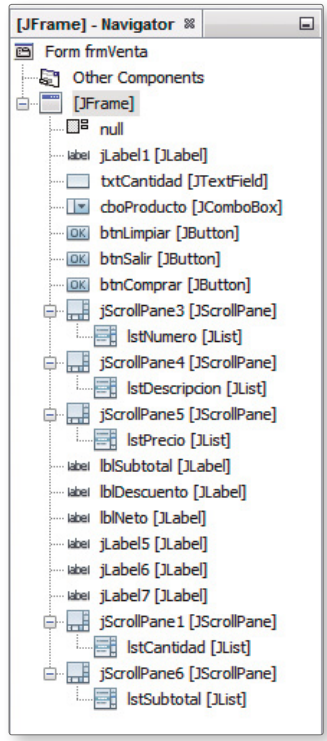


Fig. 7.8

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

cboProducto	**Border**	TitledBorder=Descripcion del Producto
	Text	Dejar vacio
txtCantidad	**Border**	TitledBorder=Cantidad Comprada
	Text	Dejar vacio
btnComprar	**Text**	COMPRAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR
lstNumero	**Border**	TitledBorder=Nº
	Model	Dejar vacio
lstDescripcion	**Border**	TitledBorder=Descripcion
	Model	Dejar vacio
lstPrecio	**Border**	TitledBorder=Precio
	Model	Dejar vacio
lstCantidad	**Border**	TitledBorder=Cantidad
	Model	Dejar vacio
lstSubtotal	**Border**	TitledBorder=Subtotal
	Model	Dejar vacio
lblSubtotal	**Border**	TitledBorder=Dejar vacio
	Text	Dejar vacio
lblDescuento	**Border**	TitledBorder=Dejar vacio
	Text	Dejar vacio
lblNeto	**Border**	TitledBorder=Dejar vacio
	Text	Dejar vacio

- Vea el script de la aplicación:

  El siguiente script muestra la declaración de las variables globales de la aplicación:

```
package pFormularios;

import java.text.DecimalFormat;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;

public class frmVenta extends javax.swing.JFrame {
    //Modelos
    DefaultListModel moNumero,moDescripcion,moPrecio;
    DefaultListModel moCantidad,moSubtotal;

    //Formato
    DecimalFormat df;

    //Numero de Producto
    int num;

    public frmVenta() {
        initComponents();

        cargaProductos();
        cargaModelos();

        df=new DecimalFormat("###0.00");
    }
```

Los puntos ya han sido definidos en el caso desarrollado 1.

El siguiente es el script de todos los métodos implementados en la aplicación.

```
//1.
void cargaModelos(){
   //
   moNumero=new DefaultListModel();
   moDescripcion=new DefaultListModel();
   moPrecio=new DefaultListModel();
   moCantidad=new DefaultListModel();
   moSubtotal=new DefaultListModel();

   //
   lstNumero.setModel(moNumero);
   lstDescripcion.setModel(moDescripcion);
   lstPrecio.setModel(moPrecio);
   lstCantidad.setModel(moCantidad);
   lstSubtotal.setModel(moSubtotal);
}

//2.
void cargaProductos(){
   cboProducto.addItem("Cuadernos LayConsa");
   cboProducto.addItem("Cuadernos Justus");
   cboProducto.addItem("Cuadernos StanFord");
   cboProducto.addItem("Cuadernos David");
   cboProducto.addItem("Cuadernos College");
   cboProducto.addItem("Cuadernos Alpha");
}
```

```
//3.
int getProducto(){
    return cboProducto.getSelectedIndex();
}

//4.
String getDescripcionProducto(){
    return String.valueOf(cboProducto.getSelectedItem());
}

//5.
int getCantidad(){
    return Integer.parseInt(txtCantidad.getText());
}

//6.
double asignaPrecio(int producto){
    switch(producto){
       case 0: return 1.5;
       case 1: return 1.9;
       case 2: return 3.5;
       case 3: return 2.5;
       case 4: return 3.0;
       default: return 4.5;
    }
}

//7.
double calculaSubtotal(double precio,int cantidad){
    return precio*cantidad;
}

//8.
String valida(){
    if (cboProducto.getSelectedIndex()==-1){
       cboProducto.requestFocus();
       return "Descripcion del Producto";
    } else if (txtCantidad.getText().equals("")||
                     Integer.parseInt(txtCantidad.getText())<0){
       txtCantidad.setText("");
       txtCantidad.requestFocus();
       return "Cantidad comprada";
    } else
       return "";
}

//9.
void limpiarControles(){
    cboProducto.setSelectedIndex(-1);
    txtCantidad.setText("");
    cboProducto.requestFocus();
}

//10.
void imprimirListas(double precio,double subtotal){
     num++;
     moNumero.addElement(num);
     moDescripcion.addElement(getDescripcionProducto());
     moPrecio.addElement(df.format(precio));
     moCantidad.addElement(getCantidad());
     moSubtotal.addElement(df.format(subtotal));
     calculaTotales();
}
```

```
//11.
void calculaTotales(){
     int n=moNumero.size();
     double subtotal=0;

     int i=0;
        do {
           subtotal+=Double.parseDouble(moSubtotal.
                                        getElementAt(i).toString());
           i++;
        }while(i<n);

     double descuento=subtotal*0.1;
     double neto=subtotal-descuento;

     lblSubtotal.setText(df.format(subtotal));
     lblDescuento.setText(df.format(descuento));
     lblNeto.setText(df.format(neto));
}
```

Los puntos del 1 al 10 han sido definidos en el caso desarrollado 1.

En el punto once se implementa el método calculaTotales, que tiene por misión recorrer todos los elementos contenidos en el modelo moSubtotal mediante la instrucción Do While a diferencia de la instrucción for y while, la instrucción Do ya recorre una vuelta antes de condicionar el valor, lo demás es similar a lo trabajado dentro de la instrucción while.

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnComprarActionPerformed(java.awt.event.ActionEvent evt) {
       if (valida().equals(""))
        {
            int producto = getProducto();
            int cantidad = getCantidad();

            double precio = asignaPrecio(producto);
            double subtotal = calculaSubtotal(precio, cantidad);

            imprimirListas(precio, subtotal);

            UIResource posicion = new UIResource();
            posicion.setHorizontalAlignment(SwingConstants.RIGHT);
            lstSubtotal.setCellRenderer(posicion);

            limpiarControles();
        } else
            JOptionPane.showMessageDialog(null,"Error en "+valida());
}
```

Los puntos ya han sido definidos en el caso desarrollado 1.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
limpiarControles();
moNumero.removeAllElements();
moDescripcion.removeAllElements();
moCantidad.removeAllElements();
moPrecio.removeAllElements();
moSubtotal.removeAllElements();

lblSubtotal.setText("");
lblDescuento.setText("");
lblNeto.setText("");

num=0;
}
```

Los puntos ya han sido definidos en el caso desarrollado 1.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Sistema de Ventas",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

Los puntos ya han sido definidos en el caso desarrollado 1.

CAP.

8

Programación orientada a objetos

CAPACIDAD:

- Reconoce los conceptos básicos de programación orientada a objetos.
- Implementa aplicaciones con atributos privados y públicos
- Implementa aplicaciones con método constructor.

CONTENIDO:

8.1. INTRODUCCIÓN

La programación orientada a objetos surge como un intento de dominar la complejidad que tiene la construcción de un software, con esto se entiende que ahora se desarrollarán aplicaciones grandes o pequeñas duraderas en el tiempo y factibles de modificar. Tradicionalmente, la forma de enfrentarse a la complejidad del software era usando la programación estructurada, la cual hacía que una aplicación compleja se descomponga en porciones pequeñas de códigos simples y fáciles de codificar. Entonces, podemos decir que la programación orientada a objetos es otra forma de descomponer el código por medio de la descomposición de objetos.

La programación Orientada a Objetos (POO) en Java es una forma especial de implementar aplicaciones, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación. Con la POO se tiene que aprender a pensar las cosas de una manera diferente, para implementar nuestras aplicaciones en términos de objetos, propiedades y métodos.

En la actualidad, la programación orientada a objetos es algo común entre los programadores ya que los lenguajes de programación promueven este estilo. Durante varios años, los desarrolladores se han dedicado a construir aplicaciones muy parecidas que permitían resolver problemas haciendo que el programador sea una persona indispensable en la organización, ya que solo el programador conoce el código que implementó; la POO se creó con la idea principal de no depender de una persona particular, de manera que otras personas puedan utilizarlas y adelantar su trabajo, así se pudo conseguir que el código se pueda reutilizar.

Con todo lo manifestado de la POO no piense que es difícil programar en programación orientada a objetos; al contrario se volverá algo familiar en muy poco tiempo para usted.

Ahora le toca pensar en términos de objetos para poder entender mejor los términos usados hasta el momento. Por ejemplo: un teléfono móvil:

CONCEPTO:	El teléfono móvil es un dispositivo inalámbrico electrónico para acceder y utilizar los servicios de la red de telefonía celular o móvil. Se denomina celular en la mayoría de países latinoamericanos debido a que el servicio funciona mediante una red de celdas, donde cada antena repetidora de señal es una célula.
CARACTERÍSTICAS:	Sistema Operativo: Android 4.0.4 Banda: 3G – 850MHz Memoria RAM: 512MB Pantalla capacitiva 5 pulgadas Resolución 480x800 pixeles Doble cámara 1.3 MP.
FUNCIONALIDAD:	Llamada a otros teléfonos Captura fotos Captura Videos Ejecuta juegos 3D Reproduce música y videos Accede a redes sociales

En programación orientada a objetos el teléfono móvil sería un objeto de la clase teléfono, las propiedades serían las características y los métodos serían las funcionalidades que tiene el dispositivo móvil.

8.2. CONCEPTOS EN PROGRAMACIÓN ORIENTADA A OBJETOS

8.2.1. Paquetes

Desde ahora, cuando implemente una aplicación en Java debe tener especial cuidado con la organización de los archivos, conforme se avance en el tema notará que la administración de los archivos de una aplicación podría ser un poco descontrolada, es decir, no puede colocar todas las clases dentro de un mismo lugar.

Un mecanismo para administrar las clases, interfaces, formularios, conexión a base de datos o imágenes es implementar paquetes. Los paquetes se construyen con un propósito común dentro de la aplicación.

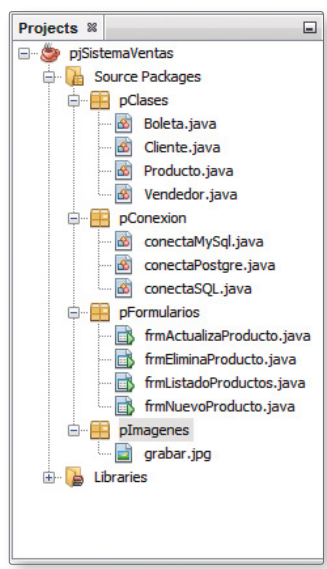


Fig. 8.1

En la Fig. 8.1 observamos un ejemplo básico de composición de paquetes para un proyecto, la organización del proyecto resulta evidente cuando se organiza en paquetes, donde pClases almacenará solo las clases que compone la aplicación, pConexion almacenará las clases que permiten conectarse a una base de datos especifica, pFormularios almacenará las clases de tipo Frame de la aplicación y pImagenes almacenará las imágenes usadas en la aplicación.

Vea un ejemplo compuesto dentro del paquete pClases:

Nombre del Archivo: Vendedor.java

```
package pClases;
import javax.swing.JOptionPane;

public class Vendedor{
   private int codigo;
   private String nombres;
   private String direccion;
   private int categoria;

   public Vendedor(int codigo, String nombres,
                   String direccion,int categoria){
    this.codigo=codigo;
    this.nombres=nombres;
    this.direccion=direccion;
    this.categoria=categoria;
  }
}
```

Nombre del Archivo: Cliente.java

```
package pClases;
import javax.swing.JOptionPane;
import java.sql.*;

public class Cliente{
   private int codigo;
   private String nombres;
   private String telefono;
   private String email;

   public Cliente(int codigo, String nombres,
                   String telefono,String email){
    this.codigo=codigo;
    this.nombres=nombres;
    this.telefono=telefono;
    this.email=email;
  }
}
```

La clase vendedor y cliente pertenecen a un mismo paquete debido a la instrucción package pClases; es decir, todas las clases bajo el entorno de la instrucción serán clases que pertenecerán al paquete del mismo nombre.

Java importa paquetes a las aplicaciones de forma nativa como por ejemplo java.util, java.lang, java.io, java.net, javax.swing, etc. Verá también que un archivo Java tiene características similares como por ejemplo:

- Solo puede haber una sentencia de referencia a un paquete, en caso no se defina un paquete; Java crea uno en forma estándar. Esto quiere decir que por archivo .java solo puede haber una instrucción package.

```
package pClases;
```

- Se puede importar tantos paquetes necesite la aplicación, aquí no hay restricciones; lo que hay que tener en cuenta es importar lo necesario para la aplicación.

```
import javax.swing.JOptionPane;
import java.sql.*;
```

- Solo está permitida la declaración de una clase pública.

```
package pClases;

public class Cliente{

}
```

CREACIÓN DE UN PAQUETE

Genéricamente tenemos el siguiente formato:

```
package nombrePaquete;
```

Con JCreator: suponga que tiene un espacio de trabajo llamado SistemaVentas y un proyecto llamado pjVentas en JCreator, usted crea 3 paquetes al proyecto llamados: pClases, pImagenes, pArreglos.

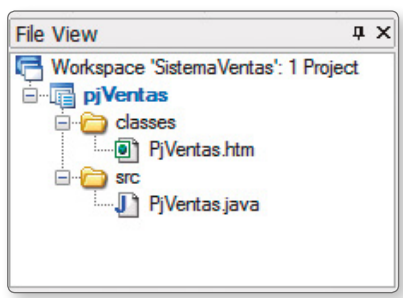


Fig. 8.2

Pasos para la creación de paquetes en JCreator:

PASO 1:

Clic derecho sobre la carpeta
src > seleccione Add > New Folder...

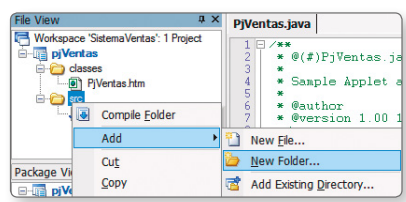


Fig. 8.3

PASO 2:

Luego se deberá colocar el nombre del paquete:

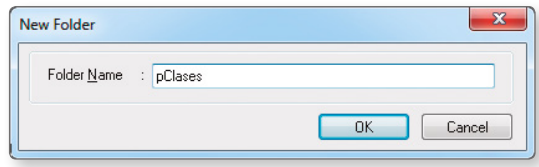


Fig. 8.4

PASO 3:

Finalmente, el panel File View se muestra de la siguiente forma:

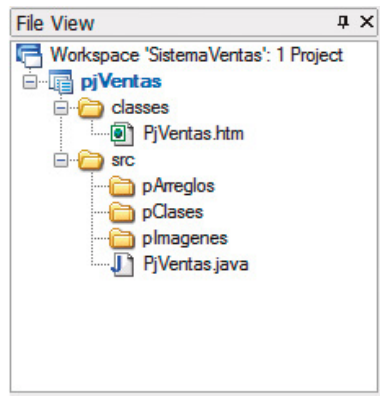


Fig. 8.5

Con NetBeans: suponga que cuenta con un proyecto en NetBeans llamado pjSistemaVentas y necesita crear los paquetes pArreglos, pClases y pImagenes.

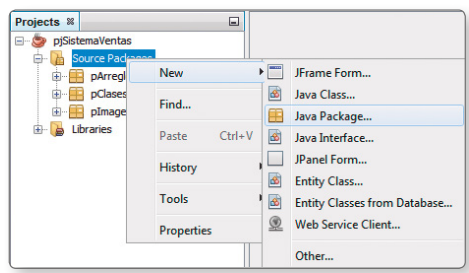


Fig. 8.6

Pasos para la creación de paquetes en NetBeans:

PASO 1:

Clic derecho sobre la carpeta
Source Packages > seleccione New > Java Package...

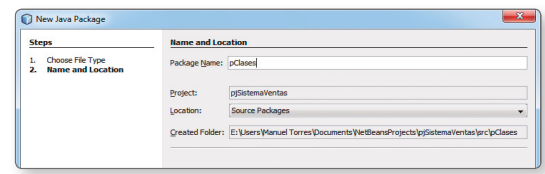


Fig. 8.7

Paso 2:

Luego colocar el nombre del paquete.

Paso 3:

Finalmente, el panel Projects se muestra de la siguiente forma:

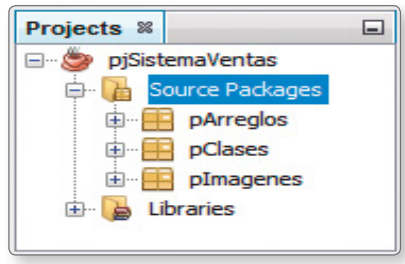


Fig. 8.8

Importar un paquete

La importación conlleva a la apertura de una o más clases contenidas dentro del paquete, es decir, se podrá usar las clases que fueron implementadas con visibilidad pública.

Formato:

```
import nombrePaquete.clase;
```

Todos los IDE's tienen el mismo formato de importación, en algunos casos se realiza en forma automática; como sucede con NetBeans que al momento de crear una clase se especifica a qué paquete se destinará la clase y la instrucción import aparece de forma automática dentro del código fuente.

Vea el siguiente caso. Si tiene el paquete pClases que contiene las clases Empleado, Producto y Boleta, la importación del paquete se puede realizar de la siguiente manera:

- Importar directamente una clase específica

```
import pClases.Empleado;
import pClases.Producto;
import pClases.Boleta;
```

- Importar todas las clases públicas del paquete pClases

```
import pClases.*;
```

Como ve el operador * hace referencia a todas las clases públicas contenidas dentro del mismo paquete.

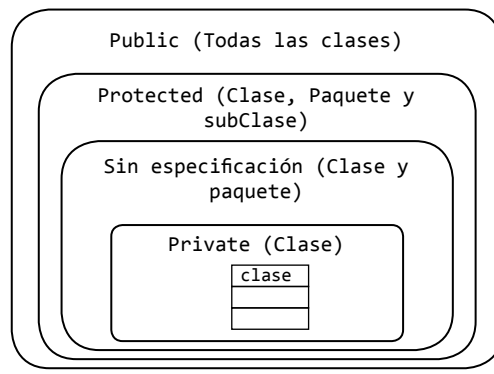
Alcance de los elementos contenidos en un paquete

Si tiene en cuenta que la construcción de un paquete conlleva una organización de clases, entonces también debe considerar el alcance o visibilidad que tienen frente a los demás paquetes. Por ejemplo,

si se necesita hacer referencia de un paquete a otro, debe considerar que dicho elemento sea privado, público, protegido o no, y por tanto, tiene un alcance definido. Vea el siguiente cuadro que muestra la situación que puede tener un elemento dentro de un paquete:

SITUACIÓN DEL ELEMENTO CONTENIDO EN EL PAQUETE	VISIBILIDAD PRIVADA (PRIVATE)	VISIBILIDAD PÚBLICA (PUBLIC)	VISIBILIDAD PROTEGIDA (PROTECTED)	SIN VISIBILIDAD
Dentro de la misma clase.	SI	SI	SI	SI
Desde una subclase.	NO	SI	SI	NO
Desde una clase implementada dentro del mismo paquete.	NO	SI	SI	SI
Desde el exterior de la misma clase.	NO	SI	NO	NO

En la siguiente gráfica se muestra el alcance que pueden tener los miembros de una clase, usted como programador debe tener en cuenta que tanto los atributos como los métodos trabajan con un fin en común y que la definición del alcance es muy importante para la aplicación.



8.3. CLASES EN JAVA

Una clase es elemento básico de la programación orientada a objetos, representa a una plantilla desde la que se pueden crear objetos.

Técnicamente una clase es una declaración de objetos, también se podría definir como una abstracción de objetos; esto quiere decir que la definición de un objeto es la clase. Cuando programamos sobre un objeto y se le define características y funcionalidades, en realidad lo que estamos haciendo es programar en una clase.

8.3.1. Partes de una clase

En el lenguaje de modelamiento unificado a la clase es representada por dos compartimientos, como se visualiza en la siguiente tabla:

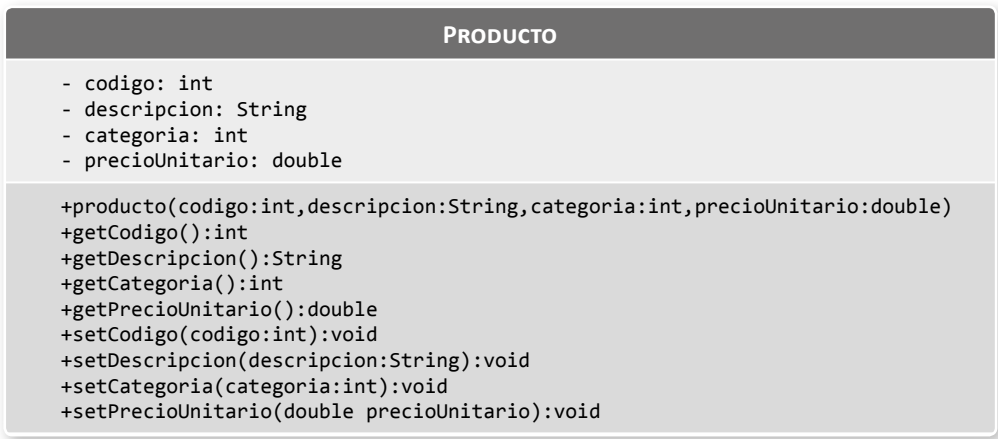
En Java se visualiza con el siguiente script:

class Nombre de la Clase

Nivel de la clase

```
class Nombre de la Clase
{
     Lista de atributos
     Lista de Metodos

}
```

Si necesita implementar la clase Producto en un sistema de ventas tendría que:



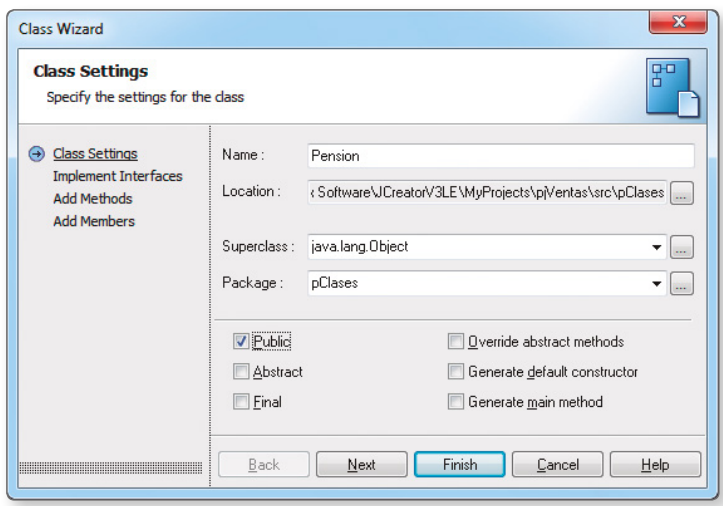
En java tendría el siguiente script de la clase Producto:

```
public class Producto{
     private int codigo;
     private String descripcion;
     private int categoria;
     private double precioUnitario;

     public Producto(int codigo,String descripcion,
                    int categoria,double precioUnitario){
              this.codigo=codigo;
              this.descripcion=descripcion;
              this.categoria=categoria;
              this.precioUnitario=precioUnitario;
     }

     public int getCodigo(){
              return codigo;
     }
```

```
        public String getDescripcion(){
                return descripcion;
        }
        public int getCategoria(){
                return categoria;
        }
        public double getPrecioUnitario(){
                return precioUnitario;
        }

        public void setCodigo(int codigo){
                this.codigo=codigo;
        }
        public void setDescripcion(String descripcion){
                this.descripcion=descripcion;
        }
        public void setCategoria(int categoria){
                this.categoria=categoria;
        }
        public void setPrecioUnitario(double precioUnitario){
                this.precioUnitario=precioUnitario;
        }
    }
```

⊙ AGREGAR UNA CLASE A UN PAQUETE EN JCREATOR

Clic derecho sobre la carpeta src > Add > New Class

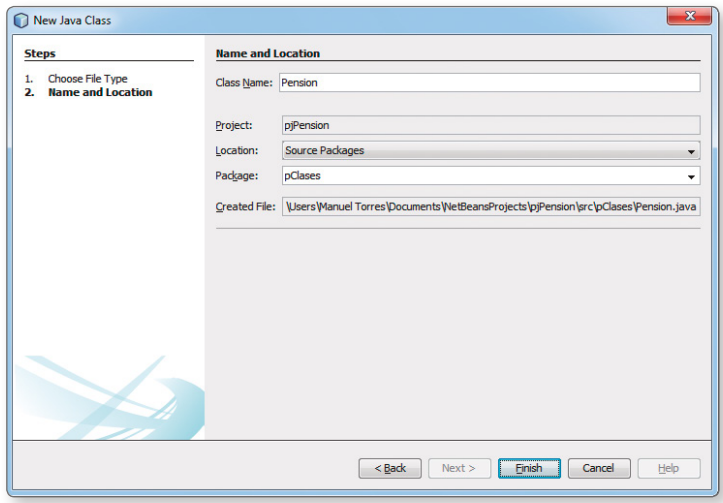


Name: es el nombre asignado a la clase.
Location: es el lugar destino de la clase.
Package: es el paquete destino de la clase y no olvide activar el check Public.

⊙ Agregar una clase a un paquete en NetBeans

Paso 1:

Clic derecho sobre el paquete pClases > New > Java Class



Class Name: es el nombre asignado a la clase.
Project: muestra el nombre del proyecto (no es modificable).
Location: es el lugar destino de las clases, en este caso debe estar seleccionado Source Packages.
Package: muestra el nombre del paquete destino, es decir, donde se almacenara la clase.
Created File: muestra la ruta del archivo creado (no es modificable).

⊙ Atributos de una clase

Un atributo de clase es la característica que puede tener un objeto. Se puede decir que las propiedades son algo así como variables donde se almacenan datos relacionados con los objetos.

Formato:

```
Visibilidad tipoDatos nombreAtributo;
```

Donde:

- Visibilidad: puede ser private, public, protected o sin definición.
- TipoDatos: son los tipos de datos primitivos, pero también se puede hacer referencia a clases específicas.
- NombreAtributo: es el nombre que se le asignará al atributo de clase.

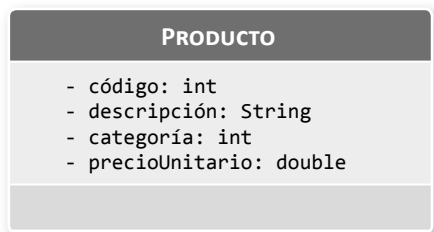
Ejemplos de declaración de atributos:

Ejemplo:

Suponga que tiene la clase Vendedor y se necesita declarar el atributo código. Vea cuántas posibilidades tiene para declarar y cuál es la consecuencia de ello:

DECLARACIÓN DE UN ATRIBUTO	EXPLICACIÓN
`int codigo;`	Al declarar código sin visibilidad solo podrá ser accedida desde la misma clase y no desde el exterior de la clase.
`public int codigo;`	Al declarar código como público todas las clases del mismo proyecto podrán hacer referencia al valor contenido en dicho atributo.
`private int codigo;`	Al declarar como private solo podrá ser accedida desde la clase que se definió. Hay que mencionar que la implementación del atributo como privado es lo más recomendado, puesto que se protege la información contenida en el atributo, es decir, no será manipulada la información.
`protected int codigo;`	Al declarar como protected solo se podrá acceder desde la clase que se definió y las clases descendientes de esta.

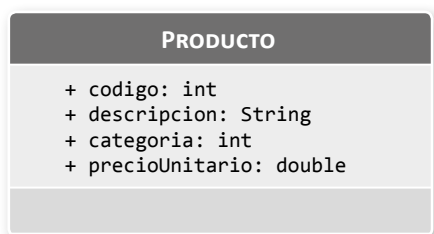
Vea el script que declara los atributos de la clase Producto como privados:



En Java tendría el siguiente script:

```
public class Producto{
       private int codigo;
       private String descripcion;
       private int categoria;
       private double precioUnitario;
}
```

Vea el script que declara los atributos de la clase Producto como públicas:



En Java tendría el siguiente script:

```
public class Producto{
       public int codigo;
       public String descripcion;
       public int categoria;
       public double precioUnitario;
}
```

⊙ Métodos en las clases

En capítulos anteriores se trató sobre la programación modular, la cual dejaba claro que una aplicación puede ser administrada por medio de módulos que dividen el problema en pequeñas porciones que ayudan al programador a realizar un mejor desarrollo de la aplicación. Los métodos de la clase tienen el mismo propósito dentro de la clase, la diferencia es que ahora tomará interés en la visibilidad de los métodos.

Formatos:

```
Visibilidad tipoDatos nombreMetodo();

Visibilidad tipoDatos nombreMetodo(Parametros);
```

Donde:

- Visibilidad: puede ser private, public, protected o sin definición.
- TipoDatos: es el tipo de datos que retornará el método, cuando no retorna valor alguno se colocará la cláusula void.
- NombreMetodo: es el nombre que se le asignará al método de clase, hay que tener en cuenta que dependiendo del trabajo que realice el método se deberá implementar los parámetros.

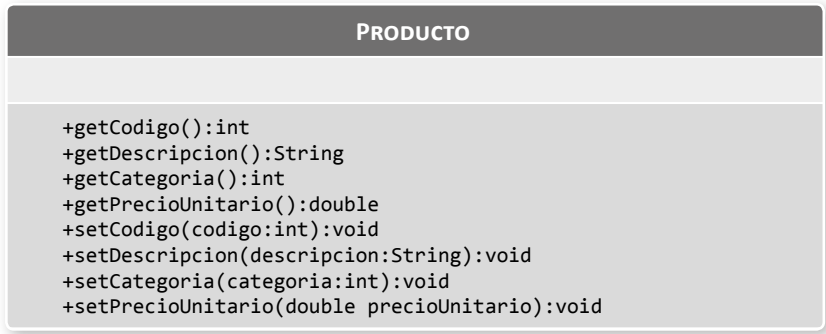
Ejemplos de declaración de métodos.

Ejemplo:

Suponga que tiene la clase Vendedor y se necesita declarar el método getCodigo:

Declaración de un atributo	Explicación
`int getCodigo(){ }`	Se declara el método getCodigo sin visibilidad; por tanto, solo será accesible dentro de la clase.
`public int getCodigo(){ }`	Al declararse como público el método getCodigo podrá ser accesible desde todas las clases de la misma aplicación.
`private int getCodigo(){ }`	Al declararse como privado el método getCodigo solo será accesible dentro de la misma clase.
`protected int getCodigo() { }`	Al declararse como protegido el método getCodigo será accesible desde la misma clase y las clases descendientes.

Si necesita implementar los métodos de la clase Producto en un sistema de ventas tendría:

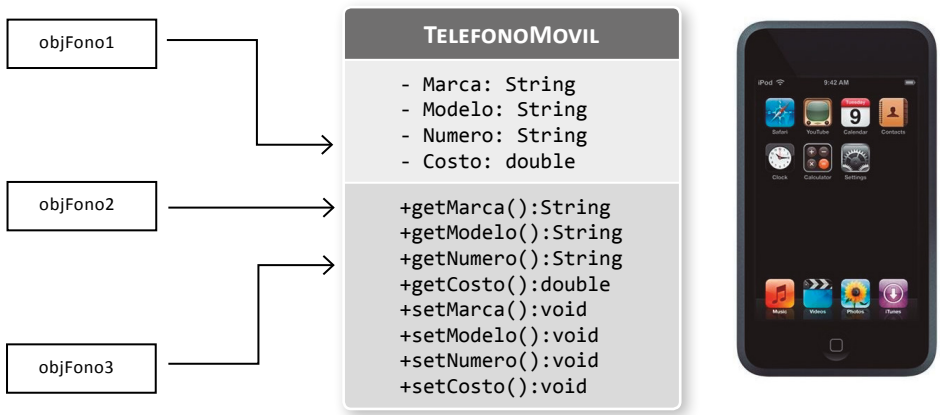


En Java tendríamos el siguiente script de la clase Producto:

```
public class Producto{
  …
        public int getCodigo(){
              return codigo;
        }
        public String getDescripcion(){
              return descripcion;
   }
        public int getCategoria(){
              return categoria;
   }
        public double getPrecioUnitario(){
              return precioUnitario;
        }
   public void setCodigo(int codigo){
              this.codigo=codigo;
        }
        public void setDescripcion(String descripcion){
              this.descripcion=descripcion;
        }
        public void setCategoria(int categoria){
              this.categoria=categoria;
        }
        public void setPrecioUnitario(double precioUnitario){
              this.precioUnitario=precioUnitario;
        }
}
```

8.4. Objetos en Java

Los objetos son copias o ejemplares de una clase. El hecho de crear un objeto permite tener acceso a los atributos y métodos definidos en la clase. La acción de crear un objeto de una clase es llamada instancia de clase. Vea la siguiente gráfica:



En la gráfica anterior se muestra la implementación de la clase telefonoMovil con sus respectivos atributos y métodos. Además hay 3 instancias de clase mediante los objetos objFono1, objFono2 y objFono3.

8.4.1. Formato de creación de objetos en Java

```
Clase Objeto = new Clase();

Clase Objeto;
Objeto = new Clase();
```

Donde:

- Clase: aquí se debe especificar la clase de donde provendrá el objeto, hay que tener en cuenta que si nos encontramos en otro paquete primero se deberá importar usando la instrucción import.
- Objeto: es el nombre que se le asigna al objeto, este será el enlace con los elementos que compone la clase.
- New: es el operador usado por Java para crear un objeto.
- Clase(): es la especificación al método constructor de la clase, consideremos que; si no se ha implementado el método constructor en la clase, Java implementa uno no visible para el usuario.

Vea la creación de los tres objetos de la gráfica anterior:

```
TelefonoMovil objFono1 = new TelefonoMovil();
TelefonoMovil objFono2 = new TelefonoMovil();
TelefonoMovil objFono3 = new TelefonoMovil();

o

TelefonoMovil objFono1,objFono2,objFono3;
objFono1 = new TelefonoMovil();
objFono2 = new TelefonoMovil();
objFono3 = new TelefonoMovil();
```

8.4.2. Formato para referenciar a los atributos de una clase

```
Objeto.atributo = valor o Expresion;
```

Donde:

- Objeto.atributo: el objeto es la referencia a la clase, el punto es obligatorio al momento de invocar algún atributo público de la clase.
- Valor o Expresión: al parámetro se le puede asignar un valor o el resultado de una expresión, solo hay que tener en cuenta el tipo de datos devuelto por el valor o por la expresión sea el mismo especificado en el parámetro.

Vea como asignar un valor a los parámetros de la clase TelefonoMovil:

```
objFono1.Marca = "LG";
objFono1.Modelo = "SmartPhone";
objFono1.Numero = "952-145211";
objFono1.Costo = 1250.00;
```

8.4.3. Formato para referenciar a los metodos de una clase:

```
Objeto.metodo();

Variable = Objeto.Metodo();
```

Donde:

- Objeto.metodo: el objeto es la referencia a la clase. El punto es obligatorio al momento de invocar algún método público de la clase, tiene que considerar el tipo de valor devuelto por el método.
- Variable: esta variable tiene la misión de obtener el resultado de invocar al método de la clase, hay que tener en cuenta que este tipo de instrucción se da solo cuando el método devuelve un valor, es decir, no es de tipo void.

Vea como invocar un método de la clase TelefonoMovil:

```
String marca = objFono1.getMarca();
String modelo = objFono1.getModelo();
String numero = objFono1.getNumero();
double costo = objFono1.getCosto();
```

Este script muestra la invocación de métodos que tienen un valor de retorno específico, ahora vea cómo se invocan a los métodos de tipo void:

```
objFono1.setMarca("LG");
objFono1.setModelo("SmartPhone");
objFono1.setNumero("952-145211");
objFono1.setCosto(1250.00);
```

Caso desarrollado 1: Pago de pensión usando atributos públicos de clase

Implemente una aplicación que permita controlar los pagos que realiza un estudiante de una universidad particular, en donde el estudiante debe realizar un pago mensual por concepto de pensión, de acuerdo a un monto especificado según su categoría:

Categoría del Estudiante	Monto $
A	550.00
B	500.00
C	460.00
D	400.00

Tener en cuenta:

- *Implementar la clase Pensión con los atributos categoría y promedio con visibilidad pública.*
- *Validar todos los valores ingresados mostrando un mensaje de error si fuera necesario.*
- *Aplicar un descuento según el promedio obtenido:*

  18 a 20 > 15%
  16 a 17.99 > 12%
  14 a 15.99 > 10%
  Menor a 13.99 no hay descuento.

- *Se deben imprimir todos los valores obtenidos en un control JList.*
- *Se debe calcular el monto de la pensión, el descuento según su promedio y la pensión actual de un determinado alumno.*

GUI Propuesto:

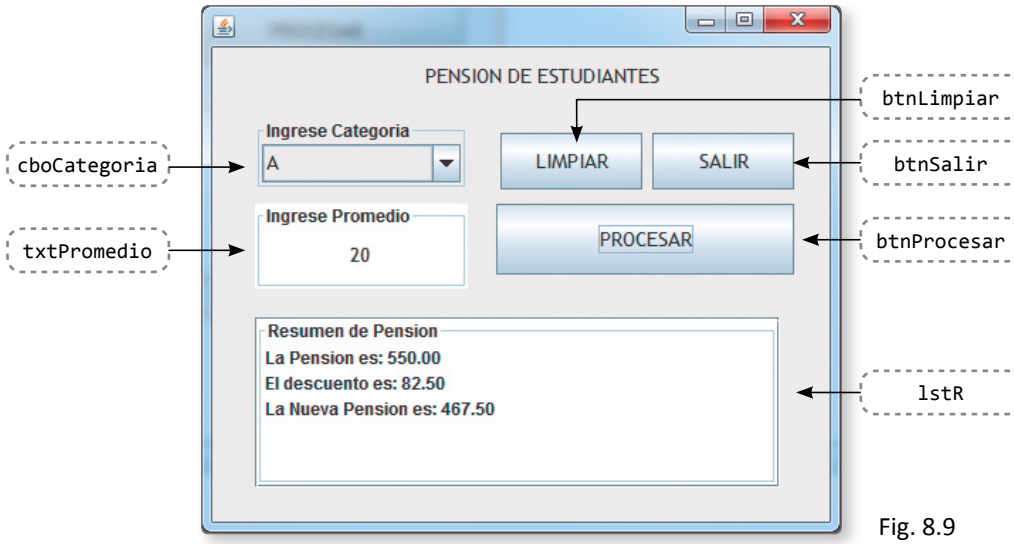
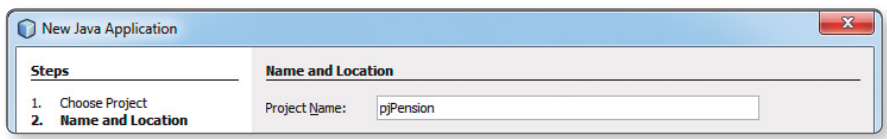


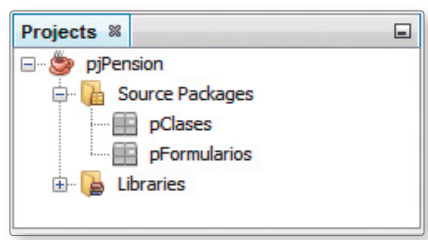
Fig. 8.9

Debe considerar:

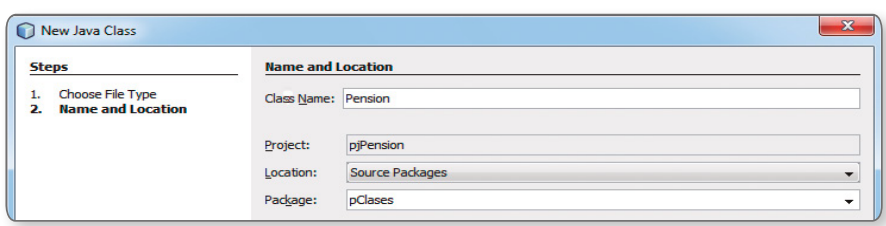
- Crear un nuevo proyecto en NetBeans llamado pjPension:



- Agregar los paquetes pClases y pFormularios al proyecto pjPension.



- Agregar la clase Pension al paquete pClases.



- Agregar la clase frmPension al paquete pFormularios, para esto presione clic derecho sobre el paquete pFormularios > New > JFrame Form.

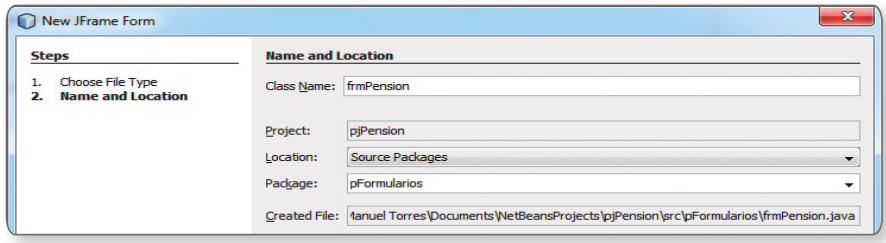


Fig. 8.10

- Asigne un nombre a cada uno de los controles, como se muestra en la Fig. 8.9, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**…
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 8.11:

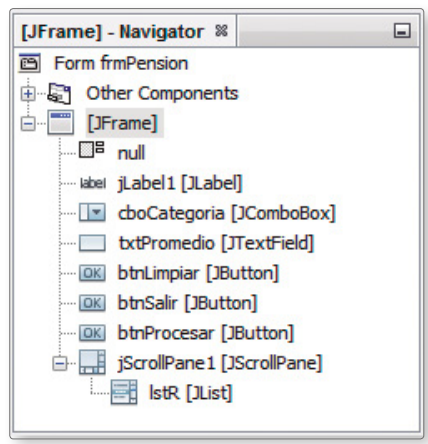


Fig. 8.11

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne las siguientes propiedades a los controles:

cboCategoria	**Border**	TitledBorder=Ingrese Categoria
	Text	Dejar vacio
txtPromedio	**Border**	TitledBorder=Ingrese Promedio
	Text	Dejar vacio
btnProcesar	**Text**	PROCESAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR
lstR	**Border**	TitledBorder=Resumen de Pension
	Model	Dejar vacio

- Vea el siguiente script de la aplicación:

```
package pFormularios;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;
import pClases.Pension;

public class frmPension extends javax.swing.JFrame {

    public frmPension() {
        initComponents();
        llenaCategoria();
}
```

Inicialmente la clase frmPension fue creada dentro del paquete pFormularios es por esa razón que la primera instrucción que muestra la aplicación es package pFormularios. Luego se importan los paquetes necesarios para la aplicación como por ejemplo import javax.swing.DefaultListModel que tiene por misión habilitar el uso de la clase DefaultListModel, import javax.JOptionPane habilita el uso de la clase JOptionPane.showMessageDialog y el más importante de la aplicación es la importación a la clase pensión que se encuentra dentro del paquete pClases.

En el método constructor de la clase frmPension se invoca al método llenaCategoria que tiene por misión llenar el control cboCategorias con las categorías descritas en el caso.

- Vea el script de la clase Pension:

```
package pClases;
public class Pension {
    //1
    public int categoria;
    public double promedio;

    //2
    public double determinaPension(){
        switch(categoria){
            case 0: return 550;
            case 1: return 500;
            case 2: return 460;
            default: return 400;
        }
    }

    public double calculaDescuento(){
        if(promedio<=13.99)
            return 0;
        else if(promedio<=15.99)
            return 0.1*determinaPension();
        else if(promedio<=17.99)
            return 0.12*determinaPension();
        else
            return 0.15*determinaPension();
    }

    public double determinaNuevaPension(){
        return determinaPension()-calculaDescuento();
    }
}
```

Al inicio de la clase aparece la instrucción package pClases que permite definir el marco de trabajo de la clase pensión.

En el punto uno se declara los atributos categoría y promedio con visibilidad o alcance público, esto quiere decir que dichos atributos serán visibles por otras clases de la misma aplicación.

En el punto dos se implemente el método determinaPension que tiene por misión asignar un monto a la pensión según la categoría del alumno, en este caso se usa la estructura switch para evaluar la categoría del alumno; según esto se le asignará un monto de pensión.

También se implementa el método calculaDescuento que tiene por misión determinar el monto de descuento que se le asigna al estudiante, según el promedio registrado. Finalmente, se implementa el método determinaNuevaPension que permite calcular el monto real que el estudiante debe cancelar.

El siguiente script muestra todos los métodos implementados en la aplicación.

```
//1.
void llenaCategoria(){
    cboCategoria.addItem("A");
    cboCategoria.addItem("B");
    cboCategoria.addItem("C");
    cboCategoria.addItem("D");
}

//2.
String valida(){
    if (cboCategoria.getSelectedIndex()==-1)
        return "Categoria del Estudiante";
    else if (txtPromedio.getText().equals("") ||
        Double.parseDouble(txtPromedio.getText())<0 ||
        Double.parseDouble(txtPromedio.getText())>20)
    return "Promedio del Alumno";
        else
    return "";
}

//3.
int getCategoria(){
    return cboCategoria.getSelectedIndex();
}

//4.
double getPromedio(){
    return Double.parseDouble(txtPromedio.getText());
}

//5.
void imprimir(double pension,double descuento,double nuevaPension){
    DefaultListModel moR= new DefaultListModel();
    moR.addElement("La Pension es: "+String.format("%.2f",pension));
    moR.addElement("El descuento es: "+
                                String.format("%.2f",descuento));
    moR.addElement("La Nueva Pension es: "+
                                String.format("%.2f",nuevaPension));
    lstR.setModel(moR);
}
```

```
//6
void limpiarControles(){
        txtPromedio.setText("");
        cboCategoria.setSelectedIndex(-1);
        cboCategoria.requestFocus();
}
```

En el punto uno se implementa el método llenaCategoria que permite llenar de categorías el control cboCategoria.

En el punto dos se implementa el método valida, que permite verificar y comprobar que valores ha ingresado por los controles el usuario. Considere que la devolución de este método es el nombre del control errado, caso contrario devolverá vacío.

En el punto tres se implementa el método getCategoria, que permite obtener la categoría seleccionada por el usuario desde el contgrol cboCategoria.

En el punto cuatro se implementa el método getPromedio, que permite obtener el valor ingresado por el usuario en el control txtPromedio, hay que tener en cuenta que dicho valor es decimal por eso la devolución del método es de tipo double.

En el punto cinco se implementa el método imprimir que permite mostrar los resultados obtenidos en la aplicación. Hay que tener en cuenta que el control que muestra los resultados es un JList; por lo tanto, se necesita un objeto de la clase DefaultListModel.

En el punto seis se implementa el método limpiaControles que permite limpiar los valores ingresados por el usuario y devolver el foco al control cboCategoria para un nuevo ingreso de valores.

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnProcesarActionPerformed(…) {
   if (valida().equals("")){
       //1
       Pension objP = new Pension();

       //2
       objP.categoria = getCategoria();
       objP.promedio=getPromedio();

       //3
       double pension=objP.determinaPension();
       double descuento=objP.calculaDescuento();
       double nuevaPension=objP.determinaNuevaPension();

       imprimir(pension,descuento,nuevaPension);
    } else
      JOptionPane.showMessageDialog(null, "El error esta en "+
                                          valida());
}
```

Lo primero es invocar al método valida para determinar si todos los valores ingresados por el usuario son los correctos; caso contrario enviar un mensaje con el nombre del objeto errado.

En el punto uno, en caso el método valida devuelva vacío, es decir fueron ingresados los valores correctamente; se crea el objeto objP de la clase Pensión, por medio de este objeto podrá tener acceso a los miembros de la clase Pensión.

En el punto dos se asigna al parámetro categoría de la clase Pensión el valor obtenido desde el método getCategoria, categoría de la clase Pensión tiene visibilidad pública; por tanto, la referencia realizada es correcta. En el caso del parámetro promedio de la clase Pensión se le asigna el promedio ingresado por el usuario y capturado por el método getPromedio.

En el punto tres se captura el valor obtenido para la pensión desde el método de la clase Pensión llamada determinaPension, este devolverá el monto asignado al estudiante según su categoría. La variable descuento captura el valor calculado en el método calculaDescuento implementado en la clase Pensión. Luego, la variable nuevaPension obtiene el valor de la nueva pensión calculada por el método determinaNuevaPension de la clase Pensión. Finalmente, se envían los valores obtenidos en las variables locales y son enviadas al método imprimir para poder mostrarlos en el control lstR.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
limpiarControles();
}
```

Como verá dentro del botón btnLimpiar se invoca al método limpiaControles que permitirá limpiar todos los controles usados en la aplicación.

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Pension",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

8.5. Métodos get y set

Los métodos set permiten actualizar el valor de un parámetro asignado con visibilidad privada, si una clase no tiene implementada un método constructor, los métodos set pueden suplir su trabajo haciendo la asignación inicial de los parámetros.

Los métodos get permiten devolver el valor asignado a un parámetro declarado como privado.

8.5.1. Formato para implementar un método set

```
Visibilidad void setNombre(Parametro)
{
    atributoPrivado = Parámetro;
}
```

Donde:

- Visibilidad: representa el alcance del método set, casi siempre se declara como public ya que los valores provienen del exterior de la clase que lo implementa.
- Void: se le asigna el tipo void ya que los métodos void solo asignan un valor al parámetro privado de la clase sin devolver valor alguno, su trabajo es solo asignación de valor.
- setNombre: es el nombre que se le asigna al método set, casi siempre se inicia con la palabra set seguida del nombre del parámetro privado.
- Parametro: es el valor recibido desde el exterior que será asignado al parámetro privado de la clase.
- atributoPrivado=Parámetro: esta asignación resuelve el valor que debe tener un parámetro privado, haciendo que el valor que viene de la clase exterior sea asignada directamente al parámetro privado de la clase.

Por ejemplo, si tiene la siguiente declaración:

```
public class Empleado {
private String codigo;
}
```

Podría implementar el método set del atributo de una de las siguientes formas:

Primero:

```
public void setCodigo(String codigo){
        this.codigo = codigo;
}
```

Segundo:

```
public void setCodigo(String cod){
        codigo = cod;
}
```

Tercero:

```
public void setCodigo(String sCodigo){
        this.codigo = sCodigo;
}
```

Usará cualquiera de ellos, ya que todos los métodos mostrados realizan la misma actividad; seguidamente se mostrará el script completo para la clase.

```
    public class Empleado {
private String codigo;

public void setCodigo (String codigo){
                    this.codigo = codigo;
}
}
```

8.5.2. Formato para implementar un método get

```
Visibilidad tipoDatos getNombre(){
    Return atributoPrivado;
}
```

Donde:

- Visibilidad: representa el alcance del método get, casi siempre se declara como public ya que este método será invocado desde fuera de la clase que se implementó.
- TipoDatos: es el tipo de datos del valor que devolverá el método, hay que tener en cuenta que el parámetro privado y su método get deben tener el mismo tipo de datos.
- getNombre: es el nombre que se le asigna al método get, casi siempre se inicia con la palabra get seguida del nombre del parámetro privado.
- atributoPrivado: Aquí se especifica qué valor devolverá el método, casi siempre se coloca el nombre del parámetro directamente; pero algunos programadores usan el operador This para hacer referencia a un miembro de la clase.

Por ejemplo, si tiene la siguiente declaración:

```
public class Empleado {
private String codigo;
}
```

Podría implementar el método get del atributo código de una de las siguientes formas:

Primero:

```
public String get Codigo(){
        return this.codigo;
}
```

Segundo:

```
public String get Codigo(){
        return codigo;
}
```

Usará cualquiera de ellos, ya que todos los métodos mostrados realizan la misma actividad; seguidamente mostrará el script completo para la clase.

```
    public class Empleado {
private String codigo;

public String getCodigo(){
                    return codigo;
}
}
```

8.5.3. Implementación de metodos get y set con NetBeans:

Para poder crear los métodos get y set en NetBeans debe realizar los siguientes pasos:

1. Tener una clase dentro del paquete respectivo de un proyecto.
2. Declarar los atributos de la clase con visibilidad privada.
3. Presionar clic derecho sobre la clase > Refactor > Encapsulate Fields

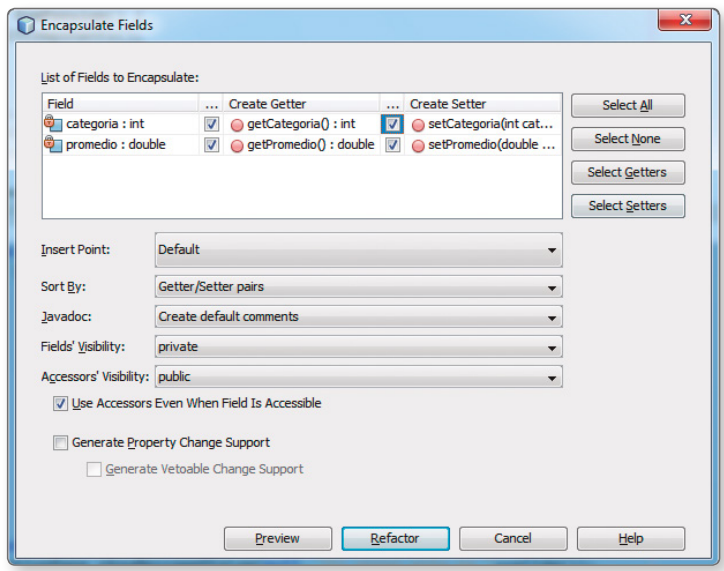


Fig. 8.11

En la fig. 811 se muestran los atributos declarados como List o Campos a encapsular, la ventana muestra una columna con los posibles métodos set llamado Create Setter, para lo cual solo debe activar el check de aquellos atributos que necesite sus métodos set. De la misma forma active los check de los atributos listados en la columna Create Getter. Finalmente, una vez activado los check necesarios presione el botón Refactor.

CASO DESARROLLADO 2: PAGO DE PENSIÓN USANDO ATRIBUTOS PRIVADOS DE CLASE

Implemente una aplicación que permita controlar los pagos que realiza un estudiante de una universidad particular, en donde el estudiante debe realizar un pago mensual por concepto de pensión, de acuerdo a un monto especificado según su categoría:

CATEGORÍA DEL ESTUDIANTE	MONTO $
A	550.00
B	500.00
C	460.00
D	400.00

Tener en cuenta:

- *Implementar la clase Pensión con los atributos categoría y promedio con visibilidad privada.*
- *Validar todos los valores ingresados mostrando un mensaje de error si fuera necesario.*
- *Aplicar un descuento según el promedio obtenido:*

18 a 20 > 15%
16 a 17.99 > 12%
14 a 15.99 > 10%
Menor a 13.99 no hay descuento.

- *Se deben imprimir todos los valores obtenidos en un control JList.*
- *Se debe calcular el monto de la pensión y el descuento según su promedio, y la pensión actual de un determinado alumno.*

GUI Propuesto:

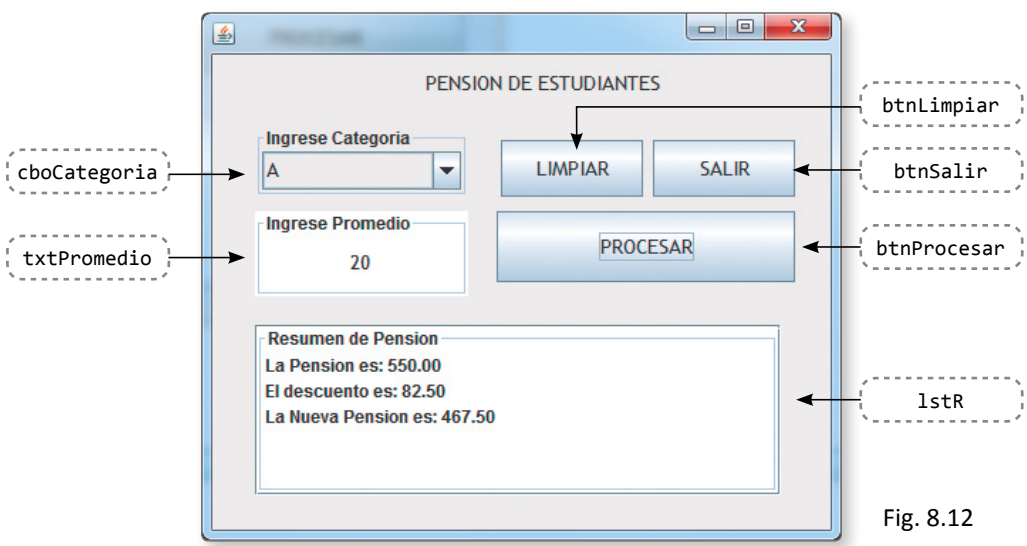
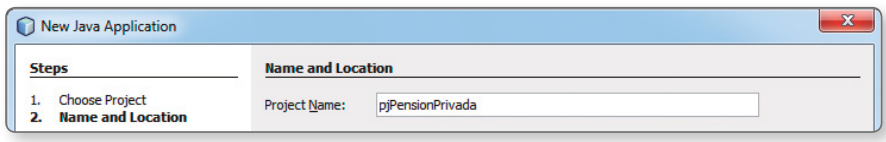


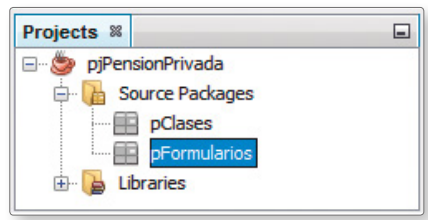
Fig. 8.12

Debe considerar:

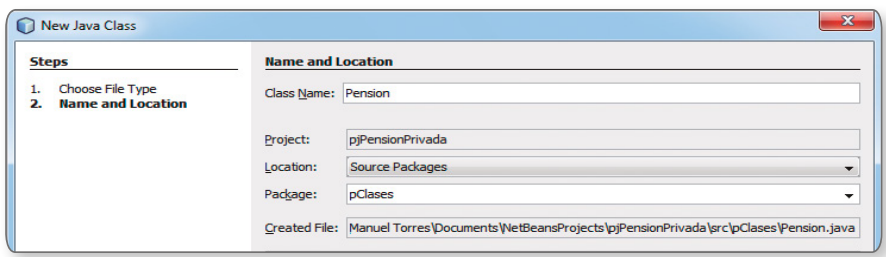
- Crear un nuevo proyecto en NetBeans llamado pjPensionPrivada:



- Agregar los paquetes pClases y pFormularios al proyecto pjPensionPrivada.



- Agregar la clase Pensión al paquete pClases.



- Agregar la clase frmPension al paquete pFormularios, para esto presione clic derecho sobre el paquete pFormularios > New > JFrame Form.

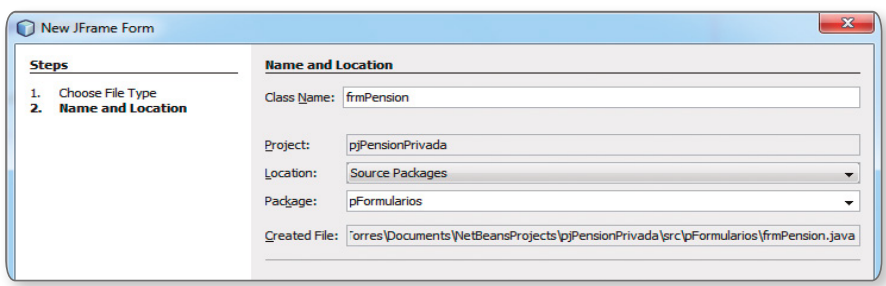


Fig. 8.13

- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 8.12, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos, para eso visualice el panel Navigator: debe mostrarse como la Fig. 8.14.

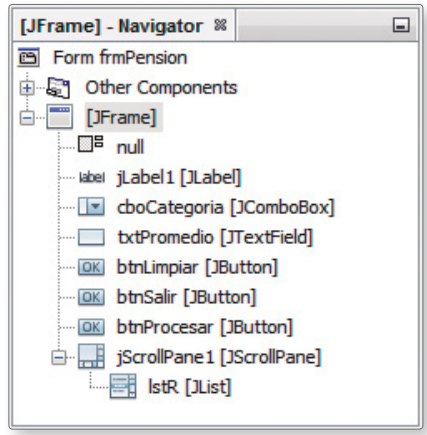


Fig. 8.14

- Asigne Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

cboCategoria	**Border**	TitledBorder=Ingrese Categoria
	Text	Dejar vacio
txtPromedio	**Border**	TitledBorder=Ingrese Promedio
	Text	Dejar vacio
btnProcesar	**Text**	PROCESAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR
lstR	**Border**	TitledBorder=Resumen de Pension
	Model	Dejar vacio

- Vea el siguiente script de la aplicación:

```
package pFormularios;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;
import pClases.Pension;

public class frmPension extends javax.swing.JFrame {

    public frmPension() {
        initComponents();
        llenaCategoria();
}
```

Inicialmente la clase frmPension fue creada dentro del paquete pFormularios, es por esa razón que la primera instrucción que muestra la aplicación es package pFormularios. Luego se importan los paquetes necesarios para la aplicación, como por ejemplo import javax.swing.DefaultListModel, que tiene por misión habilitar el uso de la clase DefaultListModel, import javax.JOptionPane habilita el uso

de la clase JOptionPane.showMessageDialog y el más importante de la aplicación es la importación a la clase pensión que se encuentra dentro del paquete pClases.

En el método constructor de la clase frmPension se invoca al método llenaCategoria que tiene por misión llenar el control cboCategorias con las categorías descritas en el caso.

Vea el script de la clase Pensión:

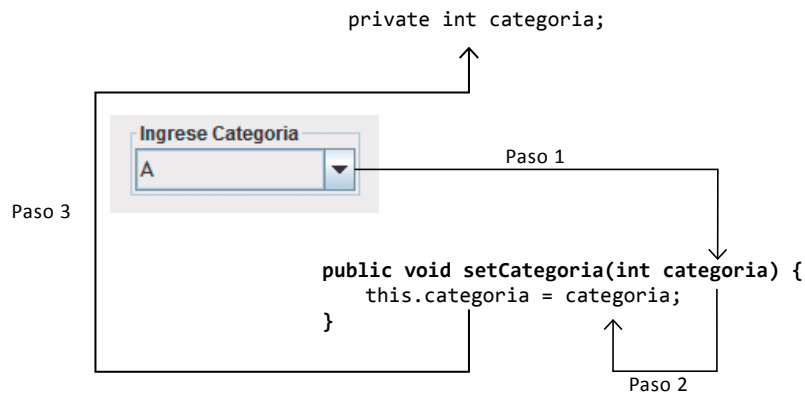
```
package pClases;
public class Pension {
    //1
    private int categoria;
    private double promedio;

    //2
    public double determinaPension(){
        switch(getCategoria()){
            case 0: return 550;
            case 1: return 500;
            case 2: return 460;
            default: return 400;
        }
    }

    public double calculaDescuento(){
        if(getPromedio()<=13.99)
            return 0;
        else if(getPromedio()<=15.99)
            return 0.1*determinaPension();
        else if(getPromedio()<=17.99)
            return 0.12*determinaPension();
        else
            return 0.15*determinaPension();
    }

    public double determinaNuevaPension(){
        return determinaPension()-calculaDescuento();
    }

    //3
    public int getCategoria() {
        return categoria;
    }

    public void setCategoria(int categoria) {
        this.categoria = categoria;
    }

    //4
    public double getPromedio() {
        return promedio;
    }

    public void setPromedio(double promedio) {
        this.promedio = promedio;
    }
}
```

Al inicio de la clase aparece la instrucción package pClases, que permite definir el marco de trabajo de la clase pensión.

En el punto uno se declara los atributos categoría y promedio con visibilidad o alcance privado; por lo tanto, no será visible por la clase frmPension solo por la clase Pensión.

En el punto dos se implementa el método determinaPension, que tiene por misión asignar un monto según la categoría del estudiante.

En el punto tres se implementa el método getCategoria que permite devolver el valor de la categoría asignada al parámetro categoría de la clase Pensión. El método setCategoria permite asignar un valor al parámetro categoría que tiene visibilidad privada para las demás clases. Vea un gráfico que demuestra cómo se asignan los valores a un parámetro privado de la clase:

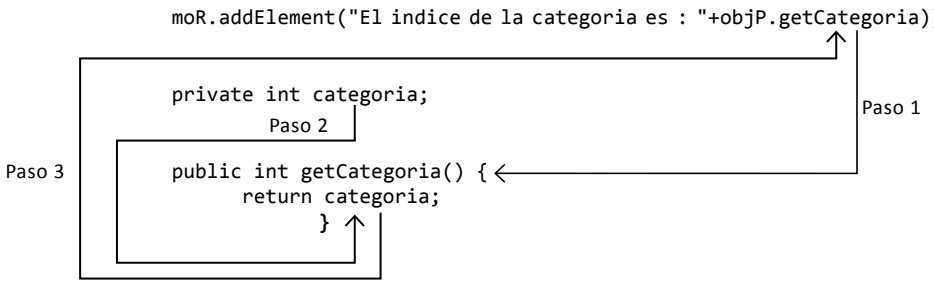


En el paso 1 se envía el valor seleccionado por el usuario al método por medio de la siguiente instrucción: objP.setCategoria(getCategoria()); es decir, se envía el índice del valor seleccionado al parámetro del método; en este caso el valor es 0 (cero).

En el paso 2 el valor 0 (cero) enviado por el parámetro es asignado al parámetro privado categoría mediante la instrucción this.categoria = categoria;

En el paso 3 se muestra como dicho valor es asignado al parámetro privado categoría, eso quiere decir que si alguien de la clase invoca a este parámetro siempre devolverá el valor 0 (cero).

En vista que el método setCategoria se encargó de asignar un valor al parámetro privado categoría de la clase pensión, el método getCategoria devolverá dicha asignación; este método tiene la función de enviar el valor asignado al parámetro privado categoría al exterior.



En el paso 1 se invoca al método getCategoria de la clase Pensión, al llegar al método de la clase; este devuelve el valor asignado al parámetro categoría.
En el paso 2, el valor del parámetro privado categoría es enviado por medio del método getCategoria a quien invocó al método desde el exterior, así como lo muestra el paso 3.

En el punto 4 se implementan los métodos Get y Set del parámetro promedio.

El siguiente script muestra todos los métodos implementados en la aplicación.

```
//1.
void llenaCategoria(){
    cboCategoria.addItem("A");
    cboCategoria.addItem("B");
    cboCategoria.addItem("C");
    cboCategoria.addItem("D");
}

//2.
String valida(){
    if (cboCategoria.getSelectedIndex()==-1)
        return "Categoria del Estudiante";
    else if (txtPromedio.getText().equals("") ||
        Double.parseDouble(txtPromedio.getText())<0 ||
        Double.parseDouble(txtPromedio.getText())>20)
        return "Promedio del Alumno";
    else
        return "";
}

//3.
int getCategoria(){
    return cboCategoria.getSelectedIndex();
}

double getPromedio(){
    return Double.parseDouble(txtPromedio.getText());
}

//4.
void imprimir(double pension,double descuento,double nuevaPension){
    DefaultListModel moR= new DefaultListModel();
    moR.addElement("La Pension es: "+String.format("%.2f",pension));
    moR.addElement("El descuento es: "+
                            String.format("%.2f",descuento));
    moR.addElement("La Nueva Pension es: "+
                            String.format("%.2f",nuevaPension));
    lstR.setModel(moR);
}

//5.
void limpiarControles(){
    txtPromedio.setText("");
    cboCategoria.setSelectedIndex(-1);
    cboCategoria.requestFocus();
}
```

Los puntos fueron explicados en el caso desarrollado 1.

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnProcesarActionPerformed(…) {
   if (valida().equals("")){
       //1
       Pension objP = new Pension();

       //2
       objP.setCategoria(getCategoria());
       objP.setPromedio(getPromedio());

       //3
       double pension=objP.determinaPension();
       double descuento=objP.calculaDescuento();
       double nuevaPension=objP.determinaNuevaPension();

       imprimir(pension,descuento,nuevaPension);
    } else
       JOptionPane.showMessageDialog(null, "El error esta en "+
                                             valida());
    }
```

Lo primero es invocar al método valida para determinar si todos los valores ingresados por el usuario son los correctos; caso contrario enviar un mensaje con el nombre del objeto errado.

En el punto uno se crea el objeto objP de la clase Pensión, por medio de este objeto podrá tener acceso a los miembros de la clase Pensión.

En el punto dos se invoca el método setCategoria de tipo void, es decir, no devuelve ningún valor pero tiene la misión de asignarle un valor al parámetro privado categoría de la clase Pensión. De la misma forma se le asignará el promedio al parámetro privado promedio de la clase Pensión, esta asignación se realizará por medio del método setPromedio.

Los demás puntos implementados no varían, es decir, la explicación de los puntos se encuentra en el caso desarrollado 1.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
limpiarControles();
}
```

Como verá dentro del botón btnLimpiar se invoca al método limpiaControles que permitirá limpiar todos los controles usados en la aplicación.

En el siguiente script se muestra como salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Pension",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

8.6. MÉTODO CONSTRUCTOR

Un constructor es un método especial dentro de la clase, este se encargará de asignar valores iniciales a los parámetros privados de la clase. Hay que tener en cuenta que esta actividad solo la realiza una vez por ejecución, vale decir, que si necesita actualizar o modificar el valor asignado por el constructor tendrá que usar los métodos set.

8.6.1. Formato para la implementación de un método constructor

```
Visibilidad nombreConstructor(Parametro){
    atributoPrivado = Parametro;
}
```

Donde:

- Visibilidad: representa el alcance del método Constructor, casi siempre se declara como public, ya que este método será invocado desde fuera de la clase que se implementó.
- nombreConstructor: es el nombre asignado al método constructor, para poder ser reconocido como constructor debe tener el mismo nombre de la clase.
- Parametro: aquí se especifican los parámetros según la cantidad los atributos a inicializar.
- atributoPrivado=parametro: esta asignación permite enviar el valor del parámetro hacia el atributo privado.

Por ejemplo, si tiene la siguiente declaración:

```
     public class Empleado {
private String codigo;
private String nombres;
private int edad;
}
```

La implementación del método constructor es de la siguiente forma:

```
public Empleado(String codigo,String nombres, int edad){
            this.codigo=codigo;
            this.nombres=nombres;
            this.edad=edad;
}
```

A continuación se verá la implementación completa de la clase empleado:

```
public class Empleado {
private String codigo;
private String nombres;
private int edad;

public Empleado(String codigo,String nombres, int edad){
            this.codigo=codigo;
            this.nombres=nombres;
            this.edad=edad;
        }
    }
```

8.6.2. Creando un objeto de la clase Empleado usando método constructor

Primera forma: enviando los valores al método constructor directamente.

```
            Empleado objEmp = new Empleado("E0001","Juan Lopez G.",50);
```

Segunda forma: enviando variables locales como valores al método constructor.

```
            String codigo=getCodigo();
            String nombres=getNombres();
            int edad=getEdad();

Empleado objEmp = new Empleado(codigo, nombres, edad);
```

Tercera forma: enviando métodos que capturan valores que son enviados al constructor de la clase.

```
        Empleado objEmp = new Empleado(getCodigo(),getNombres(),getEdad());
```

Caso desarrollado 3: Pago de pensión usando método constructor

Implemente una aplicación que permita controlar los pagos que realiza un estudiante de una universidad particular; en donde el estudiante debe realizar un pago mensual por concepto de pensión, de acuerdo a un monto especificado según su categoría:

Categoría del Estudiante	Monto $
A	550.00
B	500.00
C	460.00
D	400.00

Tener en cuenta:

- *Implementar la clase Pensión con los atributos categoría y promedio con visibilidad privada.*
- *Construya el método constructor.*
- *Validar todos los valores ingresados mostrando un mensaje de error si fuera necesario.*
- *Aplicar un descuento según el promedio obtenido:*

  18 a 20 > 15%
  16 a 17.99 > 12%
  14 a 15.99 > 10%
  Menor a 13.99 no hay descuento.

- *Se deben imprimir todos los valores obtenidos en un control JList.*
- *Se debe calcular el monto de la pensión, el descuento según su promedio y la pensión actual de un determinado alumno.*

GUI Propuesto:

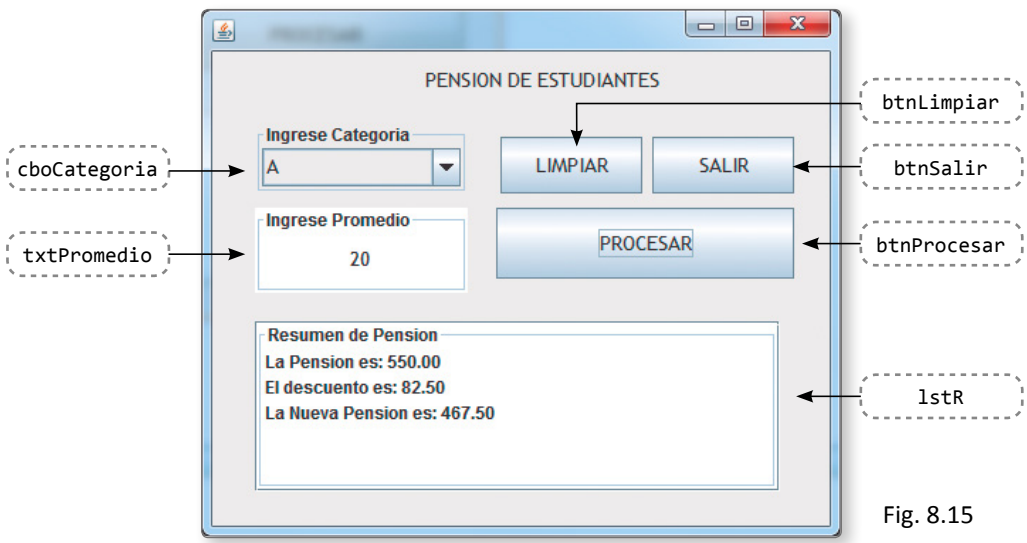
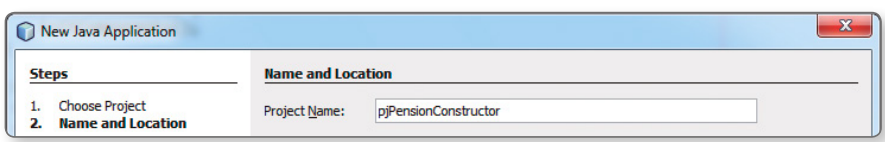


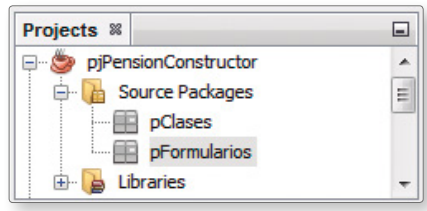
Fig. 8.15

Debe considerar:

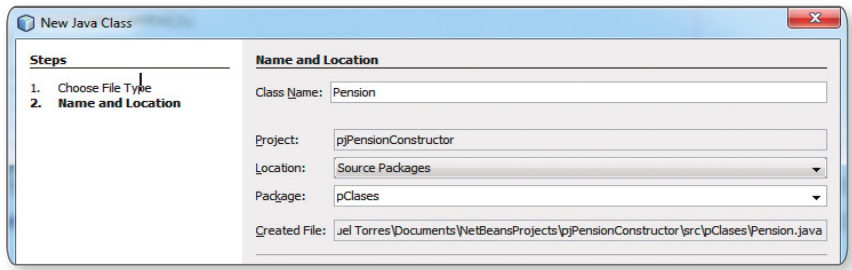
- Crear un nuevo proyecto en NetBeans llamado pjPensionConstructor:



- Agregar los paquetes pClases y pFormularios al proyecto pjPensionConstructor.



- Agregar la clase Pensión al paquete pClases.



- Agregar la clase frmPension al paquete pFormularios, para esto presione clic derecho sobre el paquete pFormularios > New > JFrame Form.

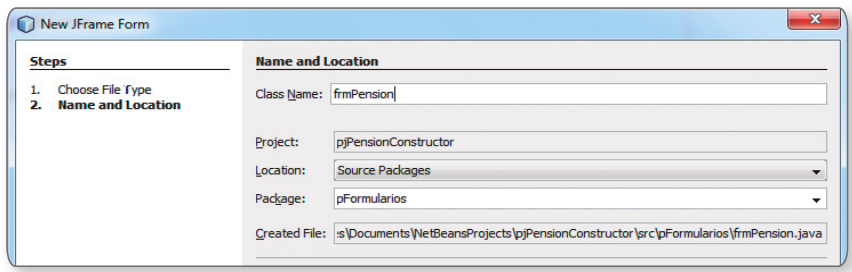


Fig. 8.16

- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 8.16, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 8.17.

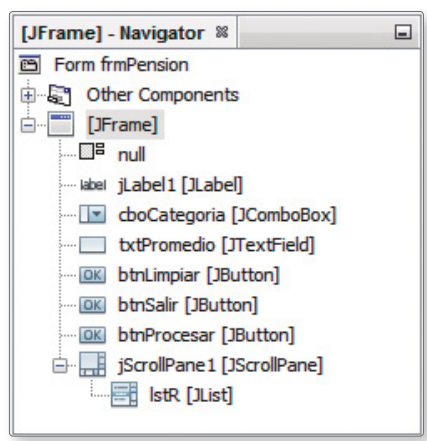


Fig. 8.17

- Asigne Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

cboCategoria	**Border**	TitledBorder=Ingrese Categoria
	Text	Dejar vacio
txtPromedio	**Border**	TitledBorder=Ingrese Promedio
	Text	Dejar vacio
btnProcesar	**Text**	PROCESAR
btnLimpiar	**Text**	LIMPIAR
btnSalir	**Text**	SALIR
lstR	**Border**	TitledBorder=Resumen de Pension
	Model	Dejar vacio

- Vea el siguiente script de la aplicación:

```
package pFormularios;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;
import pClases.Pension;

public class frmPension extends javax.swing.JFrame {

    public frmPension() {
        initComponents();
        llenaCategoria();
}
```

Inicialmente la clase frmPension fue creada dentro del paquete pFormularios es por esa razón que la primera instrucción que muestra la aplicación es package pFormularios. Luego se importan los paquetes necesarios para la aplicación como por ejemplo import javax.swing.DefaultListModel que tiene por misión habilitar el uso de la clase DefaultListModel, import javax.JOptionPane habilita el uso de la clase JOptionPane.showMessageDialog y el más importante de la aplicación es la importación a la clase pensión que se encuentra dentro del paquete pClases.

En el método constructor de la clase frmPension se invoca al método llenaCategoria que tiene por misión llenar el control cboCategorias con las categorías descritas en el caso.

- Vea el script de la clase Pensión:

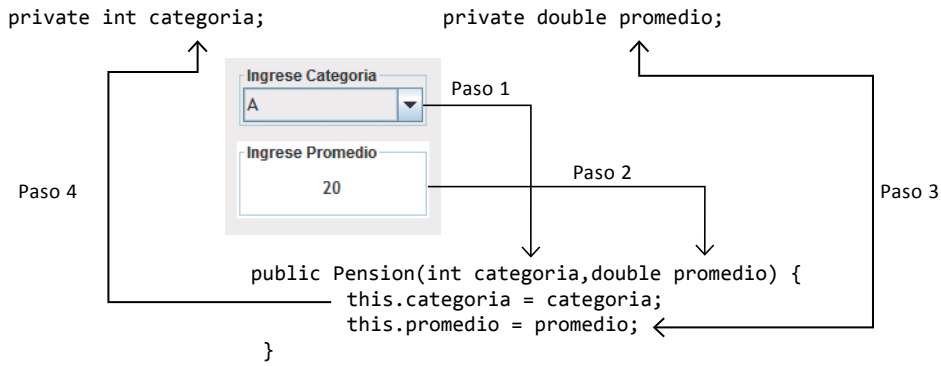
```
package pClases;
public class Pension {
    //1
    private int categoria;
    private double promedio;

    //2
    public Pension(int categoria,double promedio){
        this.categoria=categoria;
        this.promedio=promedio;
    }
```

```
    //3
    public double determinaPension(){
        switch(getCategoria()){
            case 0: return 550;
            case 1: return 500;
            case 2: return 460;
            default: return 400;
        }
    }

    //4
    public double calculaDescuento(){
        if(getPromedio()<=13.99)
            return 0;
        else if(getPromedio()<=15.99)
            return 0.1*determinaPension();
        else if(getPromedio()<=17.99)
            return 0.12*determinaPension();
        else
            return 0.15*determinaPension();
    }

    //5
    public double determinaNuevaPension(){
        return determinaPension()-calculaDescuento();
    }

    //6
    public int getCategoria() {
        return categoria;
    }

    public void setCategoria(int categoria) {
        this.categoria = categoria;
    }

    //7
    public double getPromedio() {
        return promedio;
    }

    public void setPromedio(double promedio) {
        this.promedio = promedio;
    }
}
```

Al inicio de la clase aparece la instrucción package pClases que permite definir el marco de trabajo de la clase pensión.

En el punto uno se declara los atributos de categoría y promedio con visibilidad o alcance privado; por lo tanto, no será visible por la clase frmPension solo por la clase Pensión.

En el punto dos se implementa el método constructor Empleado, que tendrá la misión de asignar un valor a cada atributo declarado como privado, así dará lugar a valores que podrán ser usados dentro de los métodos de la misma clase. Vea la gráfica de cómo recibe los parámetros por medio del método constructor de la clase:



En el paso 1 y 2 se envían los valores obtenidos desde los controles cboCategoria y txtPromedio a los parámetros implementados en el método constructor Pensión. Estos valores son asignados a los parámetros privados de la clase Pensión como se visualiza el paso 3 y 4.

Los puntos siguientes fueron explicados en el caso desarrollado 1.

El siguiente script muestra todos los métodos implementados en la aplicación.

```
//1.
void llenaCategoria(){
    cboCategoria.addItem("A");
    cboCategoria.addItem("B");
    cboCategoria.addItem("C");
    cboCategoria.addItem("D");
}

//2.
String valida(){
    if (cboCategoria.getSelectedIndex()==-1)
        return "Categoria del Estudiante";
    else if (txtPromedio.getText().equals("") ||
        Double.parseDouble(txtPromedio.getText())<0 ||
        Double.parseDouble(txtPromedio.getText())>20)
        return "Promedio del Alumno";
    else
        return "";
}

//3.
int getCategoria(){
    return cboCategoria.getSelectedIndex();
}

double getPromedio(){
    return Double.parseDouble(txtPromedio.getText());
}

//4.
void imprimir(double pension,double descuento,double nuevaPension){
    DefaultListModel moR= new DefaultListModel();
    moR.addElement("La Pension es: "+String.format("%.2f",pension));
    moR.addElement("El descuento es: "+
                            String.format("%.2f",descuento));
    moR.addElement("La Nueva Pension es: "+
                            String.format("%.2f",nuevaPension));
    lstR.setModel(moR);
}
```

```
//5.
void limpiarControles(){
    txtPromedio.setText("");
    cboCategoria.setSelectedIndex(-1);
    cboCategoria.requestFocus();
}
```

Los puntos fueron explicados en el caso desarrollado 1.

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnProcesarActionPerformed(…) {
   if (valida().equals("")){
        //1
        Pension objP = new Pension(getCategoria(),getPromedio());

        //2
        double pension=objP.determinaPension();
        double descuento=objP.calculaDescuento();
        double nuevaPension=objP.determinaNuevaPension();

        imprimir(pension,descuento,nuevaPension);
   } else
        JOptionPane.showMessageDialog(null,
                                      "El error esta en "+valida());
    }
```

En el punto uno se crea el objeto objP de la clase Pensión a la vez se le envía los valores que se usarán en la aplicación por medio del método constructor; debido a la asignación que realiza el método constructor no será necesario el uso de los métodos set.

En el punto dos se declaran las variables locales pensión, descuento y nuevaPension las cuales tienen asignado métodos que están implementados dentro de la clase Pensión.

Los demás puntos implementados no varían, es decir, la explicación de los puntos se encuentra en el caso desarrollado 1.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
limpiarControles();
}
```

Como verá dentro del botón btnLimpiar se invoca al método limpiaControles que permitirá limpiar todos los controles usados en la aplicación.

En el siguiente script se muestra cómo salir de la aplicación preguntando al usuario si está seguro de salir, dependiendo de la respuesta obtenida la aplicación finalizará.

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
      int r=JOptionPane.showOptionDialog(this,
            "Estas seguro de salir...?",
            "Pension",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,null,null);
if (r==0) System.exit(0);
}
```

8.7. REFERENCIA THIS

Permite resolver ambigüedades entre los atributos de una clase y los parámetros de los métodos, muchos métodos de un objeto necesitan acceder a los datos miembros de este, y a otras propiedades del objeto. Para hacer referencia explícitamente a una referencia desde el método se utiliza la palabra reservada this.

Formato de referencia this:

```
this.atributo;
this.metodo();
```

Donde:

- this.Atributo: es la referencia a una variable de clase.
- this.Método: es la referencia a un método de clase.

Por ejemplo, si tiene el método constructor Empleado que tiene como parámetros el código, nombres y edad, que casualmente tienen el mismo nombre que los atributos privados de la clase, debe referirse a ellos por medio de la referencia this:

```
      public Empleado(String codigo, String nombres, int edad) {
this.codigo=codigo;
this.nombres=nombres;
this.edad=edad;
}
```

8.8. Variables y métodos de clase: Modificador Static

En Java, los atributos que contienen el modificador static en su implementación son conocidos como atributos de clase. Uno de los posibles usos del modificador static es para compartir el valor de una variable miembro entre objetos de una misma clase.

8.8.1. Características del modificar Static

- Atributos que son compartidos por todas las instancias de la clase.
- Permite definir clases que pueden ser llamadas desde otras clases sin usar objetos.
- Permite compartir un valor de una variable miembro entre objetos de una misma clase.
- La referencia por medio de this a las variables estáticas genera un error, puesto que la variable estática no puede ser accedida por ningún objeto.

A. Variable de Clase

Es una variable que crea una sola vez para todos los objetos de la misma clase.

Formato:

```
visibilidad static tipoDatos nombreAtributo = valorInicial;
```

Donde:

- Visibilidad: puede ser asignada como pública o privada. Si es declarada como privada necesitara un método get que permitirá devolver el valor.
- static: modificador que asigna una variable de clase.
- tipoDatos: es el tipo de datos de la variable de clase.
- nombreAtributo: es el nombre asignado a la variable de clase.
- valorInicial: si el tipo de datos es numérico se le asignará el valor cero, en caso de ser una cadena se le asignará comillas dobles.

Por ejemplo, se necesita contabilizar las veces que se crea el objeto empleado en una aplicación. Vea la solución en tres pasos:

- Primero, se debe implementar la clase Empleado.

```
public class Empleado{
    private String codigo;
    private String nombres;
    private int edad;
    private static int contador=0;
```

- Segundo, implemente el método constructor.

```
public Empleado(String codigo,String nombres,int edad){
    this.codigo = codigo;
    this.nombres = nombres;
    this.edad = edad;
    contador++;
}
```

- Tercero, implemente un método que devolverá el valor que obtiene la variable de clase contador.

```
public int getContador(){
    return contador;
}
```

Para obtener el valor de la variable de clase, coloque el siguiente script:

```
Empleado objEmp = new Empleado();
JOptionPane.showMessageDialog(null,
            "El total de Empleados es: "+objEmp.getContador());
```

CASO DESARROLLADO 4: PAGO DE EMPLEADOS USANDO VARIABLES DE CLASE.

Una empresa desea tener el control de planilla de sus empleados. La empresa cuenta con la siguiente información: nombres y apellidos del empleado, horas trabajadas, pago por hora según el cargo y las bonificaciones por modalidad de empleo, de acuerdo a las siguientes datos:

CARGO DEL EMPLEADO	PAGO POR HORA
Gerente	$ 20.00
Administrativo	$10.00
Jefe	$ 8.00
Operario	$ 3.50

MODO DE EMPLEO	BONIFICACIÓN
Tiempo Completo	20%
Tiempo Parcial	5%

Se pide calcular:

- *Total de empleados registrados.*
- *Total de empleados registrados como Gerente.*
- *Total de empleados registrados como Administrativo.*

- *Total de empleados registrados como Jefe.*
- *Total de empleados registrados como Operario.*
- *Cantidad de empleados que ganan menos a $1200.00.*
- *Cantidad de empleados que ganan entre $1200.00 y $2500.00.*
- *Cantidad de empleados que ganas más de $2500.00*

GUI Propuesto:

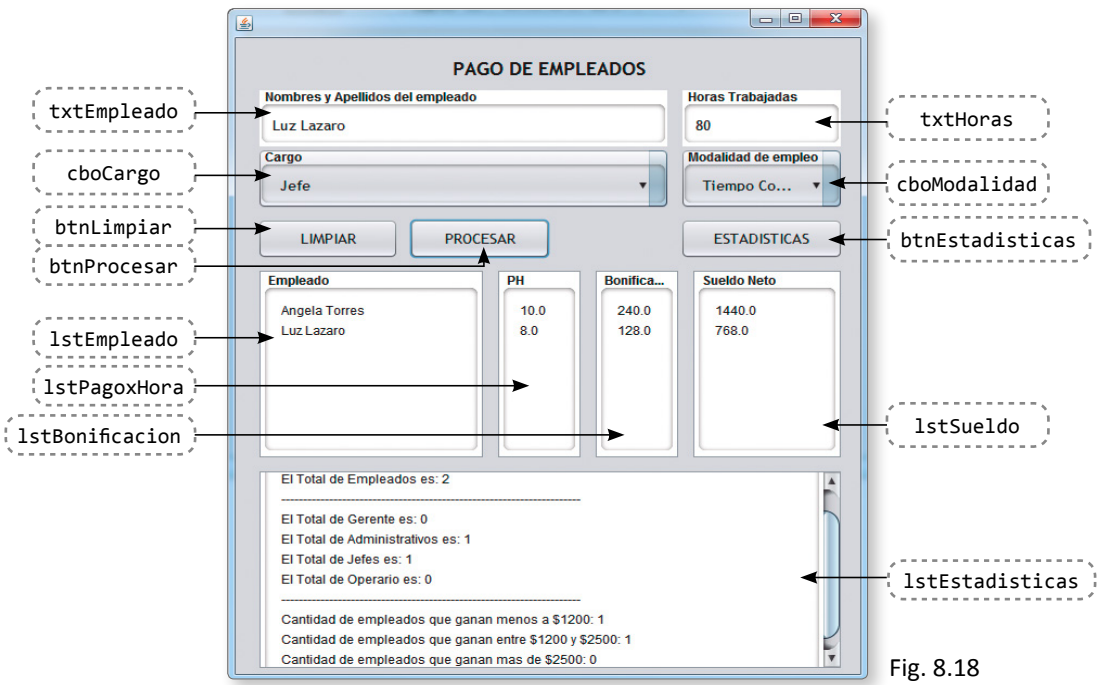
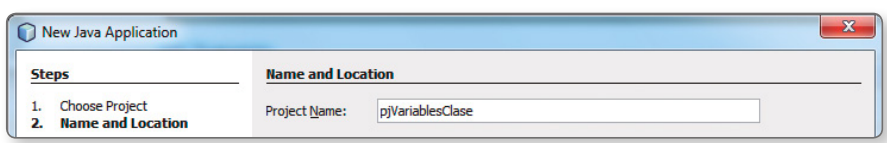


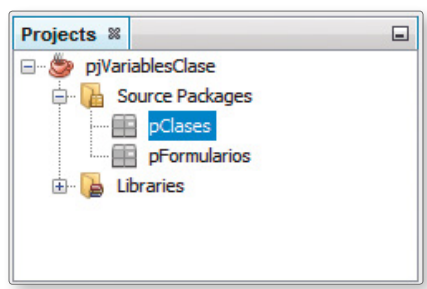
Fig. 8.18

Debe considerar:

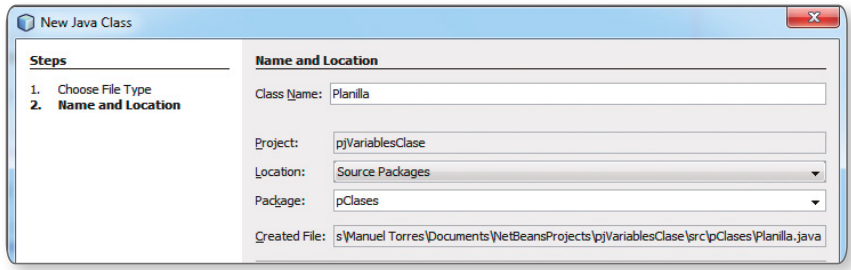
- Crear un nuevo proyecto en NetBeans llamado pjVariablesClase:



- Agregar los paquetes pClases y pFormularios al proyecto pjVariablesClase.



- Agregar la clase Planilla al paquete pClases.



- Agregar la clase frmPago al paquete pFormularios, para esto presione clic derecho sobre el paquete pFormularios > New > JFrame Form.

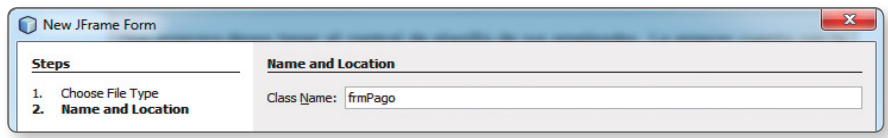


Fig. 8.19

- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 8.18, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**…
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos, para eso visualice el panel Navigator; debe mostrarse como la Fig. 8.19.

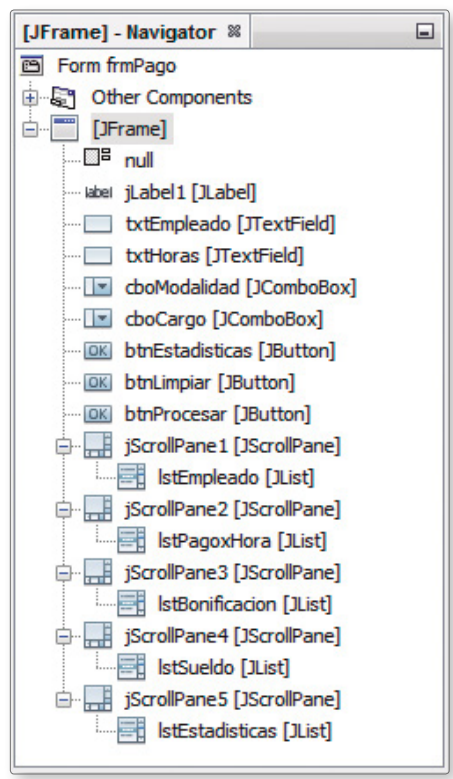


Fig. 8.19

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtEmpleado	**Border**	TitledBorder=Nombres y Apellidos del Emp..
	Text	Dejar vacio
txtHoras	**Border**	TitledBorder=Horas Trabajadas
	Text	Dejar vacio
cboCargo	**Border**	TitledBorder=Cargo
	Model	Dejar vacio
cboModalidad	**Border**	TitledBorder=Modalidad de empleo
	Model	Dejar vacio
btnProcesar	**Text**	PROCESAR
btnLimpiar	**Text**	LIMPIAR
btnEstadisticas	**Text**	ESTADISTICAS
lstEmpleado	**Border**	TitledBorder=Empleado
	Model	Dejar vacio
lstPagoxHora	**Border**	TitledBorder=PH
	Model	Dejar vacio
lstBonificacion	**Border**	TitledBorder=Bonificacion
	Model	Dejar vacio
lstSueldo	**Border**	TitledBorder=Sueldo Neto
	Model	Dejar vacio
lstEstadisticas	**Border**	TitledBorder=Resumen
	Model	Dejar vacio

- Vea el script de la clase Planilla:

```
package pClases;
public class Planilla {
    //1
    private int horas,cargo,modalidad;

    //2
    private static int total=0;
    private static int tGerente=0;
    private static int tAdministrativo=0;
    private static int tJefe=0;
    private static int tOperario=0;

    private static int cMenos1200=0;
    private static int cEntre1200y2500=0;
    private static int cMas2500=0;

    //3
    public Planilla(int horas,int cargo,int modalidad){
        this.horas=horas;
        this.cargo=cargo;
        this.modalidad=modalidad;
        total++;

        contadorCargos();
        contadorSueldo();
    }
```

```
//4
public double asignaPagoxHora(){
    switch(cargo){
        case 0: return 20;
        case 1: return 10;
        case 2: return 8;
        default: return 3.5;
    }
}

//5
public double calculaBruto(){
    return this.horas*asignaPagoxHora();
}

public double asignaBonificacion(double bruto){
    switch(modalidad){
        case 0: return 0.2*bruto;
        default: return 0.05*bruto;
    }
}

public double calculaSueldo(){
    return calculaBruto()+asignaBonificacion(calculaBruto());
}

//6
private void contadorCargos(){
    switch(cargo){
        case 0: tGerente++; break;
        case 1: tAdministrativo++; break;
        case 2: tJefe++; break;
        default: tOperario++;
    }
}
```

En el punto uno se declara los atributos horas, cargo y modalidad de la clase Planilla.

En el punto dos se declaran las variables de clase total (total de empleados registrados), tGerente (total de empleados con cargo de Gerente), tAdministrativo (total de empleados con cargo Administrativo), tJefe (total de empleados con cargo Jefe), tOperario (total de empleados con cargo Operario), cMenos1200 (total de empleados que reciben menos a 1200), cEntre1200y2500 (total de empleados que reciben entre 1200 y 2500) y cMas2500 (total de empleados que reciben más de 2500). Hay que tener en cuenta que todas las variables de clases deben tener un valor inicializador, en este caso por tratarse de conteos, el valor inicial tiene que ser cero.

En el punto tres se implementa el método constructor de la clase Planilla, el cual tiene como parámetros las horas, cargo y la modalidad del empleado, luego se inicializan los atributos de la clase con los valores enviados por los parámetros. En este método se tienen que realizar todos los conteos de la aplicación, pero por ser demasiados cálculos se optó por implementar los conteos en dos métodos: el primero llamado contadorCargos, que contabilizarán los valores según el cargo del empleado y el segundo llamado contadorSueldo, que permitirá contar los empleados cuyo sueldo coincida con la propuesta del caso.

En el punto cuatro se implementa el método asignaPagoxHora, el cual tiene por misión devolver el monto por concepto del pago por hora que se realiza al empleado según el cargo.

En el punto cinco se implementa el método calculaBruto, que permite multiplicar las horas trabajadas por el pago por hora según el cargo del empleado. El método asignaBonificacion tiene por misión determinar el monto de bonificación según la modalidad de trabajo (tiempo completo o tiempo parcial). Finalmente, calculaSueldo devuelve el acumulado entre el monto bruto obtenido y la bonificación.

En el punto seis se implementa el método contadorCargos que permite contabilizar cuantos empleados por cargo se han registrado, tenga en cuenta que 0-Gerente, 1-Administrativo, 2-Jefe y 3-Operario.

En el punto siete se implementa el método contadorSueldo que permite contabilizar la cantidad de empleados cuyo sueldo es menor a 1200 o su sueldo se encuentra entre 1200 y 2500 o supera los 2500.

En el punto ocho se implementan los métodos que permiten devolver todos los contadores.

Vea el script inicial de la clase frmPago:

```
package pFormularios;
import javax.swing.DefaultListModel;
import pClases.Planilla;

public class frmPago extends javax.swing.JFrame {
    //1
    DefaultListModel moEmpleado,moPagoHora,
                     moBonificacion,moSueldo,moEstadisticas;
    //2
    Planilla objP;

    //3
    public frmPago() {
        initComponents();
        llenaCargo();
        llenaModalidad();
        cargaModelos();
    }
```

En el punto uno se declaran los objetos de la clase DefaultListModel, considere que por cada control JList debe tener un modelo.

En el punto dos se declara el objeto objP que será instancia de la clase Planilla.

En el punto tres se invoca al método llenaCargo para llenar los cargos dentro del control cboCargos, llenaModalidad que permitirá llenar las modalidades del empleado en el control cboModalidad y cargaModelos que permitirá inicializar los modelos declarados en el punto uno.

El siguiente script muestra todos los métodos implementados en la aplicación.

```
//1
void llenaCargo(){
   cboCargo.addItem("Gerente");
   cboCargo.addItem("Administrativo");
   cboCargo.addItem("Jefe");
   cboCargo.addItem("Operario");
}

//2
void llenaModalidad(){
    cboModalidad.addItem("Tiempo Completo");
    cboModalidad.addItem("Tiempo Parcial");
}

//3
String getEmpleado(){
    return txtEmpleado.getText();
}

//4
int getHoras(){
    return Integer.parseInt(txtHoras.getText());
}

//5
int getCargo(){
    return cboCargo.getSelectedIndex();
}

//6
int getModalidad(){
    return cboModalidad.getSelectedIndex();
}

//7
void cargaModelos(){
     moEmpleado=new DefaultListModel();
     moPagoHora=new DefaultListModel();
     moBonificacion=new DefaultListModel();
     moSueldo=new DefaultListModel();
     moEstadisticas=new DefaultListModel();

     lstEmpleado.setModel(moEmpleado);
     lstPagoxHora.setModel(moPagoHora);
     lstBonificacion.setModel(moBonificacion);
     lstSueldo.setModel(moSueldo);
     lstEstadisticas.setModel(moEstadisticas);
}

//8
void imprimeEstadisticas(){
     moEstadisticas.clear();
     moEstadisticas.addElement("El Total de Empleados es: "+
                                    objP.getTotal());
     moEstadisticas.addElement("---------------------------------");
     moEstadisticas.addElement("El Total de Gerente es: "+
                                    objP.getTotalGerentes());
     moEstadisticas.addElement("El Total de Administrativos es: "+
                                    objP.getTotalAdministrativo());
     moEstadisticas.addElement("El Total de Jefes es: "+
                                    objP.getTotalJefes());
     moEstadisticas.addElement("El Total de Operario es: "+
```

```
                                        objP.getTotalOperario());
        moEstadisticas.addElement("----------------------------------");
        moEstadisticas.addElement("Cantidad de empleados que ganan
                            menos a $1200: "+objP.getTotalmenor1200());
        moEstadisticas.addElement("Cantidad de empleados que ganan
                entre $1200 y $2500: "+objP.getTotalEntre1200y2500());
        moEstadisticas.addElement("Cantidad de empleados que ganan
                               mas de $2500: "+objP.getTotalmas2500());
    }

    //9
    String valida(){
        if (txtEmpleado.getText().equals(""))
            return "Nombre del Empleado";
        else if(txtHoras.getText().equals("")||
                                Integer.parseInt(txtHoras.getText())<0)
            return "Horas de trabajo";
        else if (cboCargo.getSelectedIndex()==-1)
                return "Cargo del empleado";
        else if (cboModalidad.getSelectedIndex()==-1)
            return "Modalidad del empleado";
        else
            return "";
    }
```

En el punto uno se implementa el método llenaCargo que permite llenar los cargos en el control cboCargo.

En el punto dos se implementa el método llenaModalidad que permite llenar las modalidades del empleado en el control cboModalidad.

En el punto tres se implementa el método getEmpleado que permite obtener el nombre del empleado registrado en la aplicación.

En el punto cuatro se implementa el método getHoras que permite obtener el número de horas de trabajo registradas en la aplicación.

En el punto cinco se implementa el método getCargo que permite obtener el cargo seleccionado por el usuario.

En el punto seis se implementa el método getModalidad que permite obtener la modalidad seleccionada por el usuario.

En el punto siete se implementa el método cargaModelos que tiene la misión de crear los objetos de la clase DefaultListModel. Además de asignar a cada control JList su modelo correspondiente.

En el punto ocho se implementa el método imprimeEstadisticas que permitirá enviar los valores de conteos obtenidos desde la clase Planilla.

En el punto nueve se implementa el método valida que permite comprobar los valores ingresados y seleccionados por el usuario.

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnProcesarActionPerformed(…) {
   if (valida().equals("")){
      objP = new Planilla(getHoras(),getCargo(),getModalidad());

      moEmpleado.addElement(getEmpleado());
      moPagoHora.addElement(objP.asignaPagoxHora());
      double bruto = objP.calculaBruto();
      moBonificacion.addElement(objP.asignaBonificacion(bruto));
      moSueldo.addElement(objP.calculaSueldo());
      } else
        JOptionPane.showMessageDialog(null,
                                      "El error esta en "+valida());
}
```

Dentro del botón procesar se debe crear el objeto de la clase Planilla para poder acceder a los métodos que darán respuesta a la aplicación. Considere que dentro de la clase Planilla se implementó el método constructor; por lo tanto, al crear el objeto objP se necesitará enviar los parámetros como las horas, el cargo y la modalidad de empleo.

Luego se envían los valores obtenidos a los modelos; esto servirá para poder ver resultados en los controles JList.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
    txtEmpleado.setText("");
    txtHoras.setText("");
    cboCargo.setSelectedIndex(-1);
    cboModalidad.setSelectedIndex(-1);
    txtEmpleado.requestFocus();
}
```

El siguiente script se implementa dentro del botón Estadísticas.

```
private void btnEstadisticasActionPerformed(…) {
    imprimeEstadisticas();
}
```

8.8.2. Métodos Estáticos

Los métodos static permiten optimizar la memoria en tiempo de ejecución por no tener que crear instancias para acceder a métodos comunes de los objetos.

- **FORMATO PARA LA IMPLEMENTACIÓN DE UN MÉTODO ESTÁTICO:**

```
Visibilidad static tipoDatos nombreMetodo(){
   return valor;
}
```

Donde:

- Visibilidad: es el alcance (público, privado o protegido) que tiene el método estático implementado.
- static: define el método de clase.
- tipoDatos: es el valor devuelto por el método de clase, hay que tener en cuenta que el valor devuelto a través del return sea del mismo tipo implementado.
- nombreMetodo: es el nombre asignado al método de clase.
- return valor: es el valor devuelto por el método.

- **FORMATO PARA LA INVOCACIÓN DE UN MÉTODO ESTÁTICO:**

```
Variable = NombreClase.NombreMetodoEstatico();
```

Para la invocación del método de clase no será necesario importar el paquete, ya que al método se le llamara directamente desde la clase.

Por ejemplo, se necesita contabilizar las veces que se crea el objeto empleado en una aplicación, veamos el script:

- Primero, se debe implementar la clase Empleado.

```
public class Empleado{
    private String codigo;
    private String nombres;
    private int edad;
    private static int contador=0;
```

- Segundo, implementamos el método constructor.

```
public Empleado(String codigo,String nombres,int edad){
    this.codigo = codigo;
    this.nombres = nombres;
    this.edad = edad;
    contador++;
}
```

- Tercero, implementamos un método de clase que devolverá el valor que obtiene la variable de clase contador.

```
public static int getContador(){
    return contador;
}
```

Para invocar al método, colocamos el siguiente script:

```
JOptionPane.showMessageDialog(null,
            "El total de Empleados es: "+Empleado.getContador());
```

8.8.3. Inicializadores de variables de clase

Los atributos static son inicializados una sola vez dentro de la clase en forma automática, tenemos:

- **FORMATO PARA LA INICIALIZACION DE VARIABLES DE CLASE:**

```
static {
   variableClase=valorInicial;
}
```

Donde:

- variableClase: es el atributo declarado como estático.
- valorInicial: es el valor inicial del atributo estático.

CASO DESARROLLADO 5: PAGO DE EMPLEADOS USANDO MÉTODOS DE CLASE E INICIALIZADORES

Una empresa desea tener el control de planilla de sus empleados. La empresa cuenta con la siguiente información: nombres y apellidos del empleado, horas trabajadas, pago por hora según el cargo y las bonificaciones por modalidad de empleo, de acuerdo a las siguientes datos:

CARGO DEL EMPLEADO	PAGO POR HORA
Gerente	$ 20.00
Administrativo	$10.00
Jefe	$ 8.00
Operario	$ 3.50

Se pide calcular:

- *Total de empleados registrados.*
- *Total de empleados registrados como Gerente.*
- *Total de empleados registrados como Administrativo.*
- *Total de empleados registrados como Jefe.*
- *Total de empleados registrados como Operario.*
- *Cantidad de empleados que ganan menos a $1200.00.*
- *Cantidad de empleados que ganan entre $1200.00 y $2500.00.*
- *Cantidad de empleados que ganas más de $2500.00*

GUI Propuesto:

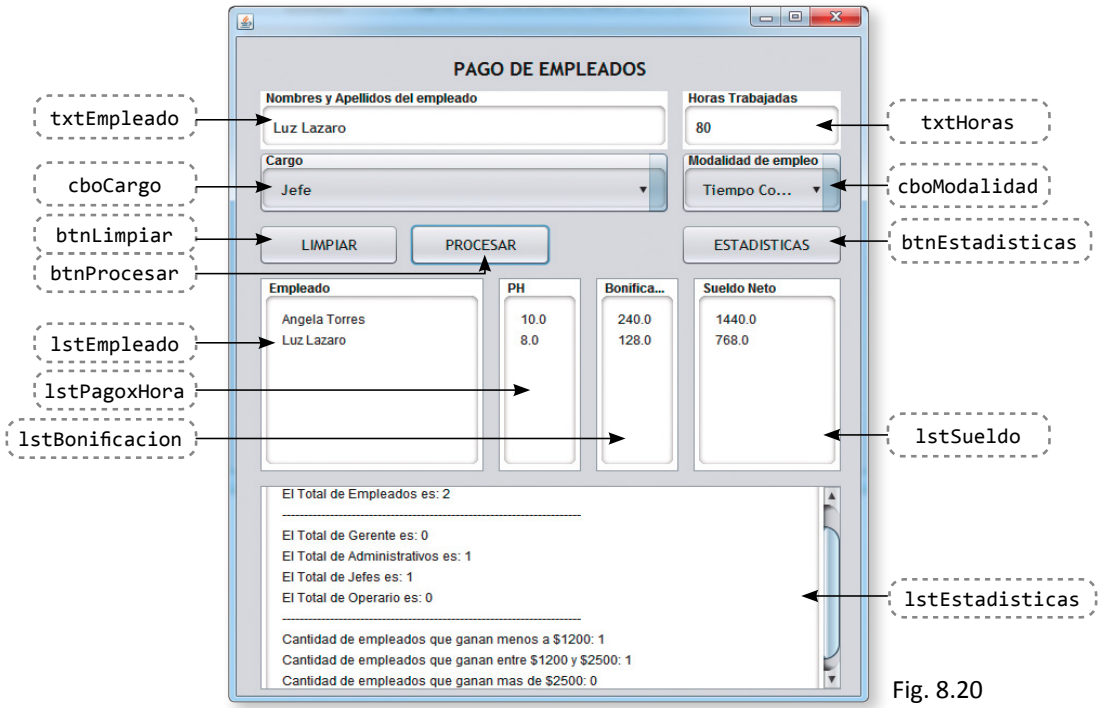
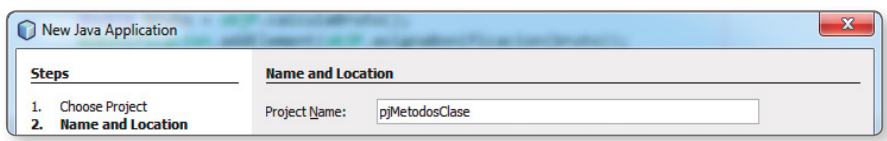


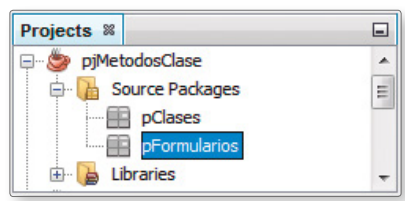
Fig. 8.20

Debe considerar:

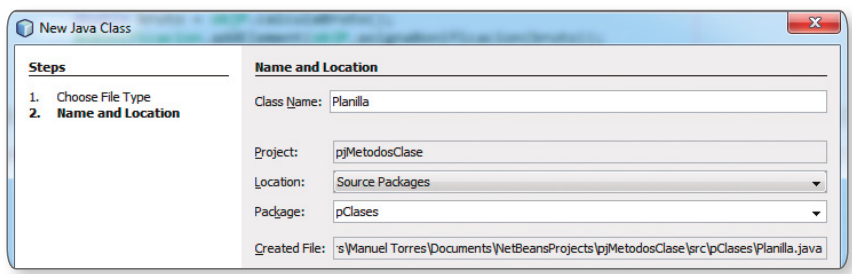
- Crear un nuevo proyecto en NetBeans llamado pjMetodosClase:



- Agregar los paquetes pClases y pFormularios al proyecto pjMetodosClase.



- Agregar la clase Planilla al paquete pClases.



- Agregar la clase frmPago al paquete pFormularios, para esto presione clic derecho sobre el paquete pFormularios > New > JFrame Form.

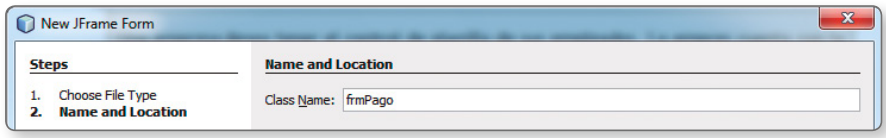


Fig. 8.21

- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 8.20, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 8.21.

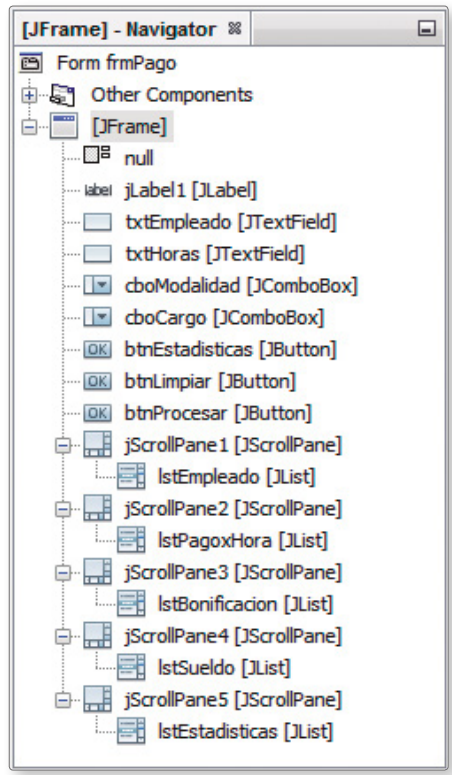


Fig. 8.21

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtEmpleado	**Border**	TitledBorder=Nombres y apellidos del emp
	Text	Dejar vacio
txtHoras	**Border**	TitledBorder=Horas trabajadas
	Text	Dejar vacio
cboCargo	**Border**	TitledBorder=Cargo
	Model	Dejar vacio
cboModalidad	**Border**	TitledBorder=Modalidad de empleo
	Model	Dejar vacio
btnProcesar	**Text**	PROCESAR
btnLimpiar	**Text**	LIMPIAR
btnEstadisticas	**Text**	ESTADISTICAS
lstEmpleado	**Border**	TitledBorder=Empleado
	Model	Dejar vacio
lstPagoxHora	**Border**	TitledBorder=PH
	Model	Dejar vacio
lstBonificacion	**Border**	TitledBorder=Bonificacion
	Model	Dejar vacio
lstSueldo	**Border**	TitledBorder=Sueldo Neto
	Model	Dejar vacio
lstEstadisticas	**Border**	TitledBorder=Resumen
	Model	Dejar vacio

- Vea el script de la clase Planilla:

```
package pClases;
public class Planilla {
    //1
    private int horas,cargo,modalidad;

    //2
    private static int total;
    private static int tGerente;
    private static int tAdministrativo;
    private static int tJefe;
    private static int tOperario;

    private static int cMenos1200;
    private static int cEntre1200y2500;
    private static int cMas2500;

    //3
    static {
        total=0;
        tGerente=0;
        tAdministrativo=0;
        tJefe=0;
        tOperario=0;

        cMenos1200=0;
        cEntre1200y2500=0;
        cMas2500=0;
    }

    //4
    public Planilla(int horas,int cargo,int modalidad){
        this.horas=horas;
```

```
        this.cargo=cargo;
        this.modalidad=modalidad;
        total++;

        contadorCargos();
        contadorSueldo();
    }

    //5
    public double asignaPagoxHora(){
        switch(cargo){
            case 0: return 20;
            case 1: return 10;
            case 2: return 8;
            default: return 3.5;
        }
    }

    //6
    public double calculaBruto(){
        return this.horas*asignaPagoxHora();
    }

    public double asignaBonificacion(double bruto){
        switch(modalidad){
            case 0: return 0.2*bruto;
            default: return 0.05*bruto;
        }
    }

    public double calculaSueldo(){
        return calculaBruto()+asignaBonificacion(calculaBruto());
    }

    //7
    private void contadorCargos(){
        switch(cargo){
            case 0: tGerente++; break;
            case 1: tAdministrativo++; break;
            case 2: tJefe++; break;
            default: tOperario++;
        }
    }

    //8
    private void contadorSueldo(){
        if (calculaSueldo()<1200)
            cMenos1200++;
        else if (calculaSueldo()<=2500)
            cEntre1200y2500++;
        else
            cMas2500++;
    }

    //9
    public static int getTotal(){
        return total;
    }
    public static int getTotalGerentes(){
        return tGerente;
    }
    public static int getTotalAdministrativo(){
```

```
            return tAdministrativo;
        }
        public static int getTotalJefes(){
            return tJefe;
        }
        public static int getTotalOperario(){
            return tOperario;
        }
        public static int getTotalmenor1200(){
            return cMenos1200;
        }
        public static int getTotalEntre1200y2500(){
            return cEntre1200y2500;
        }
        public static int getTotalmas2500(){
            return cMas2500;
        }
    }
```

En el punto uno se declara los atributos privados: horas, cargo y modalidad de la clase Planilla.

En el punto dos se declaran las variables de clase necesarias para los conteos pero no se inicializan los valores solo son declarados como estáticos.

En el punto tres se inicializan los valores estáticos, para este caso se inicializa con el valor cero por tratarse de conteos.

En el punto cuatro se implementa el método constructor de la clase Planilla, el cual tiene como parámetros las horas, cargo y la modalidad del empleado, luego se inicializan los atributos de la clase con los valores enviados por los parámetros. En este método se tienen que realizar todos los conteos de la aplicación, pero por ser demasiados cálculos se optó por implementar los conteos en dos métodos, el primero llamado contadorCargos que contabilizarán los valores según el cargo del empleado y el segundo llamado contadorSueldo que permitirá contar los empleados cuyo sueldo coincida con la propuesta del caso.

En el punto cinco se implementa el método asignaPagoxHora, el cual tiene por misión devolver el monto por concepto del pago por hora que se realiza al empleado según el cargo.

En el punto seis se implementa el método calculaBruto que permite multiplicar las horas trabajadas por el pago por hora según el cargo del empleado. El método asignaBonificacion tiene por misión determinar el monto de bonificación según la modalidad de trabajo (tiempo completo o tiempo parcial). Finalmente, calculaSueldo devuelve el acumulado entre el monto bruto obtenido y la bonificación.

En el punto siete se implementa el método contadorCargos que permite contabilizar cuantos empleados por cargo se han registrado, tenga en cuenta que 0-Gerente, 1-Administrativo, 2-Jefe y 3-Operario.

En el punto ocho se implementa el método contadorSueldo que permite contabilizar la cantidad de empleados cuyo sueldo es menor a 1200 o su sueldo se encuentra entre 1200 y 2500 o supera los 2500.

En el punto nueve se implementan los métodos de clase que permiten devolver los valores obtenidos en las variables de clase (variables estáticas).

Vea el script inicial de la clase frmPago:

```
package pFormularios;
import javax.swing.DefaultListModel;
import pClases.Planilla;

public class frmPago extends javax.swing.JFrame {
    //1
    DefaultListModel moEmpleado,moPagoHora,
                     moBonificacion,moSueldo,moEstadisticas;
    //2
    Planilla objP;

    //3
    public frmPago() {
        initComponents();
        llenaCargo();
        llenaModalidad();
        cargaModelos();
    }
```

En el punto uno se declaran los objetos de la clase DefaultListModel, considere que por cada control JList debe tener un modelo.

En el punto dos se declara el objeto objP que será instancia de la clase Planilla.

En el punto tres se invoca al método llenaCargo para llenar los cargos dentro del control cboCargos, llenaModalidad que permitirá llenar las modalidades del empleado en el control cboModalidad y cargaModelos que permitirá inicializar los modelos declarados en el punto uno.

El siguiente script muestra todos los métodos implementados en la aplicación.

```
//1
void llenaCargo(){
   cboCargo.addItem("Gerente");
   cboCargo.addItem("Administrativo");
   cboCargo.addItem("Jefe");
   cboCargo.addItem("Operario");
}

//2
void llenaModalidad(){
    cboModalidad.addItem("Tiempo Completo");
    cboModalidad.addItem("Tiempo Parcial");
}

//3
String getEmpleado(){
    return txtEmpleado.getText();
}

//4
int getHoras(){
    return Integer.parseInt(txtHoras.getText());
}

//5
int getCargo(){
```

```
    return cboCargo.getSelectedIndex();
}

//6
int getModalidad(){
    return cboModalidad.getSelectedIndex();
}

//7
void cargaModelos(){
     moEmpleado=new DefaultListModel();
     moPagoHora=new DefaultListModel();
     moBonificacion=new DefaultListModel();
     moSueldo=new DefaultListModel();
     moEstadisticas=new DefaultListModel();

     lstEmpleado.setModel(moEmpleado);
     lstPagoxHora.setModel(moPagoHora);
     lstBonificacion.setModel(moBonificacion);
     lstSueldo.setModel(moSueldo);
     lstEstadisticas.setModel(moEstadisticas);
}

//8
void imprimeEstadisticas(){
    moEstadisticas.clear();
    moEstadisticas.addElement("El Total de Empleados es: "+
                              Planilla.getTotal());
    moEstadisticas.addElement("---------------------------------");
    moEstadisticas.addElement("El Total de Gerente es: "+
                              Planilla.getTotalGerentes());
    moEstadisticas.addElement("El Total de Administrativos es: "+
                              Planilla.getTotalAdministrativo());
    moEstadisticas.addElement("El Total de Jefes es: "+
                              Planilla.getTotalJefes());
    moEstadisticas.addElement("El Total de Operario es: "+
                              Planilla.getTotalOperario());
    moEstadisticas.addElement("---------------------------------");
    moEstadisticas.addElement("Cantidad de empleados que ganan
            menos a $1200: "+Planilla.getTotalmenor1200());
    moEstadisticas.addElement("Cantidad de empleados que ganan
            entre $1200 y $2500: "+Planilla.getTotalEntre1200y2500());
    moEstadisticas.addElement("Cantidad de empleados que ganan
            mas de $2500: "+Planilla.getTotalmas2500());
}

//9
String valida(){
    if (txtEmpleado.getText().equals(""))
        return "Nombre del Empleado";
    else if(txtHoras.getText().equals("")||
                            Integer.parseInt(txtHoras.getText())<0)
        return "Horas de trabajo";
    else if (cboCargo.getSelectedIndex()==-1)
            return "Cargo del empleado";
    else if (cboModalidad.getSelectedIndex()==-1)
        return "Modalidad del empleado";
    else
        return "";
}
```

En el punto uno se implementa el método llenaCargo que permite llenar los cargos en el control cboCargo.

En el punto dos se implementa el método llenaModalidad que permite llenar las modalidades del empleado en el control cboModalidad.

En el punto tres se implementa el método getEmpleado que permite obtener el nombre del empleado registrado en la aplicación.

En el punto cuatro se implementa el método getHoras que permite obtener el número de horas de trabajo registradas en la aplicación.

En el punto cinco se implementa el método getCargo que permite obtener el cargo seleccionado por el usuario.

En el punto seis se implementa el método getModalidad que permite obtener la modalidad seleccionada por el usuario.

En el punto siete se implementa el método cargaModelos que tiene la misión de crear los objetos de la clase DefaultListModel. Además de asignar a cada control JList su modelo correspondiente.

En el punto ocho se implementa el método imprimeEstadisticas que permitirá enviar los valores de conteos obtenidos desde la clase Planilla, hay que considerar que para obtener los valores solo hay que colocar el nombre de la clase y el método de clase implementado en la clase Plantilla, no es necesario crear un objeto para la referencia.

En el punto nueve se implementa el método valida que permite comprobar los valores ingresados y seleccionados por el usuario.

El siguiente script muestra las instrucciones del botón btnProcesar.

```
private void btnProcesarActionPerformed(…) {
   if (valida().equals("")){
      objP = new Planilla(getHoras(),getCargo(),getModalidad());

      moEmpleado.addElement(getEmpleado());
      moPagoHora.addElement(objP.asignaPagoxHora());
      double bruto = objP.calculaBruto();
      moBonificacion.addElement(objP.asignaBonificacion(bruto));
      moSueldo.addElement(objP.calculaSueldo());
      } else
        JOptionPane.showMessageDialog(null,
                                    "El error esta en ""+valida());
}
```

Dentro del botón procesar se debe crear el objeto de la clase Planilla para poder acceder a los métodos que darán respuesta a la aplicación. Considere que dentro de la clase Planilla se implementa el método constructor; por lo tanto, al crear el objeto objP se necesitara enviar los parámetros como: las horas, el cargo y la modalidad de empleo.

Luego se envían los valores obtenidos a los modelos; esto servirá para poder ver resultados en los controles JList.

En el siguiente script se muestra el código para limpiar los controles y volver a ingresar valores al caso.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
    txtEmpleado.setText("");
    txtHoras.setText("");
    cboCargo.setSelectedIndex(-1);
    cboModalidad.setSelectedIndex(-1);
    txtEmpleado.requestFocus();
}
```

El siguiente script se implementa dentro del botón Estadísticas.

```
private void btnEstadisticasActionPerformed(…) {
    imprimeEstadisticas();
}
```

CAP.

9

Manejo de excepciones

CAPACIDAD:

- Reconoce el manejo del bloque Try-Catch
- Implementa aplicaciones usando el control de excepciones Java

CONTENIDO:

9.1. INTRODUCCIÓN

Cuando se desarrolla una aplicación, sea cualquier lenguaje de programación, siempre pensará en implementarla a cero errores, pero muchas veces no se prevé los posibles errores que puede ocasionar un usuario en la aplicación. Esto resulta algo particular, imagine usted comprando un producto que en la tienda comercial le aseguraron que no tendría problemas con el aparato por lo menos unos varios años, pero a veces eso no es tan real y quedamos algo decepcionados de esto y creo que jamás volverá a comprar en dicha tienda. Algo así sucede con las aplicaciones, pero esta vida es así estamos rodeados de errores pero con experiencia en ellos se podría evitar, quizás previniendo, anticipándose o siendo un vidente de los posibles errores que se pueda ocasionar en una aplicación. Con esto no quiero decir que sea complicado controlar los errores; sino más bien que el control de estos no debe quedar desapercibido por el desarrollador.

En Java las situaciones que pueden provocar un fallo en la aplicación se le denomina excepción, en este capítulo se hablará y mostrará como controlarlos de la mejor manera. Las excepciones en Java son objetos de clases derivadas de la clase base Exception. Existen también los errores internos que son objetos de la clase Error que no estudiaremos. Ambas clases Error y Exception son clases derivadas de la clase base Throwable.

Java implementa el código de intento para controlar las excepciones que se puede ocasionar en una aplicación, este bloque de código protege la secuencialidad del script y si se ocasiona una excepción, esta será controlada mediante un mensaje o alguna instrucción; y no permitirá devolverlo al script de la aplicación, como ocurre con cualquier excepción no controlada.

Los errores que podemos encontrar en las aplicaciones podrían ser:

1.- Por divisiones entre cero.

2.- Al intentar acceder a elementos de arreglos con un índice mayor al declarado.

3.- Al acceder a una base de datos.

4.- En el manejo de archivos (existencia o permiso).

5.- Errores definidos por el usuario.

9.2. TRY-CATCH

Un bloque Try permite proteger un script que podría generar algún tipo de error al ejecutar una aplicación.

Formato:

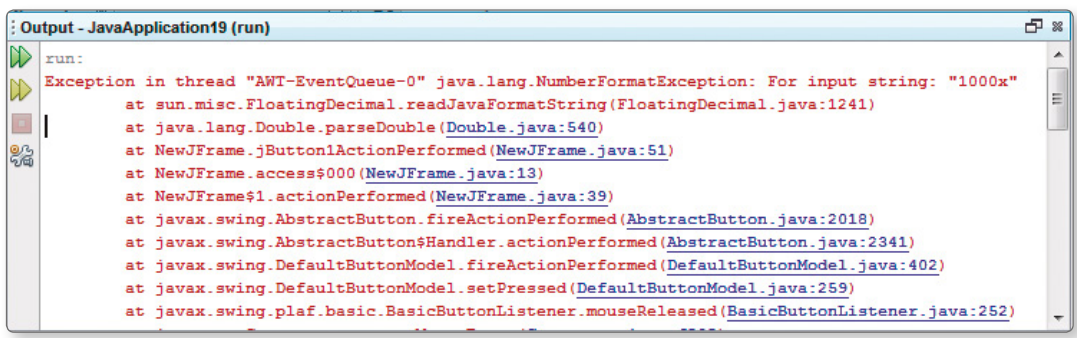
```
try{
    //Instrucciones
} catch(clase identificador){
   //instrucciones catch
}
```

Dentro del bloque try se coloca todo script que pueda generar algún tipo de error, al terminar este bloque se define un grupo de instrucciones en un bloque catch. Cada bloque catch es seguido de un paréntesis que contiene una clase y un identificador, que son considerados como datos de entrada para cada sección catch; el tipo de clase definida dentro de cada sección catch depende del número de excepciones que pueden ser generadas por una misma aplicación.

Vea un script que muestra una excepción:

```
String sSueldo="1000x";
double dSueldo=Double.parseDouble(sSueldo);
JOptionPane.showMessageDialog(null, "El resultado es: "+dSueldo);
```

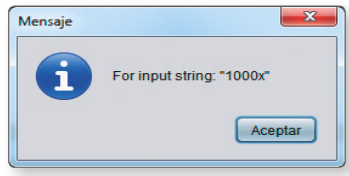
El error parece ser bien claro, pues tenemos una variable de tipo cadena con un valor inicial de 1000x al convertirlo al tipo double se genera el siguiente error:



Podría controlar la excepción por medio del siguiente script:

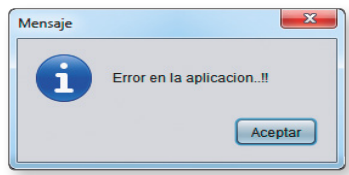
```
try{
   String sSueldo="1000x";
   double dSueldo=Double.parseDouble(sSueldo);
   JOptionPane.showMessageDialog(null, "El resultado es: "+dSueldo);
} catch(NumberFormatException ex){
         JOptionPane.showMessageDialog(null, ex.getMessage());
}
```

El resultado sería como se muestra en la siguiente imagen:



O podría mostrar un mensaje genérico de la excepción con el siguiente script:

```
try{
   String sSueldo="1000x";
   double dSueldo=Double.parseDouble(sSueldo);
   JOptionPane.showMessageDialog(null, "El resultado es: "+dSueldo);
} catch(Exception ex){
   JOptionPane.showMessageDialog(null, "Error en la aplicacion..!! ");
}
```



9.3. CLÁUSULA THROW

Es la clausula que permite listar los tipos de excepción que un método privado puede lanzar. También se le llama lanzamiento manual de excepción.

a. Se usa para proteger los métodos de una clase.
b. Es necesario usarla con todas las excepciones excepto la clase Error y RuntimeException.
c. Cuando se implementa un método se tiene que especificar la cláusula throw, caso contrario genera un error.

Formato de implementación de un método:

```
Visibilidad tipoDatos nombreMetodo(Parametros) throws nombreExcepcion{
   //Instrucciones
}
```

Formato de implementación de clase heredada de la clase Exception:

```
public class nombreExcepcion extends Exception{

    public nombreExcepcion(){
        super("Mensaje1");
    }
    public nombreExcepcion(tipo valor){
       super(valor);
    }
}
```

Vea un ejemplo de cómo validar un número entero usando throw:

PASO 1:

Implementar la clase validaNota con dos métodos constructores, el primero mostraré el mensaje "Fuera del Rango".

```
public class validaNota extends Exception{
    public validaNota(){
        super("Error al convertir");
    }
    public validaNota(String valor){
       super(valor);
    }
}
```

PASO 2:

Se implementa el método validaNumero que invoca a la clase validaNota por medio de la cláusula throws.

```
String validaNumero(int nota) throws validaNota{
        if (nota<0 || nota>20)
            throw new validaNota();

        return "Numero valido";
}
```

PASO 3:

Se implementa el bloque try…catch, este bloque puede colocarse dentro del botón de acción de un formulario.

```
try {
     int nota=22;
     String valida=validaNumero(nota);
     JOptionPane.showMessageDialog(null, valida);
} catch (validaNota ex) {
     JOptionPane.showMessageDialog(null, ex);
}
```

9.4. BLOQUE FINALLY

Es un bloque que permite ejecutar un conjunto de instrucciones ocurra o no una excepción, controlado con try..catch. Normalmente es usado para liberar recursos de la aplicación.

Formato:

```
try{
    //Instrucciones
} catch(clase identificador){
   //Instrucciones catch
} finally {
   //Instrucciones Finally
}
```

CASO DESARROLLADO 1: REGISTRO DE LIBROS

Implemente una aplicación que permita registrar la información de los libros en una biblioteca como son: Nombre del libro, Tipo de Editorial (A, B o C), clase de libro (Programación, Análisis, Diseño), año de edición, número de páginas y costo del libro. La aplicación deberá mostrar los siguientes resultados:

a. *Número de libros de Análisis de la editorial B.*

b. *Mostrar el nombre del libro con el año de edición más reciente.*

c. *Nombre de la editorial que tiene el libro con el menor número de páginas.*

d. *Nombre del libro que tenga el mayor costo.*

GUI Propuesto:

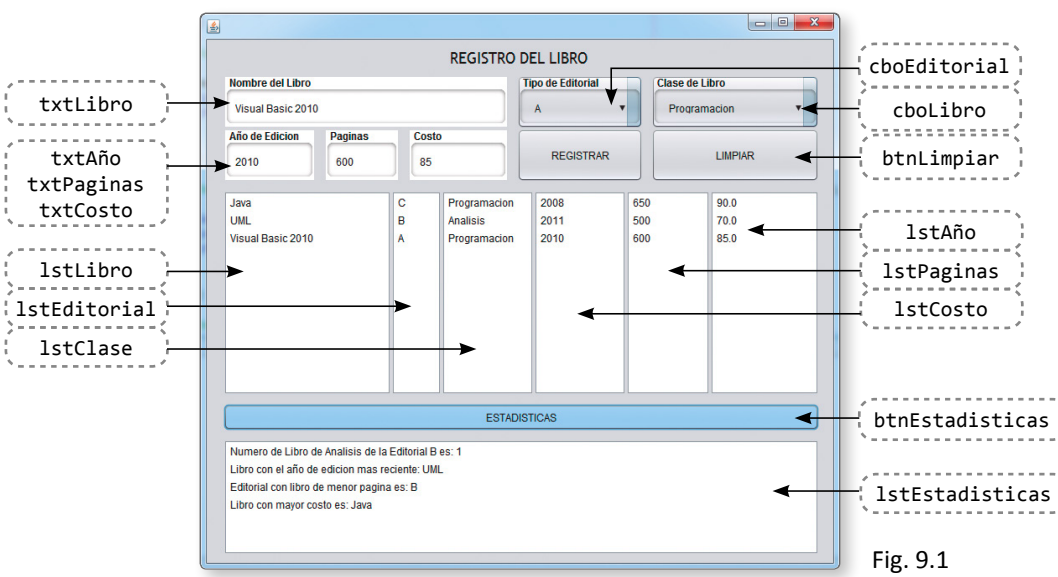
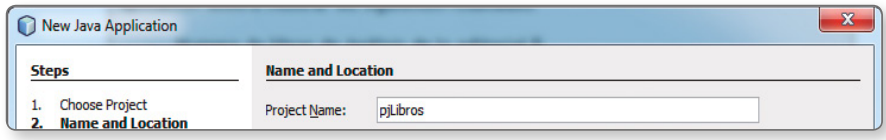


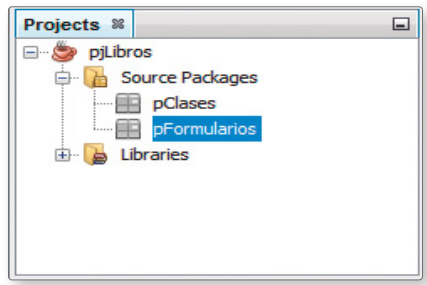
Fig. 9.1

Debe considerar:

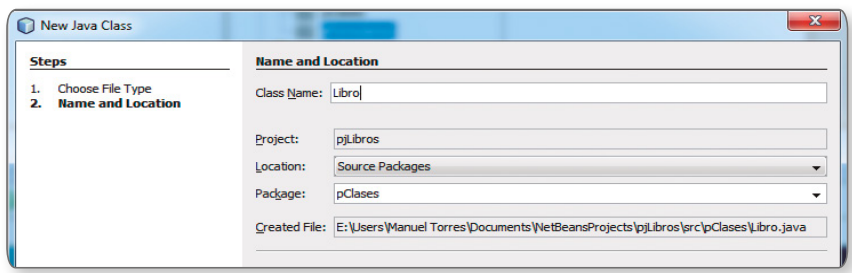
- Crear un nuevo proyecto en NetBeans llamado pjLibros:



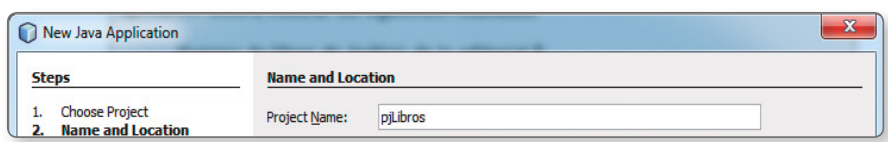
- Agregar los paquetes pClases y pFormularios al proyecto pjLibros.



- Agregar la clase Planilla al paquete pClases.



- Agregar la clase frmPago al paquete pFormularios, para esto presione clic derecho sobre el paquete pFormularios > New > JFrame Form.



- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 9.1, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 9.2.

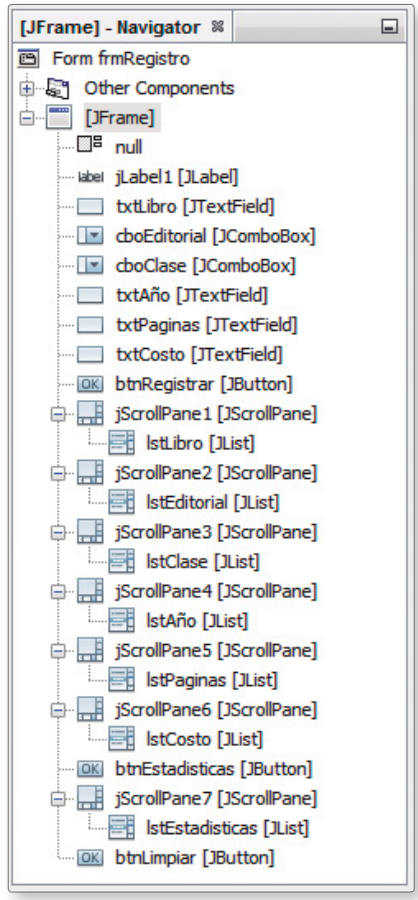


Fig. 9.2

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

txtLibro	**Border** **Text**	TitledBorder=Nombre del libro Dejar vacío
txtAño	**Border** **Text**	TitledBorder=Año de edicion Dejar vacío
txtPaginas	**Border** **Text**	TitledBorder=Paginas Dejar vacío
txtCosto	**Border** **Text**	TitledBorder=Costo Dejar vacío
cboEditorial	**Border** **Model**	TitledBorder=Tipo de Editorial Dejar vacío
cboClase	**Border** **Model**	TitledBorder=Clase de Libro Dejar vacío
btnRegistrar	**Text**	REGISTRAR
btnLimpiar	**Text**	LIMPIAR
btnEstadisticas	**TextI**	ESTADISTICAS

lstLibro	**Model**	Dejar vacío
lstEditorial	**Model**	Dejar vacío
lstClase	**Model**	Dejar vacío
lstAño	**Model**	Dejar vacío
lstPaginas	**Model**	Dejar vacío
lstCosto	**Model**	Dejar vacío
lstEstadisticas	**Model**	Dejar vacío

- Vea el script de la clase Libro:

```
package pClases;

public class Libro {
    //Atributos de la clase Libro
    private String nombre;
    private String editorial;
    private String clase;
    private int año;
    private int paginas;
    private double costo;

    //Variables de clase
    private static int tAnalisisB;

//Implementacion del metodo constructor
public Libro(String nombre,String editorial,String clase,
             int año,int paginas,double costo){
    this.nombre=nombre;
    this.editorial=editorial;
    this.clase=clase;
    this.año=año;
    this.paginas=paginas;
    this.costo=costo;

    conteos();

}

//Metodos GET/SET de la clase
public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getEditorial() {
    return editorial;
}

public void setEditorial(String editorial) {
    this.editorial = editorial;
}
```

```
public String getClase() {
    return clase;
}

public void setClase(String clase) {
    this.clase = clase;
}

public int getAño() {
    return año;
}

public void setAño(int año) {
    this.año = año;
}

public int getPaginas() {
    return paginas;
}

public void setPaginas(int paginas) {
    this.paginas = paginas;
}

public double getCosto() {
    return costo;
}

public void setCosto(double costo) {
    this.costo = costo;
}

public int getTAnalisis(){
    return tAnalisisB;
}

//Metodo que permite contar segun el conteo establecido en el caso
void conteos(){
    if (getClase().equals("Analisis") &&
                        getEditorial().equals("B"))
        tAnalisisB++;
    }
}
```

- Vea el script inicial de la clase frmRegistro:

```
package pFormularios;
//Importacion de librerias necesarias en la aplicación
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;
import pClases.Libro;

public class frmRegistro extends javax.swing.JFrame {

    //Declarando el objeto objLi de la clase Libro
    Libro objLi;
```

```
    //Declarando los modelos por cada lista
    DefaultListModel moLibro,moEditorial,moClase,moAño,
                     moPaginas,moCosto;
    DefaultListModel moEstadisticas;

    public frmRegistro() {
        initComponents();

        //Invocando a los metodos que llenan los controles JCombo
        llenaEditorial();
        llenaClase();

        //Invocando al método que crea los objetos de tipo DefaultList
        cargaModelos();
    }
```

El siguiente script se muestra todos los métodos implementados en la aplicación.

```
//Método que permite crear los objetos de la clase DefaultListModel
//Además asigna los controles JList con su modelo correspondiente
void cargaModelos(){
    moLibro=new DefaultListModel();
    moEditorial=new DefaultListModel();
    moClase=new DefaultListModel();
    moAño=new DefaultListModel();
    moPaginas=new DefaultListModel();
    moCosto=new DefaultListModel();

    moEstadisticas=new DefaultListModel();

    lstLibro.setModel(moLibro);
    lstEditorial.setModel(moEditorial);
    lstClase.setModel(moClase);
    lstAño.setModel(moAño);
    lstPaginas.setModel(moPaginas);
    lstCosto.setModel(moCosto);

    lstEstadisticas.setModel(moEstadisticas);
}

//Método muestra los valores seleccionados en las listas por medio
//de sus modelos correspondientes
void llenaModelos(){
    moLibro.addElement(objLi.getNombre());
    moEditorial.addElement(objLi.getEditorial());
    moClase.addElement(objLi.getClase());
    moAño.addElement(objLi.getAño());
    moPaginas.addElement(objLi.getPaginas());
    moCosto.addElement(objLi.getCosto());
}

//Método que llena el control cboEditorial de las Editoriales
void llenaEditorial(){
    cboEditorial.addItem("A");
    cboEditorial.addItem("B");
    cboEditorial.addItem("C");
}
```

```
//Método que llena el control cboClase de las Clases especificadas
//en el caso
void llenaClase(){
    cboClase.addItem("Programacion");
    cboClase.addItem("Analisis");
    cboClase.addItem("Diseño");
}

//Métodos que obtienes los valores ingresados y seleccionados por el //usuario.
String getLibro(){
    return txtLibro.getText();
}

String getEditorial(){
    return String.valueOf(cboEditorial.getSelectedItem());
}

String getClase(){
    return String.valueOf(cboClase.getSelectedItem());
}

int getAño(){
```

El siguiente script muestra las instrucciones del botón btnRegistrar.

```
private void btnRegistrarActionPerformed(…) {
        try{
          //Creando el objeto de la clase Libro y enviando los valores
          objLi = new Libro(getLibro(),getEditorial(),getClase(),
                            getAño(),getPaginas(),getCosto());

         //Llena los datos obtenidos en las listas por medio de los
         //modelos
         llenaModelos();
        } catch(Exception ex){
            JOptionPane.showMessageDialog(null,
                       "Error en la Aplicacion " + ex.getMessage());
        }
    }
```

El siguiente script muestra las instrucciones del botón btnEstadisticas.

```
private void btnEstadisticasActionPerformed(…) {
try{
    moEstadisticas.clear();

    //Mostrar el numero total de Libros de Analisis de la editorial B
    moEstadisticas.addElement("Numero de Libro de Analisis de
                       la Editorial B es: "+objLi.getTAnalisis());

    //Determinar el libro con el año de edicion mas reciente
    int mayor=Integer.MIN_VALUE;
    int pos=0;
    for(int i=0;i<moLibro.getSize();i++){
       if (Integer.parseInt(moAño.elementAt(i).toString())>mayor){
```

```
            mayor=Integer.parseInt(moAño.elementAt(i).toString());
            pos=i;
         }
      }

      moEstadisticas.addElement("Libro con el año de edicion
                        mas reciente: "+moLibro.getElementAt(pos));

      //Determinar la editorial con el libro de menor pagina
      int menor=Integer.MAX_VALUE;
      for(int i=0;i<moLibro.getSize();i++){
         if (Integer.parseInt(moPaginas.elementAt(i).toString())<menor){
            menor=Integer.parseInt(moPaginas.elementAt(i).toString());
            pos=i;
         }
       }
          moEstadisticas.addElement("Editorial con libro de menor
                        pagina es: "+moEditorial.getElementAt(pos));

       //Determinar el libro con mayor costo
       double mayorCosto=Double.MIN_VALUE;
       for(int i=0;i<moLibro.getSize();i++){
        if (Double.parseDouble(moCosto.elementAt(i).toString())>
                                                          mayorCosto){
         mayorCosto=Double.parseDouble(moCosto.elementAt(i).toString());
         pos=i;
        }
       }
       moEstadisticas.addElement("Libro con mayor costo
                                 es: "+moLibro.getElementAt(pos));
      }catch(Exception ex){
         JOptionPane.showMessageDialog(null, "Error en la Aplicacion " +
                                       ex.getMessage());
      }
   }
```

CAP.

10

Arrays

CAPACIDAD:

- Comprenderá los conceptos básicos de los arreglos y podrá diferenciarlos por las dimensiones.
- Podrá implementar aplicaciones matriciales de una y dos dimensiones, logrando así manipular información masiva en una misma aplicación.

CONTENIDO:

10.1. INTRODUCCIÓN

Las variables que hemos usado en todas las aplicaciones hasta el momento, son de tipo simple, es decir, consiste de una sola caja de memoria y solo puede contener un valor cada vez.

Los tipos de datos vistos hasta ahora son entero (int), real (double, float) y son considerados como tipos de datos simples, puesto que solo se puede almacenar un valor a la vez, es decir, existe una relación de uno a uno entre la variable y el elemento que es capaz de almacenar. En cambio, un dato de tipo estructurado, como el Array (arreglo), puede almacenar más de un elemento a la vez, con la condición de que todos los elementos deben ser del mismo tipo.

Vea los siguientes casos:

1. Leer una lista de calificaciones de un aula de clases.
2. Encontrar la media aritmética de todas las calificaciones.
3. Listar qué alumnos tienen la calificación más alta.
4. Ordenar la lista de calificaciones en orden ascendente o descendente.

Ahora, si se quiere registrar 100 o 200 calificaciones sin usar arreglo tendría que declarar 100 o 200 variables diferentes como por ejemplo calificacion1, calificacion2, ..., calificacion200, de esta manera estaría separando 200 variables, es decir, 200 espacios de memoria diferentes para almacenar las calificaciones. Esto no sería la forma adecuada de registrar las calificaciones, ya que se usan muchas variables.

En Java tendría el siguiente script para declarar las calificaciones:

```
int calificacion1, calificacion1, calificacion1, …,calificacion200;
```

- Calculando la media aritmética:

```
ouble mediaArit = (calificacion1+calificacion2+….calificacion200)/200;
```

- Imprimir las calificaciones en un modelo.

```
DefaultListModel moCalificaciones = new DefaultListModel();
moCalificaciones.addElement(calificacion1);
moCalificaciones.addElement(calificacion2);
moCalificaciones.addElement(calificacion3);
…
moCalificaciones.addElement(calificacionN);
```

El equivalente al script anterior usando Arreglo sería de la siguiente forma:

```
    int calificacion[] = new int(200);
```

- Calculando la media aritmetica

```
double mediaArit;
for(int i=0;i<200;i++){
mediaArit += calificacion[i];
}
mediaArit = mediaArit/200;
```

- Imprimir las calificaciones en un modelo

```
DefaultListModel moCalificaciones = new DefaultListModel();

for(int i=0;i<200;i++){
moCalificaciones.addElement(calificacion[i]);
}
```

10.1.1. Ventajas y desventajas de usar arreglos

- Si se conoce la posición dentro del arreglo del elemento que quiere consultar, la consulta toma un tiempo constante.
- Se pueden utilizar, para implementar, otras estructuras de datos como pilas, colas, tablas hash, etc.
- El tamaño de un arreglo es fijo, por lo que si no se conoce de antemano el número máximo de elemento a almacenar pueden ocurrir problemas si el espacio reservado es menor del que se necesita.
- Insertar elementos de manera ordenada es muy lento.
- Buscar un elemento en un arreglo desordenado es muy lento.

10.2. Arreglos

Un arreglo es una colección de datos del mismo tipo, estos datos se almacenan en posiciones consecutivas de memoria y reciben un nombre de variable común, es decir, varios datos estarán controlados por una sola variable. Para hacer referencia a un determinado elemento del arreglo se debe utilizar el nombre del arreglo acompañado del índice, el cual especifica la posición relativa en que se encuentra el elemento.

Los arreglos se pueden dividir en:

- Unidimensionales, también llamados vectores.

0	1000.00
1	1500.00
2	1200.00
3	1050.00
4	2500.00
5	3000.00

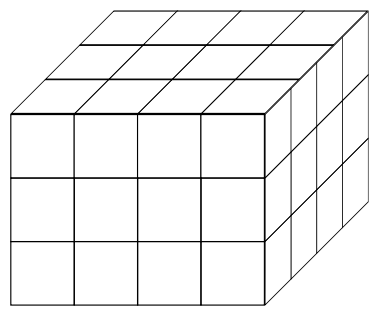
0	1	2	3	4	5
1000.00	1500.00	1200.00	1050.00	2500.00	3000.00

- Bidimensionales, también llamados matrices o simplemente tablas.

(0,0) 100	(0,1) 50	(0,2) 75
(1,0) 42	(1,1) 30	(1,2) 80
(2,0) 82	(2,1) 51	(2,2) 12

0,0	100
0,1	50
0,2	75
1,0	42
1,1	30
1,2	80
2,0	82
2,1	51
2,2	12

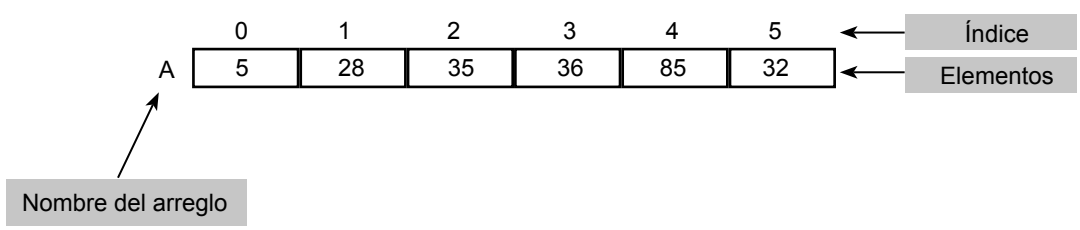
- Multidimensionales, se dice así cuando tiene de 3 a más dimensiones.



10.3. ARREGLO UNIDIMENSIONAL

Un arreglo unidimensional es un conjunto de elementos almacenados consecutivamente uno a continuación del otro, donde cada elemento conserva su propio espacio con el mismo nombre para todos los elementos. Se le dice unidimensional porque tiene un solo índice.

Forma Gráfica:



Donde:

- El nombre del arreglo es asignado dependiendo de los elementos que contiene el arreglo. Por ejemplo, un arreglo de números (número) o un arreglo de edades (edad).
- El índice de un arreglo tiene como punto de inicio la posición cero y como última posición la cantidad total de elementos menos uno (N-1). Para recorrer por cada uno de los elementos de un arreglo se necesita una estructura repetitiva como for o while.
- Cada elemento es asignado a una posición independiente del arreglo.

  Veamos:

  A[0] = 5
  A[1] = 28
  A[2] = 35
  A[3] = 36
  A[4] = 85
  A[5] = 32

10.3.1. Formato de declaración de un arreglo unidimensional

```
tipodeDatos nombreArreglo[];

tipoDato[] nombreArreglo;
```

Donde:

- TipodeDatos: es el tipo de datos especificados para los valores que se almacenarán en el arreglo unidimensional. Hay que tener en cuenta que todos los elementos serán del mismo tipo.
- nombreArreglo: es el nombre asignado al arreglo.

Vea los siguientes casos de declaración de arreglo unidimensional:

Declarar el arreglo nombres del empleado, número de hijos y pago mensual para los registros de N empleados en una empresa.

```
Forma 1: tipoDato nombreArreglo[];

String nombres[];
int nHijos[];
double pagoMensual[];

Forma 2:  tipoDato[] nombreArreglo;

String[] nombres;
int[] nHijos;
double[] pagoMensual;
```

FORMATO DE CREACIÓN DEL OBJETO DE TIPO ARREGLO

```
nombreArreglo = new tipoDatos[tamaño]
```

Donde:

- NombreArreglo: se especifica el nombre del arreglo ya declarado.
- tipoDatos: es el mismo tipo de datos especificado en la declaración.
- tamaño: se especifica el total de elementos que puede contener el arreglo, considere que aquí no se especifica el máximo tamaño sino el total de elementos.

Vea el siguiente caso:

Crear los objetos de tipo arreglo para los nombres del empleado, número de hijos y pago mensual de 100 empleados en una empresa.

```
nombres = new String[100];
nHijos = new int[100];
pagoMensual  = new double[100];
```

FORMATO PARA LA DECLARACIÓN Y CREACIÓN DEL OBJETO DE TIPO ARREGLO AL MISMO TIEMPO.

```
tipoDatos nombreArreglo[] = new tipoDatos[tamaño]
```

Donde:

- NombreArreglo: se especifica el nombre del arreglo a declarar y crear.
- tipoDatos: es el tipo de datos especificado en la declaración y creación del objeto.
- tamaño: se especifica el total de elementos que puede contener el arreglo.

Vea el siguiente caso:

Declarar y crear los objetos de tipo arreglo para los nombres del empleado, número de hijos y pago mensual de 100 empleados en una empresa.

```
String nombres[] = new String[100];
int nHijos[] = new int[100];
double pagoMensual[] = new double[100];
```

CASO DESARROLLADO 1: LISTADO DE NÚMEROS BÁSICOS

Implemente una aplicación que permita registrar un arreglo unidimensional basico de 6 elementos numericos enteros y que al final determine la suma, el promedio, la longitud y el mayor elemento registrado en el arreglo. Adicionalmente implemente un método que permita ordenar los elementos del arreglo en forma ascedente.

GUI Propuesto:

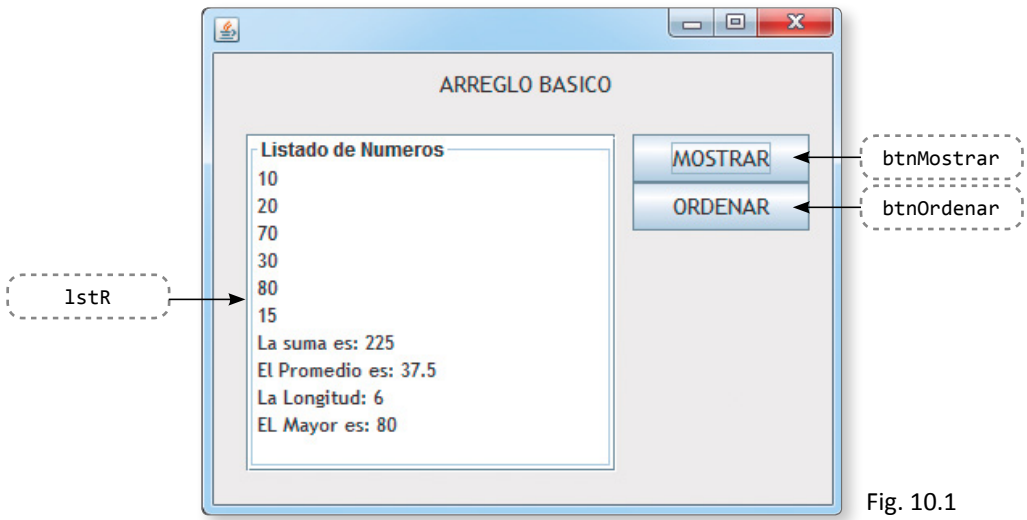
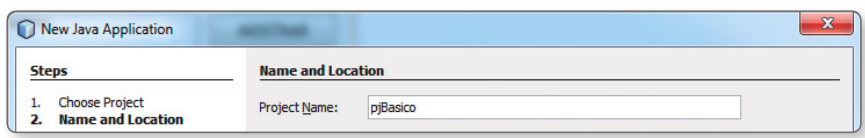


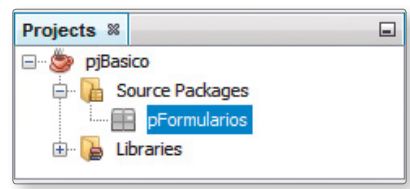
Fig. 10.1

Debe considerar:

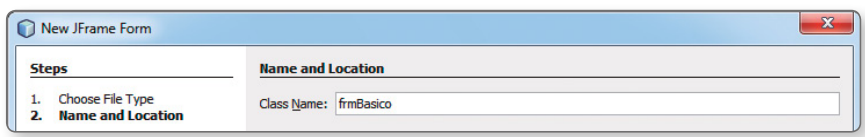
- Crear un nuevo proyecto en NetBeans llamado pjBasico.



- Agregar el paquete pFormularios al proyecto pjBasico.



- Agregar la clase frmBasico al paquete pFormularios, para esto presione clic derecho sobre el paquete pFormularios > New > JFrame Form:



- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 10.1, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 10.2.

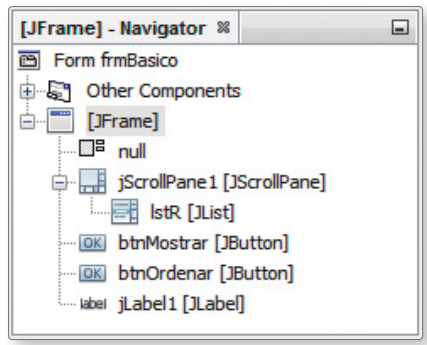


Fig. 10.2

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne las siguientes propiedades a los controles:

btnMostrar	**Text**	MOSTRAR
btnOrdenar	**Text**	ORDENAR
lstR	**Border**	TitledBorder=Listado de Numeros
	Model	Dejar vacío

El siguiente script muestra las librerías y la declaración de las variables globales de la aplicación.

```
package pFormularios;
import javax.swing.DefaultListModel;

public class frmBasico extends javax.swing.JFrame {
    //Declaracion del arreglo unidimensional
    int a[]=new int[6];

    public frmBasico() {
        initComponents();
    }
```

Se declara como variable global a[] de tipo entero que representa al arreglo, este tiene una capacidad de 6 elementos numéricos.

El siguiente script muestra los métodos implementados en la aplicación.

```
void ordenaAscendente(){
        int temp;
        for(int i=0;i<a.length;i++)
            for(int j=0;j<a.length;j++)
                if (a[i]<a[j]){
                    temp = a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }

        DefaultListModel moNumeros=new DefaultListModel();
        for(int i=0;i<=5;i++){
            moNumeros.addElement(a[i]);
        }
        lstR.setModel(moNumeros);
    }


    int determinaMayor(){
        int mayor=Integer.MIN_VALUE;
       for(int i=0;i<a.length;i++){
           if(a[i]>mayor)
               mayor=a[i];
       }
        return mayor;
    }
    double calculaPromedio(int suma){
        return (suma*1.0/a.length);
    }

    int calculaSuma(){
        int s=0;
        for(int i=0;i<=5;i++){
            s+=a[i];
        }
        return s;
    }
```

El método ordenaAscendente permite ordenador los elementos contenidos en el arreglo usando la técnica de ordenamiento por burbuja. El resultado del ordenamiento se obtiene en el mismo método.

El método determinaMayor se encarga de encontrar el mayor elemento registrado en el arreglo.

El método calculaPromedio permite determinar el promedio de los números registrados en el arreglo, para esto se necesita como parámetro el valor de la suma que lo realiza el método calculaSuma.

El método calculaSuma permite calcular la suma de los elementos contenidos en el arreglo.

El siguiente script muestra las instrucciones del botón btnMostrar.

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
        //1
        a[0]=10;
        a[1]=20;
        a[2]=70;
        a[3]=30;
        a[4]=80;
        a[5]=15;
```

```
        //2
        int suma=calculaSuma();
        double promedio=calculaPromedio(suma);
        int mayor=determinaMayor();

        //3
        DefaultListModel moNumeros=new DefaultListModel();
        for(int i=0;i<=5;i++){
            moNumeros.addElement(a[i]);
        }
        moNumeros.addElement("La suma es: "+suma);
        moNumeros.addElement("El Promedio es: "+promedio);
        moNumeros.addElement("La Longitud: "+a.length);
        moNumeros.addElement("EL Mayor es: "+mayor);
        lstR.setModel(moNumeros);
    }
```

En el punto uno se asigna directamente valores numéricos enteros a las posiciones del arreglo, dichas posiciones obedecen a la declaración de la variable global a[] la cual se le asigno el valor 6 como tope, es decir los valores tendrían la posición desde el índice 0 hasta el 5.

En el punto dos se asignan a variables locales los valores resultantes de los métodos como calculaSuma, calculaPromedio y determinaMayor.

En el punto tres se hace un recorrido por todos los elementos del arreglo y es enviado al modelo correspondiente, para lograr dicho recorrido se usa la estructura repetitiva for. Luego se envían los valores resultantes de la aplicación como la suma, el promedio, la cantidad de elementos registrados (a.length) y el mayor elemento del arreglo.

CASO DESARROLLADO 2: LISTADO DE NÚMEROS USANDO CLASE

Implemente una aplicación que permita registrar un arreglo unidimensional básico de 6 elementos numéricos enteros y que al final determine la suma, el promedio, la longitud y el mayor elemento registrado en el arreglo. Adicionalmente, implemente un método que permita ordenar los elementos del arreglo en forma ascendente.

Debe implementar la clase ArregloUnidimensional que permita controlar todos los métodos de la aplicación. Considere los siguientes aspectos:

- *Crear un arreglo para los números dentro de la clase arregloNumeros.*

Implemente los siguientes métodos dentro de la clase arregloNumeros:

- *El* **método constructor** *se encargara de crear el arreglo numérico de 6 números e inicializar los valores del arreglo con los siguientes números: 10, 20, 70, 30, 80 y 15.*
- *El* **método tamaño** *devuelve el número de elementos registrados en el arreglo.*
- *El* **método ordenaAscendente** *se encarga de ordenar los números registrados en el arreglo en forma ascendente.*

- *El* **método determinaMayor** *se encarga de obtener el mayor elemento del arreglo.*
- *El* **método calculaPromedio** *se encarga de promediar los números del arreglo.*
- *El* **método calculaSuma** *se encarga de sumar todos los elementos del arreglo.*
- *El* **método devuelveValor** *se encarga de devolver el elemento contenido en el arreglo de uno en uno.*

GUI Propuesto:

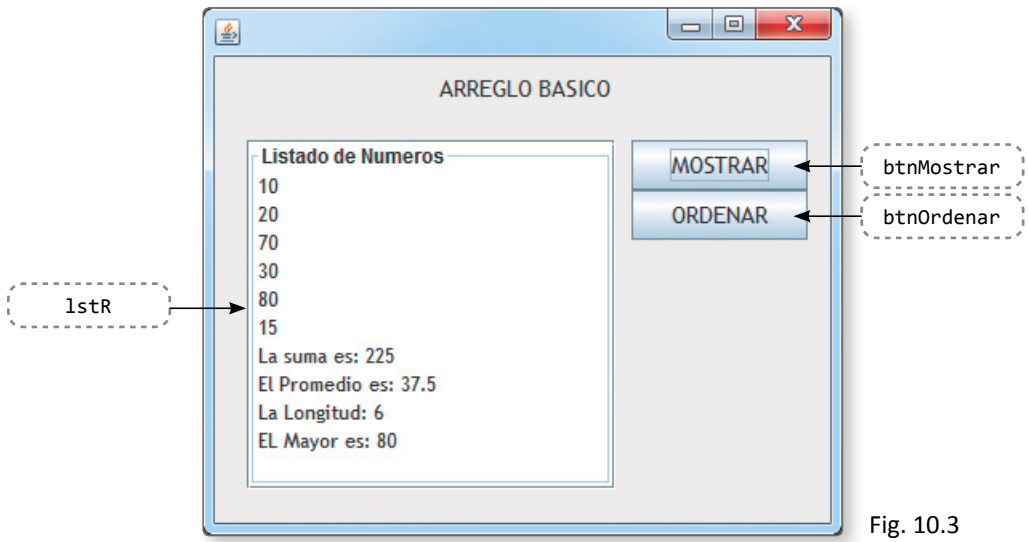
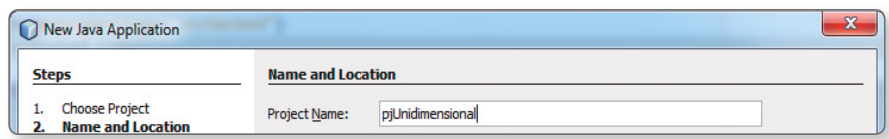


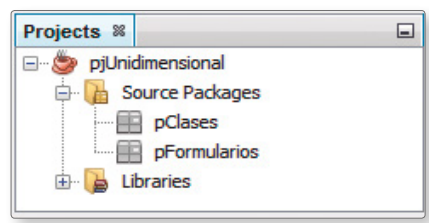
Fig. 10.3

Debe considerar:

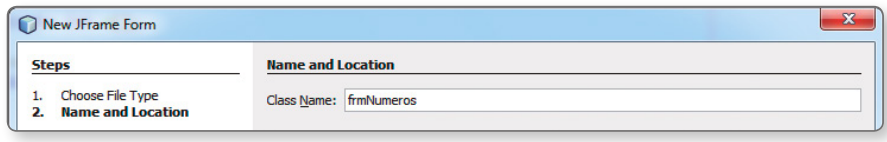
- Crear un nuevo proyecto en NetBeans llamado pjUnidimensional:



- Agregar el paquete pFormularios al proyecto pjUnidimensional.



- Agregar la clase frmNumeros al paquete pFormularios, para esto presione clic derecho sobre el paquete pFormularios > New > JFrame Form.



- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 10.3, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**…
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos, para eso visualice el panel Navigator; debe mostrarse como la Fig. 10.4.

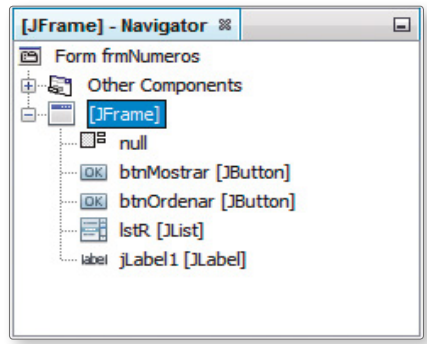


Fig. 10.4

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.
- Asigne la siguientes propiedades a los controles:

btnMostrar	**Text**	MOSTRAR
btnOrdenar	**Text**	ORDENAR
lstR	**Border**	TitledBorder=Listado de Numeros
	Model	Dejar vacío

El siguiente script muestra el contenido de la clase arregloNumeros.

```
package pClases;

public class arregloNumeros {
    //Atributos
    private int a[];

    //metodo constructor
    public arregloNumeros(){
        a=new int[6];
        a[0]=10;
        a[1]=20;
        a[2]=70;
        a[3]=30;
        a[4]=80;
        a[5]=15;
    }
```

```
    public int tamaño(){
        return a.length;
    }

    public void ordenaAscendente(){
        int temp;
        for(int i=0;i<tamaño();i++)
            for(int j=0;j<tamaño();j++)
                if (a[i]<a[j]){
                    temp = a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }
    }

    public int determinaMayor(){
        int mayor=Integer.MIN_VALUE;
       for(int i=0;i<tamaño();i++){
           if(a[i]>mayor)
               mayor=a[i];
       }
        return mayor;
    }

    public double calculaPromedio(int suma){
        return (suma*1.0/tamaño());
    }

    public int calculaSuma(){
        int s=0;
        for(int i=0;i<tamaño();i++){
            s+=a[i];
        }
        return s;
    }

    //Devuelve el elemento del arreglo segun la posicion
    public int devuelveValor(int pos){
        return a[pos];
    }
}
```

Se declara como atributo privado a[] para el arreglo de tipo entero, se implementa el método constructor en la cual se asigna como capacidad máxima del arreglo 6 elementos numéricos y se le asigna los valores. Hay que tener en cuenta que el trabajo del método constructor es inicializar los valores; por tanto, al iniciar la aplicación e instanciar a la clase arregloNumeros; el arreglo a[] ya debe tener asignado los valores numéricos.

El método tamaño determina el total de elementos que tiene asignada hasta el momento el arreglo, es decir, devolverá el valor 6 puesto que dentro del método constructor se asignó 6 valores a los espacios establecidos en el arreglo.

El método ordenaAscendente permite ordenar los elementos del arreglo en forma ascendente, aplicando el método de la burbuja.

El método determinaMayor permite devolver el mayor elemento registrado en el arreglo.

El método calculaPromedio permite calcular el promedio de los valores el arreglo. para esto necesita como parámetro el valor de la suma.

El método calculaSuma determina la suma de los valores dentro del arreglo.

Finalmente, el método devuelveValor permite devolver elemento por elemento del arreglo dependiendo de la posición de este, para lo cual se necesita como parámetro la posición del elemento que se desea obtener, considere que si este método es invocado dentro de una estructura repetitiva podrá obtener todos los elementos al mismo tiempo, ya que el solo invocar al método devolverá un solo valor.

El siguiente script muestra las librerías y declaración de las variables globales de la aplicación.

```
package pFormularios;
import javax.swing.DefaultListModel;
import pClases.arregloNumeros;

public class frmNumeros extends javax.swing.JFrame {

    arregloNumeros aNum=new arregloNumeros();

    public frmNumeros() {
        initComponents();
    }
```

La variable de clase aNum permite tener acceso a todos los métodos y atributos públicos de la clase arregloNumeros.

El siguiente script muestra las instrucciones del botón btnMostrar.

```
private void btnMostrarActionPerformed(java.awt.event.ActionEvent evt) {
        int suma=aNum.calculaSuma();
        double promedio=aNum.calculaPromedio(suma);
        int mayor=aNum.determinaMayor();

        //enviarlo a la lista
        DefaultListModel moNumeros=new DefaultListModel();
        for(int i=0;i<=5;i++){
            moNumeros.addElement(aNum.devuelveValor(i));
        }
        moNumeros.addElement("La suma es: "+suma);
        moNumeros.addElement("El Promedio es: "+promedio);
        moNumeros.addElement("La Longitud: "+aNum.tamaño());
        moNumeros.addElement("EL Mayor es: "+mayor);
        lstR.setModel(moNumeros);
    }
```

Empieza el botón de mostrar por obtener los valores de la suma, promedio y el mayor elemento por medio del objeto aNum que es la instancia de la clase arregloNumeros.

Luego se envían los valores del arreglo al control lstR; para esto debe recorrer por todos los elementos del arreglo, por eso implementó la estructura repetitiva for cuyo valor inicial es cero y valor tope de repeticiones es menor o igual a 5; pero también se pudo implementar de la siguiente forma:

```
for (int i=0;i<aNum.tamaño();i++)
```

Recuerde que dentro de la clase arregloNumeros se implementó el método devuelveValor que tenía la misión de devolver uno a uno los elementos del arreglo, dependiendo del valor del índice, en este caso la variable i mediante la estructura for tiene todas las posiciones del arreglo; por tanto, i se vuelve el parámetro para la obtención de los valores.

El siguiente script se muestra las instrucciones del botón btnOrdenar.

```
private void btnOrdenarActionPerformed(java.awt.event.ActionEvent evt) {
aNum.ordenaAscendente();

//enviarlo a la lista
DefaultListModel moNumeros=new DefaultListModel();
for(int i=0;i<=5;i++){
     moNumeros.addElement(aNum.devuelveValor(i));
}
lstR.setModel(moNumeros);
}
```

Caso desarrollado 3: Certamen de belleza

Implemente una aplicación que permita registrar un número de candidatas de un certamen de belleza, el cual necesita determinar la ganadora del concurso y por cuántos puntos lo logró. Hay que tener en cuenta que son 3 los criterios del jurado que al final se reflejan en los puntajes, este puntaje tiene un rango de 0 a 100 puntos.

Considere los siguientes aspectos:

- *Crear 2 arreglos unidimensionales uno para el nombre de la candidata y otro para el promedio de los puntajes dentro de una clase llamada arreglo.*
- *El* **método constructor** *se encargará de crear los arreglos e inicializar la posición del primer elemento de los arreglos en cero.*
- *El* **método registraCandidata** *que permitirá registrar los datos y el promedio de una determinada candidata.*
- *El* **método devuelveNombre** *que permitirá devolver el nombre de la candidata.*
- *El* **método devuelvePromedio** *que permitirá devolver el promedio obtenido por una determinada candidata.*
- *El* **método totalCantidatas** *se encarga en devolver el total de candidatas registradas en la aplicación.*
- *El* **método puntajeAlto** *se encarga de devolver el puntaje más alto obtenido por una candidata.*
- *El* **método ganadora** *se encarga de devolver quién es la ganadora del certamen.*

GUI Propuesto:

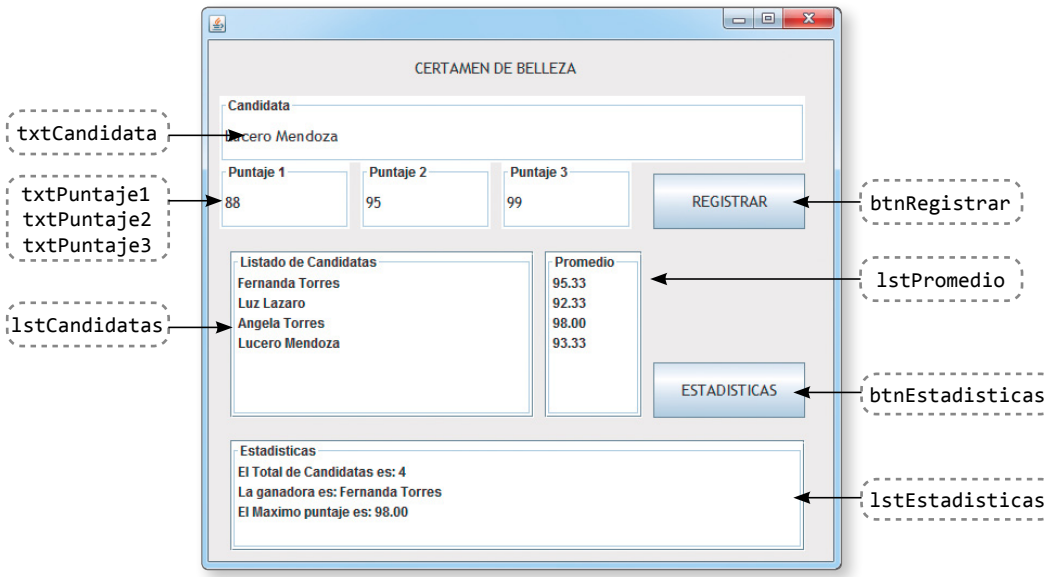


Fig. 10.5

Debe considerar:

- Crear un nuevo proyecto en NetBeans llamado pjCertamen.
- Agregar el paquete pFormularios al proyecto pjCertamen.
- Agregar la clase frmNumeros al paquete pFormularios.
- Asigne un nombre a cada uno de los controles como se muestra en la Fig. 10.5, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**…
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos, para eso visualice el panel Navigator; debe mostrarse como la Fig. 10.6.

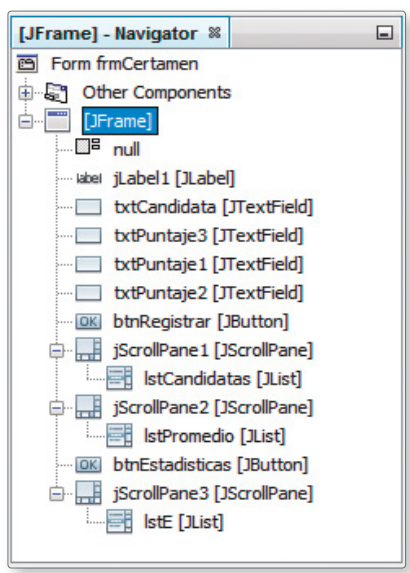


Fig. 10.6

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.

El siguiente script muestra el contenido de la clase arreglo.

```
package pClases;
public class arreglo {

    //Atributos de clase
    private String nombre[];
    private double promedio[];
    private int pos=0;
    private int posicion;

    //constructor
    public arreglo(){
        nombre=new String[100];
        promedio=new double[100];
        pos=0;
    }

    public void registraCandidata(String nombre,double promedio){
        this.nombre[pos]=nombre;
        this.promedio[pos]=promedio;
        pos++;
    }

    public String devuelveNombre(int pos){
        return nombre[pos];
    }
    public double devuelvePromedio(int pos){
        return promedio[pos];
    }

    public int totalCandidatas(){
        return pos;
    }

    public double puntajeAlto(){
        double mayor=Double.MIN_VALUE;
        for(int i=0;i<totalCandidatas();i++){
            if (promedio[i]>mayor){
                mayor=promedio[i];
                posicion=i;
            }
        }
        return mayor;
    }

    public String ganadora(){
        return nombre[posicion];
    }

}
```

Se declaran los atributos de la clase nombre y promedio de tipo Arreglo. Pos es el atributo encargado de posicionarse dentro de los índices del arreglo y qué posición tendrá la misión de obtener la posición de la candidata ganadora. Es decir, al momento de encontrar a la ganadora ubicar en qué posición del

arreglo se encuentra este valor y se guardará en la variable posición, ya que otro método solicitará dicho valor para determinar el nombre de la ganadora.

El método constructor crea los objetos de tipo Arreglo con una capacidad de 100 elementos, es decir, tendrá como máximo 100 candidatas que se podrían registrar en la aplicación. Además, se inicializa la variable pos en cero ya que es el punto de inicio de un arreglo.

El método registraCandidata permite enviar los nombres y el promedio de una determinada candidata en la posición actual de los arreglos. Al final se aumenta en uno la variable pos ya que así se posicionará en un valor más el índice del arreglo para la siguiente candidata.

El método devuelveNombre y devuelvePromedio permiten devolver los valores registrados según la posición del índice.
El método totalCandidatas devuelve el total de participantes registradas hasta el momento.

El método puntajeAlto permite devolver cuál es el puntaje más alto y a la vez registra en qué posición del arreglo encontró dicho valor y lo guardará dentro de la variable posición.

El método ganadora permite devolver el nombre de la candidata ganadora según la posición obtenida en el método puntajeAlto.

El siguiente script muestra las librerías y declaración de las variables globales de la aplicación.

```
package pFormularios;

import java.text.DecimalFormat;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;
import pClases.arreglo;

public class frmCertamen extends javax.swing.JFrame {

    arreglo aCan = new arreglo();

    DefaultListModel moCandidatas=new DefaultListModel();
    DefaultListModel moPromedio=new DefaultListModel();
    DefaultListModel moEstadisticas=new DefaultListModel();

    DecimalFormat df;

    public frmCertamen() {
        initComponents();
        df=new DecimalFormat("#0.00");
    }
```

La librería DecimalFormat la usará para redondear los puntajes obtenidos. También se declara en forma global el objeto aCan de la clase arreglo. Luego se definen los modelos a usar en la aplicación y finalmente se declara df como objeto de la clase DecimalFormat.

En el método constructor de la clase frmCertamen se declara el objeto df como instancia de la clase DecimalFormat con un formato de solo 2 decimales.

El siguiente script muestra los metodos de la clase frmCertamen.

```
String getnombre(){
        return txtCandidata.getText();
    }

double getPuntaje1(){
        return Double.parseDouble(txtPuntaje1.getText());
    }

double getPuntaje2(){
        return Double.parseDouble(txtPuntaje2.getText());
    }

double getPuntaje3(){
        return Double.parseDouble(txtPuntaje3.getText());
    }

double calculaPromedio(){
        return (getPuntaje1()+getPuntaje2()+getPuntaje3())/3.0;
    }

void imprimir(){
        moCandidatas.clear();
        moPromedio.clear();
        moEstadisticas.clear();

        for(int i=0;i<aCan.totalCandidatas();i++){
           moCandidatas.addElement(aCan.devuelveNombre(i));
           moPromedio.addElement(df.format(aCan.devuelvePromedio(i)));
        }
        lstCandidatas.setModel(moCandidatas);
        lstPromedio.setModel(moPromedio);
    }

void limpiar(){
        txtCandidata.setText("");
        txtPuntaje1.setText("");
        txtPuntaje2.setText("");
        txtPuntaje3.setText("");
        txtCandidata.requestFocus();
    }

String valida(){
        if (txtCandidata.getText().equals("")){
            txtCandidata.requestFocus();
            return "Nombre de la candidata";
        } else if(txtPuntaje1.getText().equals("")||
                  Double.parseDouble(txtPuntaje1.getText())<0 ||
                  Double.parseDouble(txtPuntaje1.getText())>100){
            txtPuntaje1.setText("");
            txtPuntaje1.requestFocus();
            return "Puntaje 1";
        }else if(txtPuntaje2.getText().equals("")||
                  Double.parseDouble(txtPuntaje2.getText())<0 ||
                  Double.parseDouble(txtPuntaje2.getText())>100){
            txtPuntaje2.setText("");
            txtPuntaje2.requestFocus();
            return "Puntaje 2";
        }else if(txtPuntaje3.getText().equals("")||
                  Double.parseDouble(txtPuntaje3.getText())<0 ||
                  Double.parseDouble(txtPuntaje3.getText())>100){
            txtPuntaje3.setText("");
            txtPuntaje3.requestFocus();
```

```
            return "Puntaje 3";
        } else
            return "";

}
```

El siguiente script muestra las instrucciones del botón btnRegistrar:

```
private void btnRegistrarActionPerformed(…) {
   try{
       if (valida().equals("")){
          String candidata = getnombre();
          double puntaje1=getPuntaje1();
          double puntaje2=getPuntaje2();
          double puntaje3=getPuntaje3();
          aCan.registraCandidata(candidata, calculaPromedio());
          imprimir();
          limpiar();
          }else
          JOptionPane.showMessageDialog(null,"Error en "+valida());
    } catch(Exception ex){
        JOptionPane.showMessageDialog(null,"Error en la Aplicacion");
  }
}
```

Dentro del botón btnRegistrar se implementa un Try…Catch para prevenir algunas excepciones que se pudiera ocasionar en la aplicación. Luego se valida la entrada de los valores a la aplicación, si todo es correcto se capturan dichos valores y son enviados a la clase por medio del objeto aCan. En caso el método valida devuelva una respuesta esta será impresa.

El siguiente script muestra las instrucciones del botón btnEstadisticas:

```
private void btnEstadisticasActionPerformed(…) {
  int total=aCan.totalCandidatas();
  moEstadisticas.clear();
  moEstadisticas.addElement("El Total de Candidatas es: "+total);
  moEstadisticas.addElement("La ganadora es: "+aCan.ganadora());
  moEstadisticas.addElement("El Maximo puntaje es:"+
                            df.format(aCan.puntajeAlto()));
  lstE.setModel(moEstadisticas);
}
```

10.4. ARREGLO BIDIMENSIONAL

Es considerado un vector de vectores o un conjunto de elementos, todos del mismo tipo, en el cual el orden de los componentes es significativo; por lo tanto, necesita 2 índices, ahí radica su principal característica.

Forma Gráfica:

(0,0) 100	(0,1) 50	(0,2) 75
(1,0) 42	(1,1) 30	(1,2) 80
(2,0) 82	(2,1) 51	(2,2) 12

Los elementos se registran en una posición de doble índice donde el primer índice es la posición de la fila, mientras el segundo indica la posición de la columna; si esto es correcto para poder registrar sobre ella se necesitará una estructura de repeticion doble. Considere que internamente el arreglo bidimensional trabaja de la siguiente forma:

0,0	100
0,1	50
0,2	75
1,0	42
1,1	30
1,2	80
2,0	82
2,1	51
2,2	12

Es decir, se visualiza como un arreglo unidimensional de valores, pero aun se mantiene el doble índice, factor indicativo del arreglo bidimensional.

10.4.1. Formato de declaración de un arreglo Bidimensional

```
tipoDato nombreArreglo[][];

tipoDato[][] nombreArreglo;
```

Donde:

- TipodeDatos: es el tipo de datos especificados para los valores que se almacenarán en el arreglo bidimensional. Hay que tener en cuenta que todos los elementos serán del mismo tipo.
- nombreArreglo: es el nombre asignado al arreglo.

Tenga en cuenta que a diferencia del arreglo unidimensional solo se coloca un corchete en el bidimensional, se asignan dos bloques de corchetes el primero para las filas y el segundo para las columnas.

Vea los siguientes casos de declaración de arreglo unidimensional:

Declarar el arreglo bidimensional numeros de tipo entero y pagos de tipo double.

```
Forma 1: tipoDato nombreArreglo[][];
int numeros[][];
double pagos[][];

Forma 2:  tipoDato[][] nombreArreglo;
int[][] numeros;
double[][] pagos;
```

Formato de declaración de un arreglo Bidimensional

```
nombreArreglo =  new tipoDato[tamañoF][tamañoC]
```

Donde:

- TipodeDatos: es el tipo de datos especificado en la declaración del arreglo.
- nombreArreglo: es el nombre asignado al arreglo.
- tamañoF: determina el máximo tamaño en filas declarado por el usuario.
- tamañoC: determinar el máximo tamaño en columnas declarado por el usuario.

Crear el arreglo bidimensional números de tipo entero y pagos de tipo double ya declarados.

```
-    numeros = new int[10][10];
-    pagos  = new double[10][10];
```

10.4.2. Operaciones sobre un arreglo Bidimensional

- **Ingreso de datos**: para este caso implementaremos un script que permita llenar un arreglo bidimensional de 3x3 de números enteros.

```
for(int i=0;i<3;i++)
      for(int j=0;j<3;j++)
   N[i][j]=Integer.parseInt(JOptionPane.
                            showInputDialog(this, "Ingrese un numero"))
```

- **Ingreso de datos aleatorios a una matriz:** para este caso implementaremos un script que permita llenar un arreglo bidimensional de 3x3 con números enteros de dos cifras en forma aleatoria de números enteros.

```
for(int i=0;i<3;i++)
     for(int j=0;j<3;j++)
        N[i][j]=(int)((99-10+1)*Math.random()+10);
```

- **Recorrer por los datos**: para este caso enviaremos la matriz N[][] al modelo moNumeros.

```
for(int i=0;i<3;i++)
     for(int j=0;j<3;j++)
            moNumeros.addElement(N[i][j]);
```

- **Operar sobre una matriz**: para este caso acumularemos los valores contenidos en la matriz N.

```
int s=0;
for(int i=0;i<3;i++)
     for(int j=0;j<3;j++)
            s+=N[i][j]);
```

Caso desarrollado 4: Matriz de números enteros

Aplicación que genera un arreglo bidimensional de números enteros de 2 cifras en forma aleatoria de acuerdo a una cantidad de filas y columnas ingresadas por el usuario como máximo de tamaño será de 3x3.

Declare como atributos privados la cantidad de filas (f), la cantidad de columnas (c) y un arreglo dibimensional N.

Crear la clase **ArregloBiNumeros** *con los siguientes métodos*

- *Método* **constructor** *que defina la cantidad de filas y columnas de la matriz y a la vez genere los números aleatorios de la matriz.*
- *Método que devuelva la cantidad de filas definidas para la matriz.*
- *Método que devuelva la cantidad de columnas definidas para la matriz.*
- *Método* **obtener** *que devuelva el elemento del arreglo matricial dependiendo de la posición de la fila y la columna.*
- *Método* **generar** *que permite generar automáticamente valores numéricos de 2 cifras.*
- *Método* **sumaFila** *que permita sumar una determinada fila ingresada por el usuario utilizar JOptionPane.showInputDialog().*
- *Método* **sumaColumna** *que permita sumar una determinada columna ingresada por el usuario utilizar JOptionPane.showInputDialog().*

GUI Propuesto:

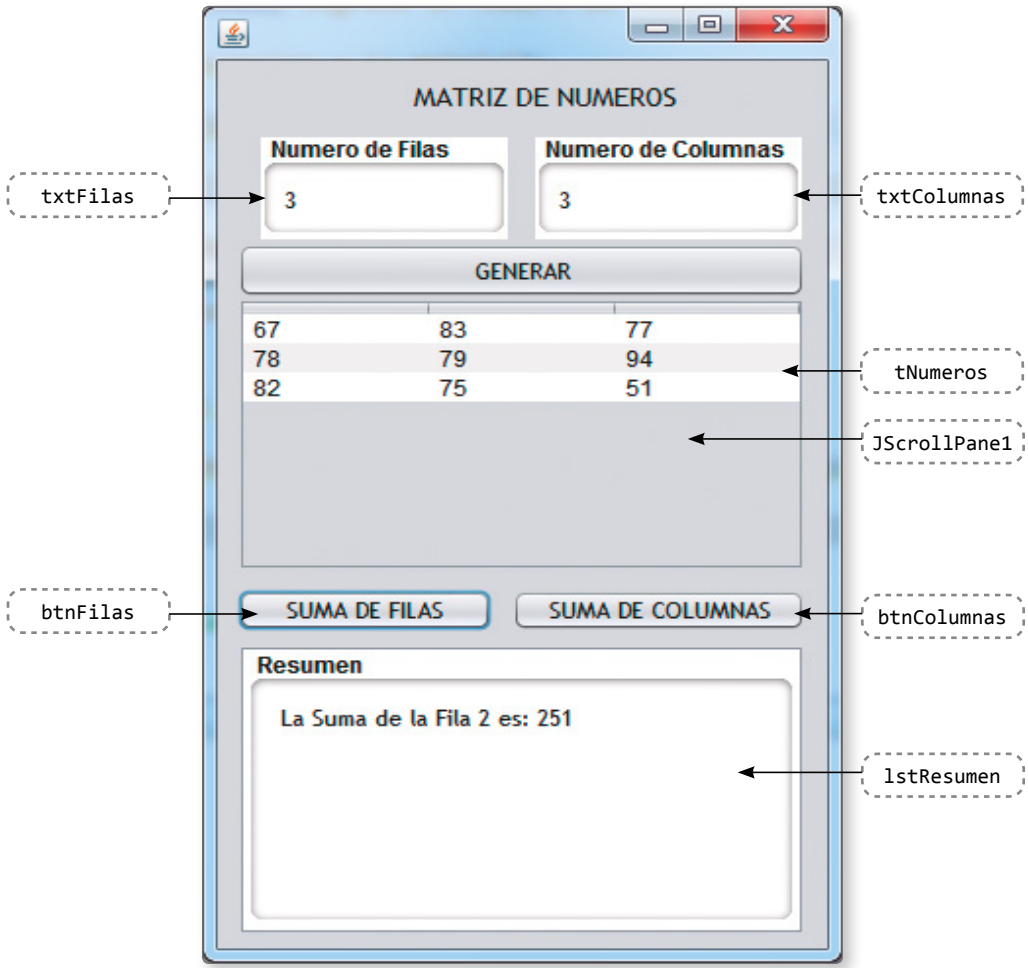
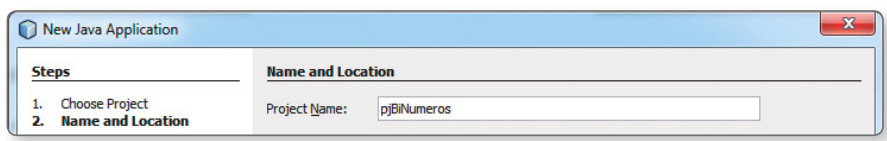


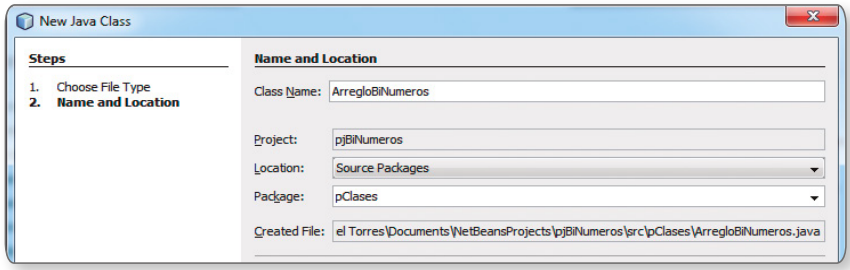
Fig. 10.7

Debe considerar:

- Crear un nuevo proyecto en NetBeans llamado pjBiNumeros.



- Agregar el paquete pFormularios al proyecto pjBiNumeros.
- Agregar el paquete pClases al proyecto pjBiNumeros.
- Agregar la clase ArregloBiNumeros al paquete pClases.



- Agregar la clase frmBiNumeros al paquete pFormularios.

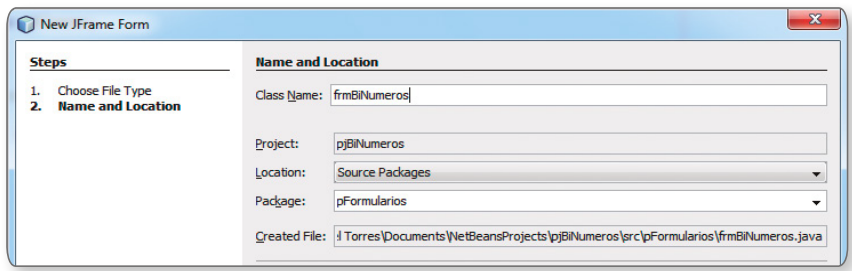


Fig. 10.8

- Asigne un nombre a cada uno de los controles como se muestra en la Fig. 10.7, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**…
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos, para eso visualice el panel Navigator; debe mostrarse como la Fig. 10.9.

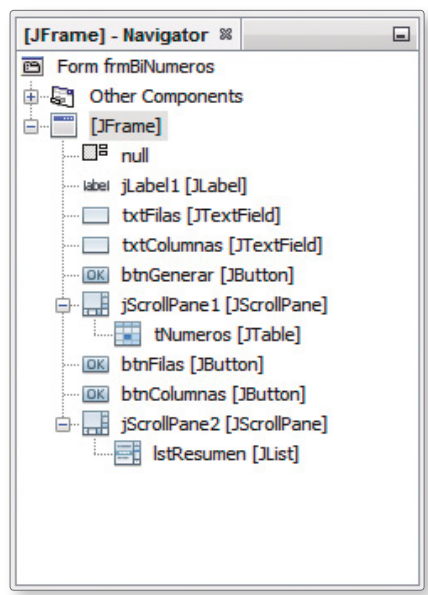


Fig. 10.9

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.

El siguiente script muestra el contenido de la clase ArregloBiNumeros.

```
package pClases;
public class ArregloBiNumeros {
    private int f,c;
    private int n[][];

    //1
    public ArregloBiNumeros(int f, int c){
        this.f=f;
        this.c=c;
        n=new int[f][c];
        generar();
    }

    //2
    public int getFilas(){
        return f;
    }
    public int getColumnas(){
        return c;
    }

    //3
    public int obtener(int posF, int posC){
        return n[posF][posC];
    }

    //4
    private void generar(){
        for(int i=0;i<f;i++)
            for(int j=0;j<c;j++)
                n[i][j]=(int)((99-10+1)*Math.random()+10);
    }

    //5
    public int sumaFila(int f){
        int s=0;
        int i=f-1;
            for(int j=0;j<c;j++)
                s+=n[i][j];
        return s;
    }

    //6
    public int sumaColumna(int c){
        int s=0;
        int j=c-1;
            for(int i=0;i<f;i++)
                s+=n[i][j];
        return s;
    }
}
```

En la clase ArregloBiNumeros se declaran las variables globales f para las filas, c para las columnas y n[][] para la matriz bidimensional.

En el punto uno se implementa el método constructor de la clase ArregloBiNumeros que tiene como parámetros a f y c que serán enviados desde el formulario con valores para la construcción de la matriz. Dentro del método constructor se asignan dichos valores a los atributos privados f y c respectivamente. Luego se crea el objeto n con la capacidad en filas y columnas de f y c que se reciben como parámetros de exterior. Finalmente, se invoca al método generar que permitirá llenar la matriz con elementos aleatorios.

En el punto dos se implementa el método getFilas y getColumnas que permitirá devolver el total de filas y columnas implementadas en el arreglo bidimensional.

En el punto tres se implementa el método obtener que permitirá devolver los valores contenidos en el arreglo bidimensional, esto dependerá de las posición de la fila y columna que serán obtenidas por el parámetro del método, en el formulario este método tendrá que ser invocado dentro de una estructura repetitiva donde posF representa a i de filas y posC representa a j de columnas.

En el punto cuatro se implementa el método generar que permite asignar a la matriz valores numéricos en forma aleatoria, para esto se usa la función random de la clase Math.

En el punto cinco se implementa el método sumaFila que permite sumar una de las filas del arreglo bidimensional, la fila considerada para la suma dependerá del usuario y este valor sería recibido como parámetro (f) por el método sumaFila.

En el punto seis se implementa el método sumaColumna que permite sumar los valores de una determinada columna del arreglo bidimensional, la columna será evaluada según el valor del parámetro (c).

El siguiente script muestra las librerías y declaración de las variables globales de la clase frmBiNumeros.

```
package pFormularios;

import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;
import pClases.ArregloBiNumeros;

public class frmBiNumeros extends javax.swing.JFrame {

    DefaultListModel moRespuesta=new DefaultListModel();
    ArregloBiNumeros aN;

    public frmBiNumeros() {
        initComponents();
        lstResumen.setModel(moRespuesta);
    }
```

Dentro de la clase frmBiNumeros se declara como variables globales al objeto moRespuesta que tendrá la misión de asimilar la respuesta de la suma de filas o columnas en el control lstResumen. También se declara la variable a N de la clase ArregloBiNumeros. Finalmente, dentro del método constructor de la clase frmBiNumeros se asigna al control lstResumen el modelo moRespuesta.

El siguiente script muestra los métodos de la clase frmBiNumeros.

```
public int getFilas(){
    return Integer.parseInt(txtFilas.getText());
}

public int getColumnas(){
    return Integer.parseInt(txtColumnas.getText());
}

void listar(){
    limpiar();
    int x;
    for(int i=0;i<getFilas();i++){
       for(int j=0;j<getColumnas();j++){
          x=aN.obtener(i,j);
          tNumeros.setValueAt(x, i, j);
       }
    }
}

void limpiar(){
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            tNumeros.setValueAt("", i, j);
        }
    }
}

void mensaje(String s){
    JOptionPane.showMessageDialog(null, s);
}
```

El método getFilas y getColumnas permitirá obtener el número de filas y columnas que el usuario desea implementar.

El método listar permite enviar los valores obtenidos desde la clase ArregloBiNumeros hacia el control tNumeros, recuerde que tenemos un método llamado obtener desde la clase ArregloBiNumeros que permite devolver los valores del arreglo, cada elemento del arreglo bidimensional es asignado a una variable local x y este se asignará al control tNumeros mediante una estructura repetitiva.

El método limpiar permitirá limpiar los valores contenidos en el control tNumeros como el máximo número de filas y columnas es 3x3, la estructura for también debe recorrer a 3x3 para poder limpiar el control.

El método mensaje permite mostrar un cuadro de mensaje desde cualquier parte de la aplicación.

El siguiente script muestra las instrucciones del botón btnGenerar:

```
private void btnGenerarActionPerformed(java.awt.event.ActionEvent evt) {
        try{
        moRespuesta.clear();

        //Capturando los valores ingresados en el Formulario
        int filas=getFilas();
        int columnas=getColumnas();

        //Verificando la cantidad de filas y columnas
        if (filas>0 && columnas>0 && filas<=3 && columnas<=3){
            aN=new ArregloBiNumeros(filas, columnas);
            listar();
        }else
            JOptionPane.showMessageDialog(null,"Error de Datos");
    }catch(Exception ex){
        JOptionPane.showMessageDialog(null,
                                      "Error de Ingreso de Datos");
    }
}
```

El siguiente script muestra las instrucciones del botón btnFilas:

```
private void btnFilasActionPerformed(java.awt.event.ActionEvent evt) {
    int f=Integer.parseInt(JOptionPane.showInputDialog
                           (this,"Ingrese un numero de Fila: "));
  if (f>getFilas())
      mensaje("El numero de Filas es superior a la matriz");
      else
      moRespuesta.addElement("La Suma de la Fila "+f+"
                              es: "+aN.sumaFila(f));
}
```

Se declara la variable local f y se le asignará un número de filas que el usuario ingresará mediante un cuadro de diálogo de entrada. Luego se compara si dicho valor no sobrepasa el límite de la matriz; si es así se envía un mensaje al usuario de lo contrario se invoca al método sumaFila de la clase ArregloBiNumeros mediante el objeto An.

El siguiente script muestra las instrucciones del botón btnColumnas:

```
private void btnColumnasActionPerformed(java.awt.event.ActionEvent evt) {
        int c=Integer.parseInt(JOptionPane.showInputDialog
                           (this,"Ingrese un numero de Columna: "));

   if (c>getColumnas())
     mensaje("El numero de Columnas es superior a la matriz");
     else
     moRespuesta.addElement("La Suma de la Columna "+c+"
                             es: "+aN.sumaColumna(c));
}
```

Se declara la variable local c y se le asignará un número de columnas que el usuario ingresará mediante un cuadro de diálogo de entrada. Luego se compara si dicho valor no sobrepasa el límite de la matriz, si es así se envía un mensaje al usuario, de lo contrario se invoca al método sumaColumna de la clase ArregloBiNumeros mediante el objeto An.

CAP.

11

Vector de objetos y arrayList

CAPACIDAD:

- Reconocerá un vector de objetos.
- Empleará vectores de objetos en aplicaciones comerciales.

CONTENIDO:

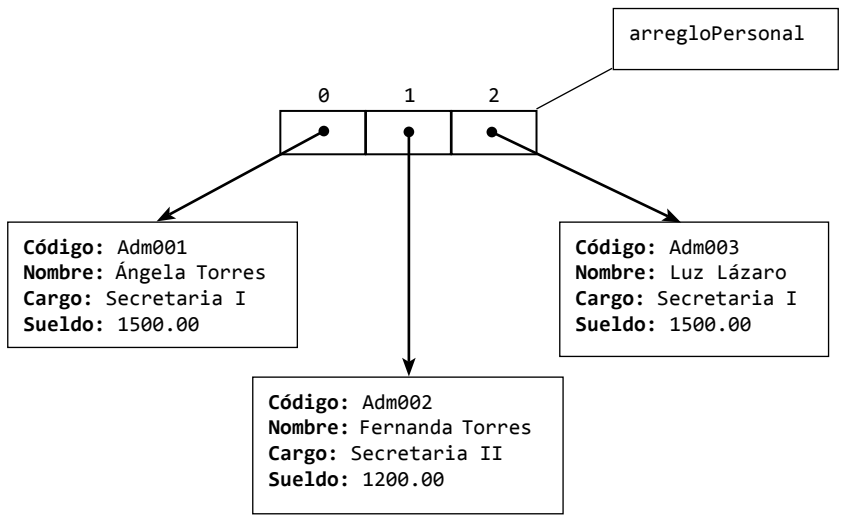
11.1. VECTOR DE OBJETOS

Los vectores de objetos son colecciones de tipos de datos definidos por el programador o que forman parte de la API JAVA y que son referenciados desde el arreglo.

Si tiene una clase llamada Personal puede crear muchos registros del personal, pero se perderían los valores; ahora si implementa un arreglo de la clase Personal estaría en la capacidad de almacenar muchos registros de personal dentro de un vector y no perdería la información anterior al último registro.

Las aplicaciones que se implementan en los casos desarrollados son de mantenimiento de registros, es decir, agregar, modificar, eliminar y listar; así como sucede cuando se trabaja con un motor de base de datos.

Forma Gráfica:



Debe considerar que la clase Personal contiene los parámetros de código, nombre, cargo y sueldo y que arregloPersonal es una arreglo unidimensional del tipo Personal, y que al registrar al primer personal será en la posición 0 del arregloPersonal.

11.1.1. Formato de declaración del vector de objetos

```
NombreClase[][] nombreVector;
```

Donde:

- NombreClase: es el nombre de la clase implementada previamente a la declaración del arreglo.
- nombreVector: es el nombre asignado al vector de objetos, tenga en cuenta que no debe ser igual al nombre de la clase.

11.1.2. Formato de creación del vector de objetos

```
nombreVector = new NombreClase[Numero];
```

Donde:

- nombreVector: es el nombre del vector declarada previamente.
- NombreClase: es el nombre de la clase que se desea crear el vector.
- Numero: define el número total de elementos a registrar en el vector de objetos.

Caso desarrollado 1: Mantenimiento de mpleados

Implementa una aplicación que permita realizar el mantenimiento de los empleados de una empresa, para lo cual se debe crear 3 Clases dentro de sus respectivos paquetes: la clase **Empleado** *(clase que controla los atributos privados del empleado y los metodos set-get),* **ArregloEmpleados** *(clase que maneja el vector de objetos) y la clase* **frmMantenimineto** *(clase que interactúa con el usuario a través de la GUI).*

Dentro de la clase **Empleado** *implementar:*

- *Atributos privados del empleado como código, nombre, horas y tarifa.*
- *Método constructor que inicialice los atributos.*
- *Métodos get de todos los atributos.*
- *Métodos set de todos los atributos.*

Dentro de la clase **ArregloEmpleados***:*

- *Declarar como privado el arreglo unidimensional aEmp de tipo Empleado y el índice.*
- *Método constructor que inicialice el vector de empleados en 100 elementos y un índice en cero.*
- *Método* **adiciona***, que se encargará de registrar un empleado en el arreglo e incrementara el índice por cada empleado registrado.*
- *Método* **getTamaño***, que devuelva el tamaño del vector, es decir, el total de empleados registrados.*
- *Método* **obtener***, que devuelve la matriz de elementos de acuerdo a la posición en el arreglo.*
- *Método* **eliminar***, que dependiendo de la posición del código buscado deberá eliminar a todo un registro del arreglo de Objetos.*
- *Método* **buscar***, es el encargado de comparar si el código ingresado existe en la matriz o no; dependiendo de eso deberá devolverá el índice de la posición del elemento encontrado, caso contrario devolverá -1.*

GUI Propuesto:

Fig. 11.1

Debe considerar:

- Crear un nuevo proyecto en NetBeans llamado pjEmpresa.
- Agregar el paquete pFormularios al proyecto pjEmpresa.
- Agregar el paquete pClases al proyecto pjEmpresa.
- Agregar la clase Empleado al paquete pClases.
- Agregar la clase ArregloEmpleados al paquete pClases.
- Agregar la clase frmMantenimiento al paquete pFormularios
- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 11.1, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 11.2.

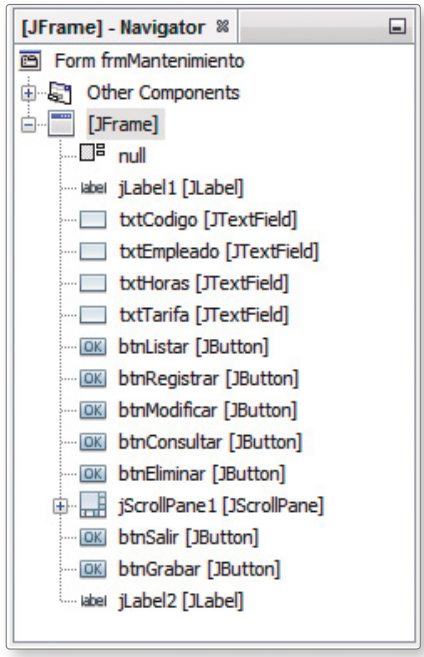


Fig. 11.2

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.

El siguiente script muestra el contenido de la clase Empleado.

```
package pClases;
public class Empleado {

    //Atributos de la clase
    private int codigo;
    private String nombre;
    private int horas;
    private double tarifa;

    //Metodo Constructor
    public Empleado(int codigo,String nombre,
                    int horas,double tarifa){
        this.codigo=codigo;
        this.nombre=nombre;
        this.horas=horas;
        this.tarifa=tarifa;

    }

    //Metodos Get y Set de cada atributo
    public int getCodigo() {
        return codigo;
    }
```

```
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getHoras() {
        return horas;
    }

    public void setHoras(int horas) {
        this.horas = horas;
    }

    public double getTarifa() {
        return tarifa;
    }

    public void setTarifa(double tarifa) {
        this.tarifa = tarifa;
    }
}
```

El siguiente script muestra el contenido de la clase ArregloEmpleados.

```
package pClases;
import javax.swing.JOptionPane;

public class ArregloEmpleados {
    //Atributos de la clase ArregloEmpleados
    private Empleado aE[];
    private int indice;


    //Metodo Constructor
    public ArregloEmpleados(){
        aE=new Empleado[100];
        indice=0;
    }

    //Metodo que registrar los valores de un empleado
    //en el arreglo de empleados
    public void adiciona(Empleado objEmp){
        aE[indice]=objEmp;
        indice++;
    }

    //Metodo que devuelve los datos del Empleado desde el arreglo
    public Empleado devolver(int pos){
        return aE[pos];
    }
```

```
    //Metodo que devuelve el tamaño del arreglo de Empleados
    public int getTamaño(){
        return indice;
    }

    //Metodo que busca un empleado por su codigo
    public int buscar(int codigo){
        for(int i=0;i<getTamaño();i++)
            if (codigo==aE[i].getCodigo())
                return i;
        return -1;
    }

    //Metodo que permite eliminar un empleado
    public void eliminar(int codigo){
        int pos=buscar(codigo);
        for(int i=pos;i<getTamaño()-1;i++){
            aE[i]=aE[i+1];
        }
        indice--;
    }
}
```

El siguiente script muestra las librerías y variables globales de la clase frmMantenimiento.

```
package pFormularios;
import java.text.DecimalFormat;
import javax.swing.JOptionPane;
import pClases.ArregloEmpleados;
import pClases.Empleado;

public class frmMantenimiento extends javax.swing.JFrame {

    //Crear el objeto de la clase ArregloEmpleado
    ArregloEmpleados aEmp=new ArregloEmpleados();

    //Declaracion de la variable df
    DecimalFormat df;

    public frmMantenimiento() {
        initComponents();
        //Asignacion del formato de decimales para dinero
        df=new DecimalFormat("##0.00");
    }
```

El siguiente script muestra los métodos de la clase frmMantenimiento.

```
//Metodo que captura los valores ingresados por el usuario
int getCodigo(){
    return Integer.parseInt(txtCodigo.getText());
}

String getEmpleado(){
    return txtEmpleado.getText();
}

int getHoras(){
    return Integer.parseInt(txtHoras.getText());
}

double getTarifa(){
    return Double.parseDouble(txtTarifa.getText());
}

//Metodo que limpia la tabla de Empleados
void limpiaTabla(){
     for(int i=0;i<50;i++){
        tEmpleado.setValueAt("", i, 0);
        tEmpleado.setValueAt("", i, 1);
        tEmpleado.setValueAt("", i, 2);
        tEmpleado.setValueAt("", i, 3);
        tEmpleado.setValueAt("", i, 4);
     }
}

//Metodo que lista los valores a la tabla
void listar(){
     limpiaTabla();
     for(int i=0;i<aEmp.getTamaño();i++){
        tEmpleado.setValueAt(aEmp.devolver(i).getCodigo(), i, 0);
        tEmpleado.setValueAt(aEmp.devolver(i).getNombre(), i, 1);
        tEmpleado.setValueAt(aEmp.devolver(i).getHoras(), i, 2);
        tEmpleado.setValueAt(df.format(aEmp.devolver(i).
                                       getTarifa()), i, 3);
        tEmpleado.setValueAt(df.format(aEmp.devolver(i).getTarifa()*
                                  aEmp.devolver(i).getHoras()), i, 4);
     }
}

//Metodo que valida el ingreso de los datos del usuario
String valida(){
  if (txtCodigo.getText().equals("") ||
                      Integer.parseInt(txtCodigo.getText())<0){
     txtCodigo.requestFocus();
     return "Codigo del Empleado";
     } else if (txtEmpleado.getText().equals("")){
         txtEmpleado.requestFocus();
         return "Nombre del Empleado";
       } else if (txtHoras.getText().equals("") ||
                  Integer.parseInt(txtHoras.getText())<0){
         txtHoras.requestFocus();
         return "Hora de trabajo";
       } else if (txtTarifa.getText().equals("") ||
                  Double.parseDouble(txtTarifa.getText())<0){
         txtHoras.requestFocus();
         return "Monto de Tarifa";
       } else
         return "";
}
```

El siguiente script muestra las instrucciones del botón btnRegistrar.

```
private void btnRegistrarActionPerformed(java.awt.event.ActionEvent evt) {
   try{
     if (valida().equals("")){
       //Creando un objeto de la clase Empleado
       Empleado objEmp = new Empleado(getCodigo(), getEmpleado(),
                                      getHoras(),getTarifa());

       //Invocar al metodo registrar de la clase ArregloEmpleados
       aEmp.adiciona(objEmp);
       limpiarControles();
       listar();
       } else {
           JOptionPane.showMessageDialog(null,"Error en "+valida());
       }
      } catch(Exception ex){
        JOptionPane.showMessageDialog(null,
                                      "Error en la aplicacion..!!");
   }
}
```

El siguiente script muestra las instrucciones del botón btnConsultar.

```
private void btnConsultarActionPerformed(…) {
   try {
     limpiaTabla();
     //Solicitar el codigo del empleado a consultar
     int codigo=Integer.parseInt(JOptionPane.
                showInputDialog(null,"Ingrese codigo a buscar"));

     //El codigo es enviado al metodo buscar
     int pos=aEmp.buscar(codigo);
     if (pos==-1)
        JOptionPane.showMessageDialog(null,"Codigo no existe");
     else{
      //Se muestran los valores en el control tEmpleado
     tEmpleado.setValueAt(aEmp.devolver(pos).getCodigo(), 0, 0);
     tEmpleado.setValueAt(aEmp.devolver(pos).getNombre(), 0, 1);
     tEmpleado.setValueAt(aEmp.devolver(pos).getHoras(), 0, 2);
     tEmpleado.setValueAt(df.format(aEmp.devolver(pos).
                                    getTarifa()), 0, 3);
     tEmpleado.setValueAt(df.format(aEmp.devolver(pos).getTarifa()*
                            aEmp.devolver(pos).getHoras()), 0, 4);
     }
   } catch(Exception ex){
     JOptionPane.showMessageDialog(null,"Error en la aplicacion..!!");
  }
}
```

El siguiente script muestra las instrucciones del botón btnListar.

```
private void btnListarActionPerformed(java.awt.event.ActionEvent evt) {
    listar();
}
```

El siguiente script muestra las instrucciones del botón btnEliminar.

```
private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
  try {
     //Solicitando el codigo del empleado a eliminar
     int codigo=Integer.parseInt(JOptionPane.
                showInputDialog(null,"Ingrese codigo a eliminar: "));

     //Invocando al metodo eliminar del arreglo
     aEmp.eliminar(codigo);
     JOptionPane.showMessageDialog(null,"Empleado eliminado...!!");
     } catch(Exception ex){
     JOptionPane.showMessageDialog(null,"Error en la aplicacion..!!");
     }
}
```

El siguiente script muestra las instrucciones del botón btnModificar.

```
private void btnModificarActionPerformed(…) {
   try {
     int codigoB=Integer.parseInt(JOptionPane.showInputDialog(this,
                                  "Ingrese Codigo a Modificar:"));

     //Buscar si el codigo del empleado Existe en el Arreglo
     int pos=aEmp.buscar(codigoB);
     if (pos!=-1){
         //Devuelve los datos del Empleado encontrado
         Empleado x=aEmp.devolver(pos);

         //Habilita el boton btnGrabar y oculta el boton btnModificar
         btnGrabar.setVisible(true);
         btnModificar.setVisible(false);

         //Se muestra el codigo del empleado y se bloquea
         txtCodigo.setText(""+codigoB);
         txtCodigo.setEditable(false);

         //Se envian los valores desde el arreglo a los controles
         txtEmpleado.setText(x.getNombre());
         txtHoras.setText(""+x.getHoras());
         txtTarifa.setText(""+x.getTarifa());

         txtEmpleado.requestFocus();
         } else {
         //Mensaje de error cuando el codigo del empleado sea errado
            JOptionPane.showMessageDialog(null,"Codigo no existe");
       }
      } catch(Exception ex){
            JOptionPane.showMessageDialog(null,
                                     "Error en la aplicacion..!!");
      }
}
```

El siguiente script muestra las instrucciones del botón btnGrabar.

```
private void btnGrabarActionPerformed(java.awt.event.ActionEvent evt) {
   try{
      //buscando el codigo del empleado a modificar
      int pos=aEmp.buscar(getCodigo());

      //La posicion es enviada al arreglo para devolver
      //el empleado buscado
      Empleado x=aEmp.devolver(pos);

      //Actualizando los valores del empleado
      x.setNombre(getEmpleado());
      x.setHoras(getHoras());
      x.setTarifa(getTarifa());

      JOptionPane.showMessageDialog(null,
                  "Empleado actualizado correctamente",
                  "Confirmacion",JOptionPane.INFORMATION_MESSAGE);
      btnModificar.setVisible(true);
      btnGrabar.setVisible(false);
      }catch(Exception ex){
        JOptionPane.showMessageDialog(null,"Error al actualizar...!!",
                                      "Error",
                                      JOptionPane.ERROR_MESSAGE);
   }
}
```

11.2. Clase ArrayList

La clase ArrayList permite tener un control de los datos haciendo el trabajo de un vector de objetos, esta clase dispone de diversos métodos para manipular una colección de objetos dinámicamente.

11.2.1. Formato para crear un ArrayList

```
ArrayList <nombre_clase> objeto = new ArrayList < nombre_clase > ();
```

Donde:

- ArrayList: es la representación de la clase ArrayList.
- <nombre_clase>: es el nombre de la clase donde se creará el vector.
- Objeto: es el nombre asignado al ArrayList.

Por ejemplo tenemos: ArrayList <**Producto**> **prod** = new ArrayList <**Producto**> ();

11.2.2. Métodos que componen la clase ArrayList

MÉTODOS	DESCRIPCIÓN

add(Object)	Agrega un elemento al final.
	`public void adicionar(Producto x){`
	`prod.add(x);`
	`}`

add(int, Object)	Agrega un elemento en la posición especificada en el primer parámetro. `prod.add(0,x);`
clear()	Elimina todos los elementos. `prod.clear();`

get(int)	Devuelve el elemento de la posición especificada.
	`public Producto obtener(int pos){`
	`return prod.get(pos);`
	`}`

indexOf(Object)	Devuelve el índice del elemento especificado, de no encontrarlo devuelve -1.
	`public int posicion(Producto x){`
	`return prod.indexOf(x);`
	`}`

remove(int)	Elimina el elemento de la posición especificada.
	`public void eliminar(int x){`
	`prod.remove(x);`
	`}`

remove(Object)	Elimina el elemento especificado.
	`public void eliminar(Producto x){`
	`prod.remove(x);`
	`}`

set(int, Object)	Reemplaza el elemento de la posición especificada en el primer parámetro por el elemento del segundo parámetro.
	`public void modificar(int pos, Producto x){`
	`prod.set(pos,x);`
	`}`

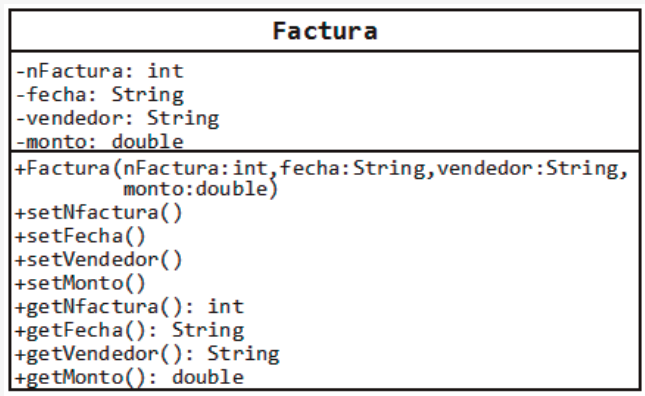
size()	Devuelve la cantidad de elementos agregados.
	`public int tamaño(){`
	`return prod.size();`
	`}`

CASO DESARROLLADO 2: MANTENIMIENTO DE FACTURAS

Implementar una aplicación que permita controlar las facturas de una tienda comercial, para lo cual se deben crear 3 Clases en sus paquetes respectivos. La clase **Factura** *(clase que controla los atributos privados de la Factura),* **ArregloFacturas** *(clase que maneja el vector de tipo Factura) y la clase* **frmVenta** *(clase que interactúa con el usuario a través de la GUI).*

Dentro de la clase **Factura** *implementar:*

- *Atributos privados para Nfactura(int), fecha(String), vendedor(String) y monto(double).*
- *Método constructor que inicialice los atributos*
- *Métodos get de todos los atributos*
- *Métodos set de todos los atributos.*



Dentro de la clase **ArregloFacturas***:*

- *Declarar como atributo privado el arreglo unidimensional* **fact** *de tipo* **ArrayList** *y el índice.*
- *Método constructor que inicialice el arreglo de* **fact** *de tipo* **ArrayList***.*
- *Método* **agregar***, que se encargará de registrar una factura en el* **ArrayList***.*
- *Método* **getTamaño***, que devuelva el tamaño del ArrayList.*
- *Método* **obtener***, que devuelva todos los datos registrados de la Factura de acuerdo a la posición en el* **ArrayList***.*
- *Método* **buscar***, es el encargado de comparar si el* **número de Factura** *ingresado existe en la matriz o no, dependiendo de eso deberá devolver el* **objeto de la Factura** *encontrado, caso contrario devolver* **null***. Aquí utilizar la estructura for de la siguiente forma:*

```
for(Clase varReferencia : NombreDelArrayList){
  return varReferencia;
}
```

Ejm. *Si tiene la Clase* **Personal** *y quisiera buscarla por su código de tipo* **int***, dentro del ArrayList Per.*

```
public Personal buscar(int codigo){
```

- *Método* **eliminar**, *que dependiendo del tipo de objeto deberá eliminarlo del* **ArrayList**.

ArregloFacturas
-fact: ArrayList <Factura> -indice: int
+ArregloFacturas() +getTamaño(): int +agregar(f:Factura) +obtener(pos:int): Factura +buscar(num:int): Factura +eliminar(x:Factura)

Adicionalmente, implemente en la Clase **ArregloFacturas** *los siguientes métodos:*

- *Método que retorne el nombre del empleado que tiene la Venta más alta.*
- *En el formulario diseñar las opciones de* **mantenimiento** *como* **registrar** *(para un nuevo registro de Factura generando el número de Factura en forma correlativa),* **consultar** *(que permite buscar una determinada Factura,* **modificar** *(que permite modificar el nombre del vendedor y el monto registrado mas no el número ni la fecha ya que estas son autogeneradas) y* **Eliminar** *(que de acuerdo a un determinado objeto deberá eliminar un registro de empleado del ArrayList).*

GUI Propuesto:

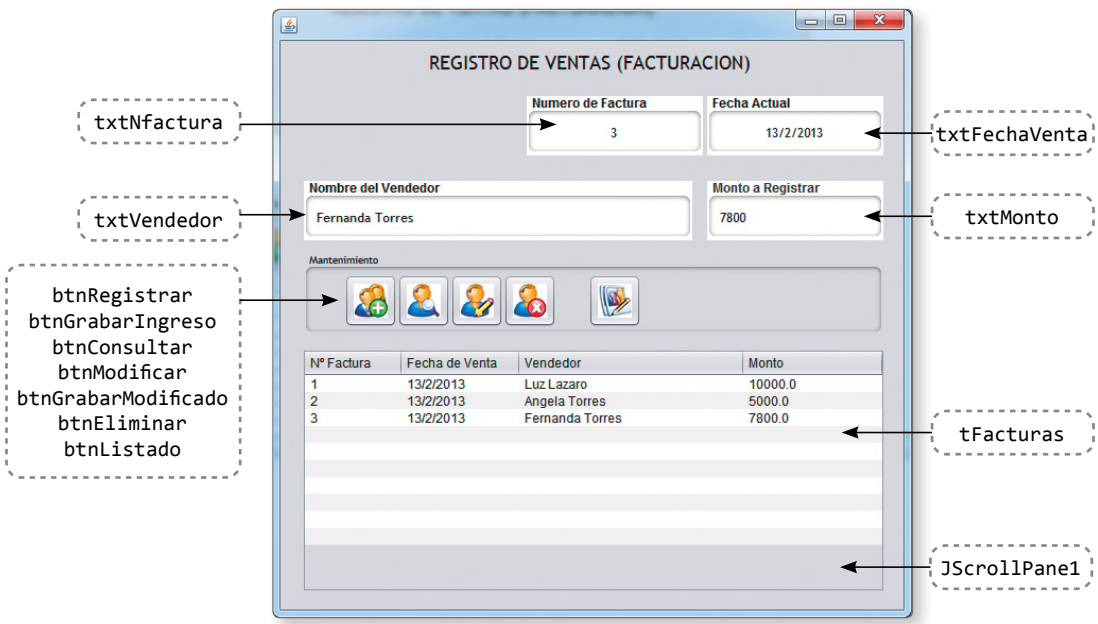


Fig. 11.3

Debe considerar:

- Crear un nuevo proyecto en NetBeans llamado pjArrayList.
- Agregar el paquete pFormularios al proyecto pjArrayList.

- Agregar el paquete pClases al proyecto pjArrayList.
- Agregar la clase Factura al paquete pClases.
- Agregar la clase ArregloFacturas al paquete pClases.
- Agregar la clase frmMantenimiento al paquete pFormularios
- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 11.3, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**...
- Después de asignar nombres a los objetos asegúrese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 11.4.

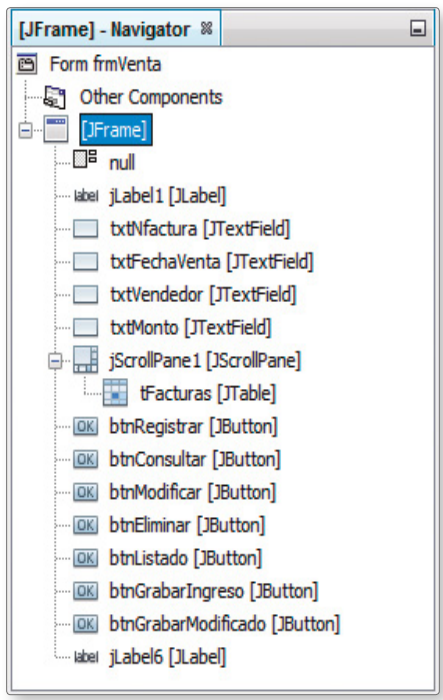


Fig. 11.4

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.

El siguiente script muestra el contenido de la clase Factura.

```
package pClases;

public class Factura {
    private int nFactura;
    private String fecha;
    private String vendedor;
    private double monto;

    //Metodo constructor
    public Factura(int nFactura,String fecha,
                   String vendedor,double monto){
```

```
        this.nFactura=nFactura;
        this.fecha=fecha;
        this.vendedor=vendedor;
        this.monto=monto;
    }

    //Metodos set
    public void setNfactura(int nFactura){
        this.nFactura=nFactura;
    }
    public void setFecha(String fecha){
        this.fecha=fecha;
    }
    public void setVendedor(String vendedor){
        this.vendedor=vendedor;
    }
    public void setMonto(double monto){
        this.monto=monto;
    }

    //Metodos get
    public int getNfactura(){
        return nFactura;
    }
    public String getfecha(){
        return fecha;
    }
    public String getVendedor(){
        return vendedor;
    }
    public double getMonto(){
        return monto;
    }
}
```

El siguiente script muestra el contenido de la clase ArregloFacturas.

```
package pClases;
import java.util.ArrayList;

public class ArregloFacturas {
    private ArrayList <Factura> fact;
    private int indice;

    //Metodo constructor
    public ArregloFacturas(){
        fact=new ArrayList<Factura>();
    }

    //Metodo que devuelve el tamaño actual del vector
    public int getTamaño(){
        return fact.size();
    }

    //Metodo que permite agregar una factura al vector
    public void agregar(Factura F){
        fact.add(F);
    }

    //Metodo que devuelve el objeto factura
```

```
    public Factura obtener(int pos){
        return fact.get(pos);
    }

    //Metodo que busca una factura
    public Factura buscar(int num){
        for (Factura f: fact)
            if (f.getNfactura()==num)
                return f;
        return null;
    }

    //Metodo que elimina una factura
    public void eliminar(Factura x){
        fact.remove(x);
    }
}
```

El siguiente script muestra los librerías y variables globales de la clase frmMantenimiento.

```
package pFormularios;
import java.util.GregorianCalendar;
import javax.swing.JOptionPane;
import javax.swing.table.TableColumn;
import pClases.ArregloFacturas;
import pClases.Factura;

public class frmVenta extends javax.swing.JFrame {

    ArregloFacturas f=new ArregloFacturas();
    int num=0;

    public frmVenta() {
        initComponents();
        DefinirAnchos();
        asignaFecha();
        habilitaCajas(false);
        btnGrabarIngreso.setVisible(false);
        btnGrabarModificado.setVisible(false);
    }
```

El siguiente script muestra los métodos de la clase frmMantenimiento.

```
//Metodo que define el ancho de las columnas de la tabla
    void DefinirAnchos(){
        TableColumn columna;
        columna=tFacturas.getColumnModel().getColumn(0);
        columna.setPreferredWidth(30);
        columna=tFacturas.getColumnModel().getColumn(1);
        columna.setPreferredWidth(50);
        columna=tFacturas.getColumnModel().getColumn(2);
        columna.setPreferredWidth(150);
        columna=tFacturas.getColumnModel().getColumn(3);
        columna.setPreferredWidth(70);
        tFacturas.getTableHeader().setReorderingAllowed(false);
        tFacturas.getTableHeader().setResizingAllowed(false);
    }

    //Metodo que bloquea y desbloquea los controles JTextField
```

```
void habilitaCajas(boolean opcion){
    txtNfactura.setEditable(opcion);
    txtVendedor.setEditable(opcion);
    txtMonto.setEditable(opcion);
    txtFechaVenta.setEditable(opcion);

}

//Metodo que limpia los controles JTextField
void limpiaCajas(){
    txtNfactura.setText("");
    txtVendedor.setText("");
    txtFechaVenta.setText("");
    txtMonto.setText("");
}

//Metodo que limpia el control tFacturas
void limpiaMatriz(){
    for(int i=0;i<10;i++){
        tFacturas.setValueAt("", i, 0);
        tFacturas.setValueAt("", i, 1);
        tFacturas.setValueAt("", i, 2);
        tFacturas.setValueAt("", i, 3);
    }
}

//Metodo que genera el numero de factura
public int generaNumero(){
    num++;
    return num;
}

//Metodos que capturan los valores ingresados por el usuario
public int getNumFact(){
    return Integer.parseInt(txtNfactura.getText());
}
public String getFecha(){
    return txtFechaVenta.getText();
}
public String getVendedor(){
    return txtVendedor.getText();
}
    public double getMonto(){
    return Double.parseDouble(txtMonto.getText());
}

//Metodo que lista las facturas en el control tFacturas
void listar(){
    if (f.getTamaño()>0){
        for(int i=0;i<f.getTamaño();i++){
            Factura fact=f.obtener(i);
            tFacturas.setValueAt(fact.getNfactura(),i,0);
            tFacturas.setValueAt(fact.getfecha(), i, 1);
            tFacturas.setValueAt(fact.getVendedor(), i, 2);
            tFacturas.setValueAt(fact.getMonto(), i, 3);
        }
    } else {
        JOptionPane.showMessageDialog(this,
                "No hay facturas registradas",
                "Confirmacion",JOptionPane.INFORMATION_MESSAGE);
        limpiaMatriz();
    }
}
```

```
    //Metodo que muestra la fecha actual en el control txtFechaVenta
    void asignaFecha(){
        GregorianCalendar cal=new GregorianCalendar();
        txtFechaVenta.setText(cal.get(cal.DAY_OF_MONTH)+"/"
                              +cal.MONTH+"/"
                              +cal.get(cal.YEAR));
    }
```

El siguiente script muestra las instrucciones del botón btnRegistrar.

```
private void btnRegistrarActionPerformed(…) {
        txtNfactura.setText(""+generaNumero());
        asignaFecha();
        txtVendedor.requestFocus();

        habilitaCajas(true);
        txtVendedor.setEditable(true);
        txtMonto.setEditable(true);

        txtVendedor.setText("");
        txtMonto.setText("");

        btnRegistrar.setVisible(false);
        btnGrabarIngreso.setVisible(true);
}
```

El siguiente script muestra las instrucciones del botón btnConsultar.

```
private void btnConsultarActionPerformed(…) {
    try{
       limpiaCajas();
       limpiaMatriz();
       int buscoFactura=Integer.parseInt(JOptionPane.
                        showInputDialog(null,
                        "Ingrese un numero de Factura:"));

       //objeto fact que busca el numero de factura en el ArrayList f
       Factura fact=f.buscar(buscoFactura);
       if (fact!=null){
           tFacturas.setValueAt(fact.getNfactura(), 0, 0);
           tFacturas.setValueAt(fact.getfecha(), 0, 1);
           tFacturas.setValueAt(fact.getVendedor(), 0, 2);
           tFacturas.setValueAt(fact.getMonto(), 0, 3);
       }else
           JOptionPane.showMessageDialog(null,"Factura NO encontrada",
                        "Confirmacion",JOptionPane.ERROR_MESSAGE);
        } catch(Exception ex){
       JOptionPane.showMessageDialog(null,"Error de Entrada de Datos",
                   "Confirmacion",JOptionPane.ERROR_MESSAGE);
    }
}
```

El siguiente script muestra las instrucciones del botón btnModificar.

```
private void btnModificarActionPerformed(…) {
   try{
      limpiaCajas();
      limpiaMatriz();

      btnModificar.setVisible(false);
      btnGrabarModificado.setVisible(true);

      int buscoFactura=Integer.parseInt(JOptionPane.
      showInputDialog(null,"Ingrese un numero de Factura:"));

   //objeto fact que busca el numero de factura en el ArrayList f
      Factura fact=f.buscar(buscoFactura);
      if (fact!=null){
          tFacturas.setValueAt(fact.getNfactura(), 0, 0);
          tFacturas.setValueAt(fact.getfecha(), 0, 1);
          tFacturas.setValueAt(fact.getVendedor(), 0, 2);
          tFacturas.setValueAt(fact.getMonto(), 0, 3);

          txtNfactura.setText(""+fact.getNfactura());
          txtFechaVenta.setText(fact.getfecha());
          txtVendedor.setText(fact.getVendedor());
          txtMonto.setText(""+fact.getMonto());

          habilitaCajas(true);
          txtNfactura.setEditable(false);
          txtFechaVenta.setEditable(false);
       }else
         JOptionPane.showMessageDialog(null,"Factura NO encontrada",
                           "Confirmacion",JOptionPane.ERROR_MESSAGE);
       }catch(Exception ex){
        JOptionPane.showMessageDialog(null,"Factura NO encontrada",
               "Confirmacion",JOptionPane.ERROR_MESSAGE);
        btnModificar.setVisible(true);
        btnGrabarModificado.setVisible(false);
     }
}
```

El siguiente script muestra las instrucciones del botón btnEliminar.

```
private void btnEliminarActionPerformed(…) {
  try{
     int buscoFactura=Integer.parseInt(JOptionPane.
                      showInputDialog(null,
                      "Ingrese un numero de Factura a Eliminar:"));
     //objeto fact que busca el numero de factura en el ArrayList f
     Factura fact=f.buscar(buscoFactura);
     if (fact!=null){
        f.eliminar(fact);
        JOptionPane.showMessageDialog(null,"Factura Eliminada",
              "Confirmacion",JOptionPane.INFORMATION_MESSAGE);
        listar();
     } else
       JOptionPane.showMessageDialog(null,
                     "NO existe el Numero de Factura ingresada",
                     "Confirmacion",JOptionPane.INFORMATION_MESSAGE);
     }catch(Exception ex){
        JOptionPane.showMessageDialog(null,
                    "NO existe el Numero de Factura ingresada",
                    "Confirmacion",JOptionPane.INFORMATION_MESSAGE);
     }
}
```

El siguiente script muestra las instrucciones del botón btnListar.

```
private void btnListadoActionPerformed(java.awt.event.ActionEvent evt) {
   listar();
}
```

El siguiente script muestra las instrucciones del botón btnGrabarIngreso.

```
private void btnGrabarIngresoActionPerformed(…) {
  try{
     habilitaCajas(false);
     btnRegistrar.setVisible(true);
     btnGrabarIngreso.setVisible(false);

     Factura fact=new Factura(getNumFact(),getFecha(),
                              getVendedor(),getMonto());

     f.agregar(fact);
     listar();
     JOptionPane.showMessageDialog(null,
                    "Factura ingresada correctamente",
                    "Confirmacion",JOptionPane.INFORMATION_MESSAGE);
     }catch(Exception ex){
     JOptionPane.showMessageDialog(null,
                     "Error de Ingreso de Datos",
                     "Error",JOptionPane.ERROR_MESSAGE);
     num--;
     }
}
```

El siguiente script muestra las instrucciones del botón btnGrabarModificado.

```
private void btnGrabarModificadoActionPerformed(…) {
  try{
     Factura fact=f.buscar(getNumFact());
     fact.setVendedor(getVendedor());
     fact.setMonto(getMonto());
     JOptionPane.showMessageDialog(null,
              "Factura Modificada Correctamente",
              "Confirmacion",JOptionPane.INFORMATION_MESSAGE);
     listar();
     }catch(Exception ex){
       JOptionPane.showMessageDialog(null,
              "Ocurrio un error al intentar Grabar",
              "Confirmacion",JOptionPane.INFORMATION_MESSAGE);
     }
     btnGrabarModificado.setVisible(false);
     btnModificar.setVisible(true);
}
```

CAP.

12

Archivos de texto

CAPACIDAD:

- Reconocerá el trabajo de los archivos de texto.
- Implementará una aplicación de mantenimiento con datos registrados en un archivo de texto.

CONTENIDO:

12.1. CONFIGURACIÓN DEL JDK

El JDK deberá estar preparado para manipular archivos de texto, es decir, habilitar la escritura, sobreEscritura, Eliminación. En el caso de Netbeans tiene habilitado todos estos servicios, pero es necesario conocer que en otros IDE'S deberán ser habilitados.

PASO 1:

Direccionar a → C:\Program Files\Java\jdk1.6.0_18\bin

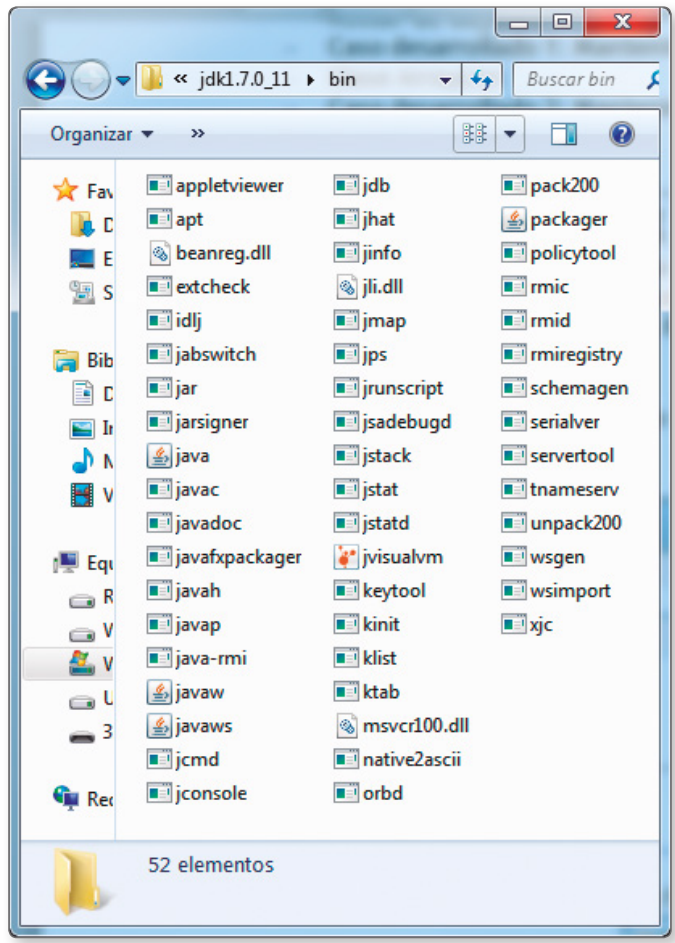


Fig. 12.1

En la Fig. 12.1 se muestran los archivos que contiene la carpeta bin del JDK de Java.

PASO 2:

Seleccionar y ejecutar el archivo > **policytool**; OJO..!! Si usted se encuentra en Windows 7 o superior tendrá que ejecutar el archivo en modo Administrador, para esto deberá presionar clic derecho sobre el archivo **policytool** y seleccionar **Ejecutar como Administrador**.

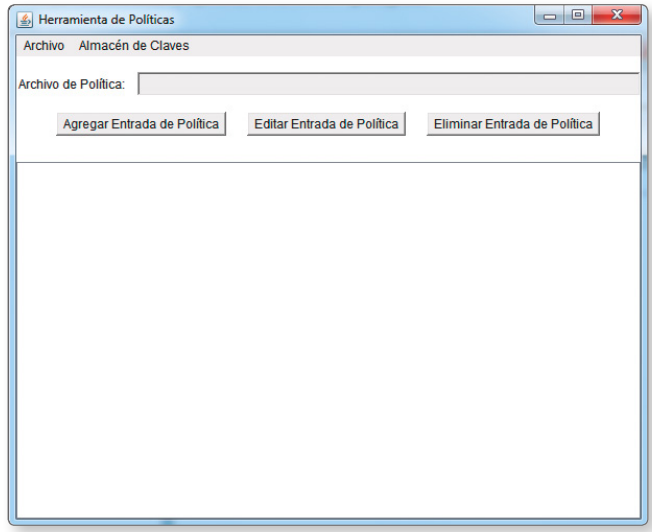


Fig. 12.2

PASO 3:

Seleccione Archivo→Abrir de la ventana Herramienta de políticas > buscar y ejecutar el archivo **java. policy;** en la siguiente ruta:

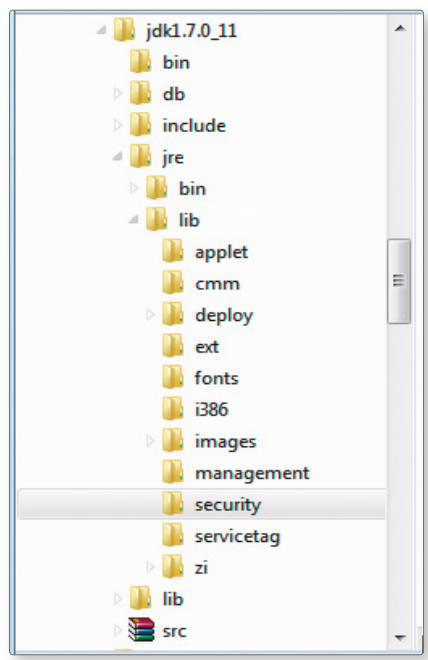


Fig. 12.3

Si usted cuenta con una versión distinta de JDK no debe preocuparse, puesto que la ruta que indicamos es la misma en todas las versiones.

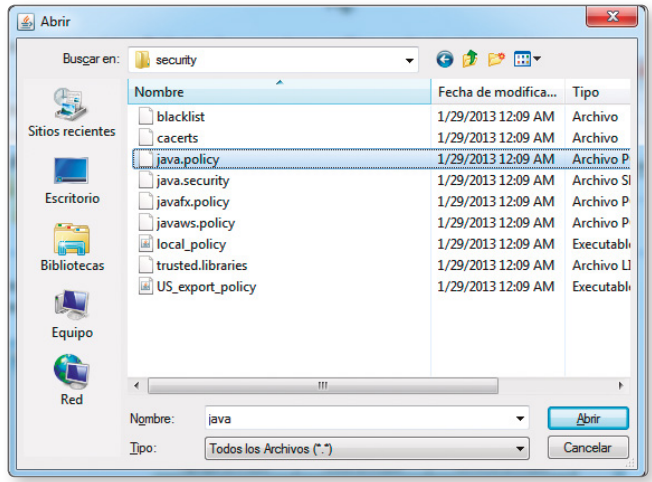


Fig. 12.4

Una vez abierto el archivo java.policy se muestra la siguiente ventana:

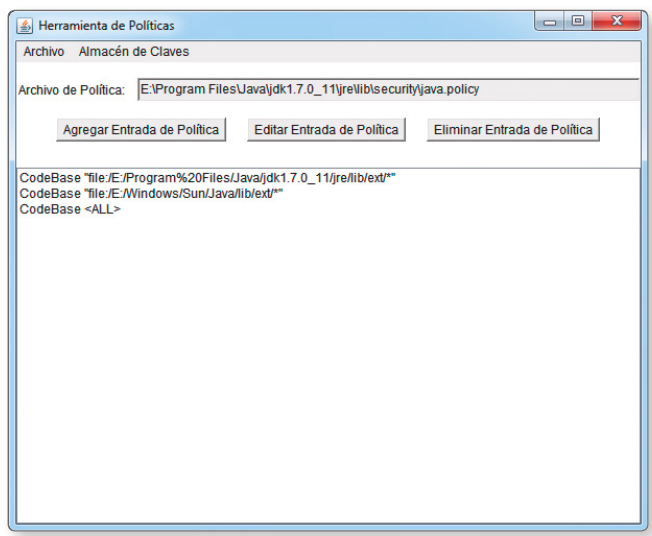


Fig. 12.5

PASO 4:

Seleccionar: **CodeBase <ALL>** luego presione > **Editar entrada de Política**.

PASO 5:

Seleccionar: **Agregar Permiso** de la ventana Entrada de política.

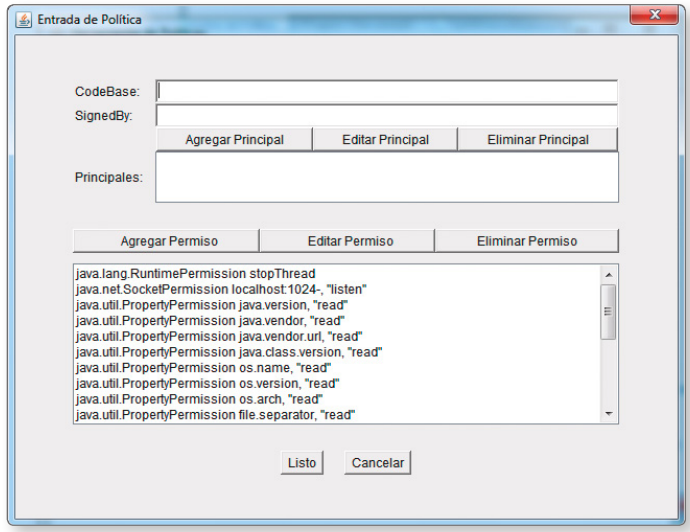


Fig. 12.6

PASO 6:

Seleccionar:

a. Permiso : FilePermission

b. Nombre de Destino : <<ALL FILES>>

c. Acciones : read, write, delete, execute (en este caso tendrá que seleccionar las 4 acciones).

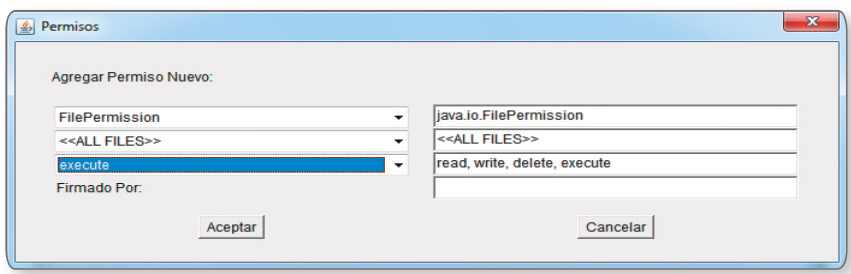


Fig. 12.7

PASO 7:

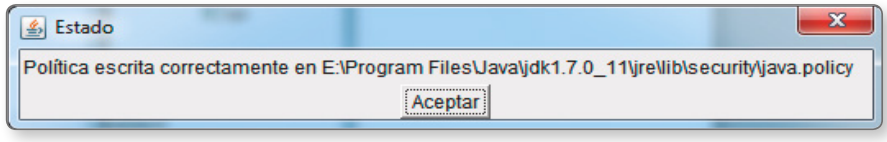
Seleccionar **Aceptar** > **Listo**

PASO 8:

Seleccione: **Archivo** > **Guardar**

PASO 9:

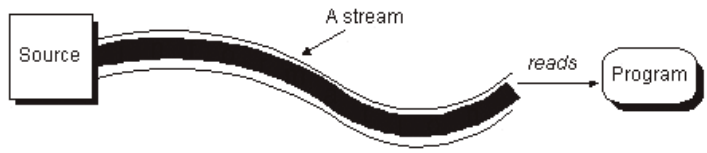
Para constatar que todos los cambios han sido registrados en forma correcta, deberá mostrarse un mensaje como la siguiente imagen:



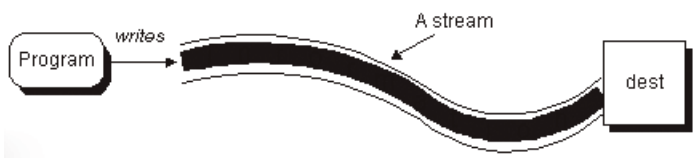
12.2. LIBRERÍAS A UTILIZAR PARA EL MANEJO DE ARCHIVOS

Frecuentemente los programas necesitan traer información desde una fuente externa o enviar información a una fuente externa. La información puede estar en cualquier parte, en un fichero, en disco, en algún lugar de la red, en la memoria o en otro programa. También puede ser de cualquier tipo: objetos, caracteres, imágenes o sonidos.

Para traer la información, un programa abre un stream sobre una fuente de información (un fichero, memoria, un socket) y lee la información serialmente, de esta forma:



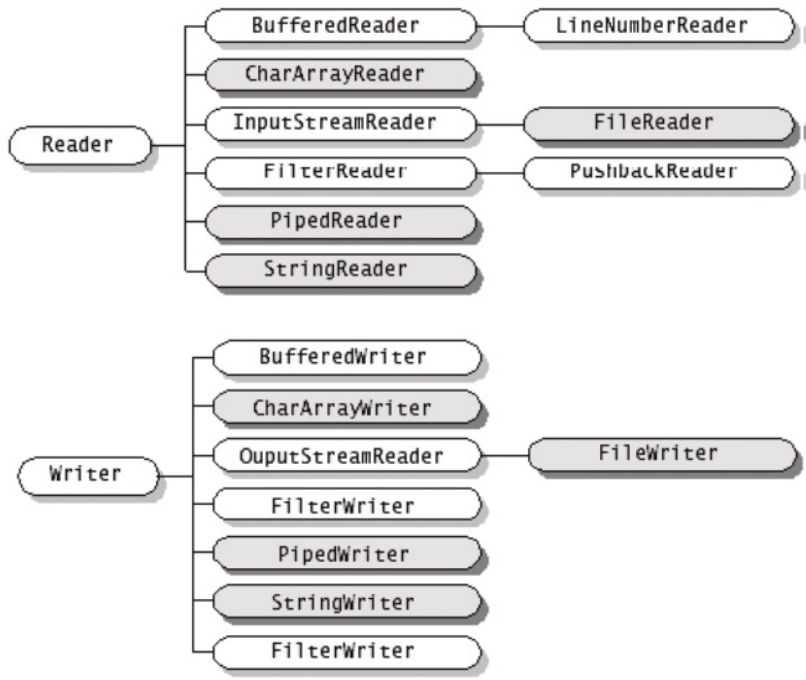
Similarmente, un programa puede enviar información a un destino externo abriendo un stream sobre un destino y escribiendo la información serialmente, de esta forma:



El paquete java.io contiene una colección de clases stream que soportan estos algoritmos para leer y escribir. Estas clases están divididas en dos árboles basándose en los tipos de datos (caracteres o bytes) sobre los que opera.

Sin embargo, algunas veces es más conveniente agrupar las clases basándose en su propósito en vez que en los tipos de datos que lee o escribe. Así, podemos agrupar los streams dependiendo de si leen u escriben lados en las "profundidades" o procesan la información que está siendo leída o escrita.

12.3. Clases y métodos para el manejo y control de Archivos de texto



BufferedReader	La clase **BufferedReader** deriva de la clase **Reader**. Esta clase añade un buffer para realizar una lectura eficiente de caracteres. Dispone del método **readLine** que permite leer una línea de texto y devolverla como String.
Declaración de una variable tipo bufferedReader	`bufferedReader br;`
Invocación al Método constructor	`br=new BufferedReader(new FileReader(archivotexto));`

FileReader	Esta clase tiene métodos que nos permiten leer caracteres. Sin embargo, suele ser habitual querer las líneas completas, bien porque nos interesa la línea completa, bien para poder analizarla luego y extraer campos de ella. **FileReader** no contiene métodos que nos permitan leer líneas completas, pero sí **BufferedReader**.
Crear un BufferedReader a partir de un FileReader	File archivo = new File (.C:\\archivo.txt"); FileReader fr = new FileReader (archivo); BufferedReader br = new BufferedReader(fr);
Observaciones	◆ Una vez abierto el archivo, br apunta a la primera cadena de bits. ◆ El método **readLine()** asociado a br captura una cadena de bits y saldrá a la siguiente línea de bits. Dicha cadena es convertida seguidamente al tipo de datos requerido por la aplicación. ◆ Cuando no existe más cadena que mostrar br apunta a null. ◆ El método **close()** cierra el acceso al archivo.
Declaración Final	`BuffereReader br=new BufferedReader(new FileReader(archivo))`

PrintWriter	Es la clase que permite escribir los datos de la memoria hacia un archivo de texto (output). Lo primero que tenemos que hacer es crear una variable de tipo PrintWriter
Declaración de una variable tipo PrintWriter	`PrintWriter pw;`
Invocación al Método constructor	`Pw=new PrintWriter(new FileWriter(archivo));`

FileWriter	Esta clase se encarga de abrir el archivo en modo de escritura. Es decir, si el archivo contiene información esta se pierde. Si el archivo no existe lo crea.
Observaciones	◆ Una vez abierto el archivo, pw apunta al inicio. ◆ El método **println(data)** asociado a pw graba como cadena de bits la data indicada y genera un salto de línea en el archivo. ◆ El método **close()** cierra el acceso al archivo.
Declaración	`Pw=new PrintWriter(new FileWriter(archivo));`

CASO DESARROLLADO 1: MANTENIMIENTO DE ESTUDIANTES

Aplicación que permita realizar el mantenimiento de los registros de un estudiante, el cual ingresará un código, nombres, ciclo y pensión. Para lo cual deberá crear 3 clases:

- *Clase* **Estudiante***: que permitirá definir los atributos del Estudiante como su código, nombres, ciclo de estudio y el monto de la pensión.*
- *Clase* **ArregloEstudiantes***: que permitirá almacenar a los registros del estudiante en un vector de objetos.*
- *Clase* **frmMantenimiento***: que permitirá interactuar con el usuario para el control de los registros del estudiante.*

Botón **Ingresar***: tiene por objetivo habilitar las cajas de texto para un nuevo registro de estudiante y generar un código al estudiante de forma automática.*
Botón **Consultar***: tiene por objetivo solicitar un código de estudiante y mostrar los datos del estudiante, para buscarlo en el vector. En caso de no encontrarlo mostrar un mensaje de error.*
Botón **Modificar***: tiene por objetivo solicitar un código de estudiante para buscarlo en el vector y mostrar sus datos en los controles JTextField. Al modificar se habilitará el botón grabar.*
Botón **Eliminar***: tiene por objetivo solicitar un código de estudiante para eliminarlo del vector de estudiantes.*
Botón **Listar***: permite listar los registros contenidos en el vector de estudiantes.*
Botón **GrabarI***: permite grabar los valores del estudiante que se ingresa por primera vez.*
Botón **GrabarM***: permite grabar las modificaciones realizadas sobre un registro de estudiante.*

GUI Propuesto:

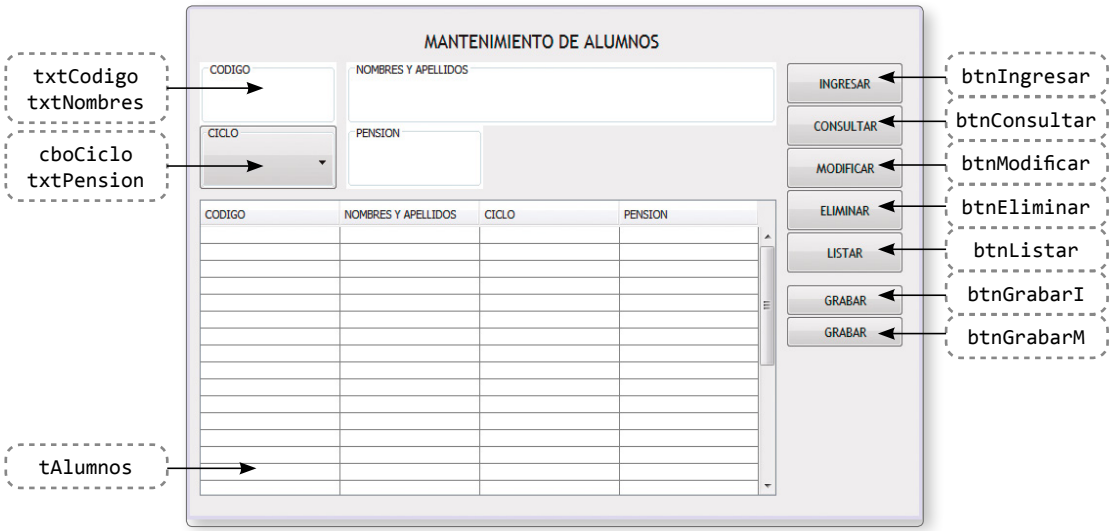


Fig. 12.8

Debe considerar:

- Crear un nuevo proyecto en NetBeans llamado pjInstituto.

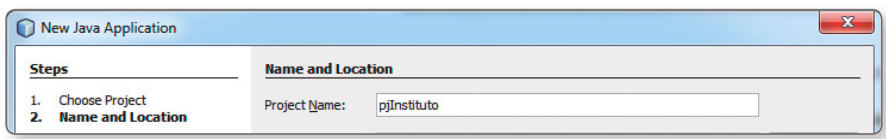


Fig. 12.9

- Agregar el paquete **pFormularios** al proyecto **pjInstituto**.
- Agregar el paquete **pClases** al proyecto **pjInstituto**.
- Agregar la clase **Estudiante** al paquete **pClases**.
- Agregar la clase **ArregloEstudiantes** al paquete **pClases.**
- Agregar la clase **frmMantenimiento** al paquete **pFormularios.**
- Asigne un nombre a cada uno de los controles; como se muestra en la Fig. 12.8, para esto debe presionar clic derecho sobre el objeto y seleccionar > **Change Variable Name**…
- Después de asignar nombres a los objetos, asegúrese que los controles sean los correctos para eso visualice el panel Navigator; debe mostrarse como la Fig. 12.10.

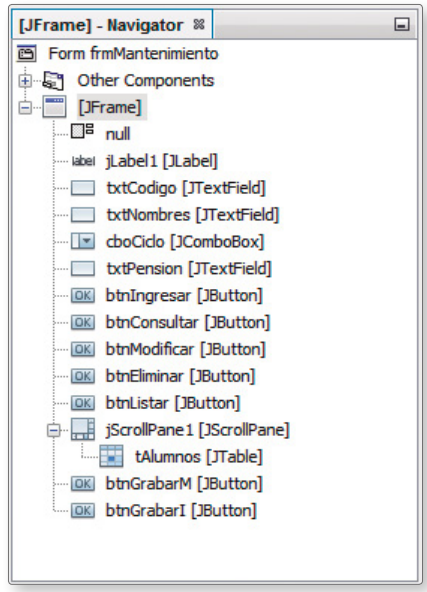


Fig. 12.10

- Asignar Null Layout al setLayout del Frame, presionando clic derecho sobre el Frame.
- Al Frame active el **Form Size Police** con el valor **Generate Resize Code**.

El siguiente script muestra el contenido de la clase **Estudiante**.

```
package pClases;

public class Estudiante {
    //Atributos privados de la clase Estudiante
    private int codigo,ciclo;
    private String nombre;
    private double pension;

    //Metodo constructor
    public Estudiante(int codigo,String nombre,
                      int ciclo,double pension){
        this.codigo=codigo;
        this.nombre=nombre;
        this.ciclo=ciclo;
        this.pension=pension;
    }

    //Metodos GET y SET de los atributos de la clase Estudiante
    public int getCodigo(){
        return codigo;
    }
    public String getNombre(){
        return nombre;
    }
    public int getCiclo(){
        return ciclo;
    }
    public double getPension(){
```

```
        return pension;
    }
    public void setCodigo(int codigo){
        this.codigo=codigo;
    }
    public void setNombre(String nombre){
        this.nombre=nombre;
    }
    public void setCiclo(int ciclo){
        this.ciclo=ciclo;
    }
    public void setPension(double pension){
        this.pension=pension;
    }
}
```

El siguiente script muestra el contenido de la clase **ArregloEstudiantes**.

```
package pClases;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.StringTokenizer;
import javax.swing.JOptionPane;

public class ArregloEstudiantes {

     private ArrayList <Estudiante> aEst;

    /*
       Metodo constructor que crea el objeto aEst de tipo ArrayList
       y carga el contenido del archivo estudiantes.txt
    */
    public ArregloEstudiantes(){
        aEst=new ArrayList<Estudiante>();
        cargar();
    }

    /*Metodo que adiciona un objeto de tipo Estudiante al ArrayList*/
    public void adicionar(Estudiante e){
        aEst.add(e);
    }

    /*Metodo que devuelve los elementos contenidos en el arreglo
     segun una posicion especificada
     */
    public Estudiante obtener(int pos){
        return aEst.get(pos);
    }

    /*Metodo que busca un estudiante desde el arreglo y devuelve
     el objeto de tipo estudiante, sino lo encuentra devolvera NULL*/
    public Estudiante buscar(int cod){
        for(int i=0;i<aEst.size();i++){
            if (cod==aEst.get(i).getCodigo())
                return aEst.get(i);
```

```
        }
        return null;
    }

    /*Metodo que devuelve el total de estudiantes registrados*/
    public int getTamaño(){
        return aEst.size();
    }

    /*Metodo que permite eliminar un estudiante del arreglo mediante
     el objeto de tipo Estudiante*/
    public void eliminar(Estudiante e){
        aEst.remove(e);
    }

    /*Metodo que permite abrir el archivo Estudiante.txt
     y obtener los valores contenidos almacenandolo en variables
     locales*/
    public void cargar(){
    try{
        File archivo = new File(.Estudiantes.txt");
        if (archivo.exists()){
            BufferedReader br=new BufferedReader(
                              new FileReader(.Estudiantes.txt"));
            String linea;
        while((linea=br.readLine())!=null){
            //Definición del separador de valores en Estudiantes.txt
            StringTokenizer st=new StringTokenizer(linea,",");

            int cod=Integer.parseInt(st.nextToken().trim());
            String nom=st.nextToken().trim();
            int cic=Integer.parseInt(st.nextToken().trim());
            double pen=Double.parseDouble(st.nextToken().trim());

            //Crear un nuevo registro de estudiante obtenido del
            //archivo Estudiantes.txt
            Estudiante x=new Estudiante(cod,nom,cic,pen);
            adicionar(x);
        }
        br.close();
        } else
        JOptionPane.showMessageDialog(null,
                                      .El archivo de texto no existe");
    } catch(Exception x){
        JOptionPane.showMessageDialog(null, .Se produjo un Error .+x);
  }
}

/*Metodo que permite grabar los nuevos valores de un estudiante*/
public void grabar(){
    try{
      PrintWriter pw=new PrintWriter(
                                  new FileWriter(.Estudiantes.txt"));
      for(int i=0;i<getTamaño();i++){
          pw.println(obtener(i).getCodigo()+","+
          obtener(i).getNombre()+","+
          obtener(i).getCiclo()+","+
          obtener(i).getPension());
      }
      pw.close();
      JOptionPane.showMessageDialog(null,"Operacion Exitosa");
      }catch(Exception ex){
```

```
            JOptionPane.showMessageDialog(null,"Grabacion Exitosa",
                                          .Aviso",
                                          JOptionPane.ERROR_MESSAGE);
        }
       }
 }
```

El siguiente script muestra las librerías y declaración de las variables globales de la clase frmMantenimiento.

```
package pFormularios;

import pClases.ArregloEstudiantes;
import javax.swing.JOptionPane;
import javax.swing.table.TableColumn;
import pClases.Estudiante;

public class frmMantenimiento extends javax.swing.JFrame {

    //Objeto de la clase ArregloEstudiantes
    ArregloEstudiantes a=new ArregloEstudiantes();

    public frmMantenimiento() {
        initComponents();
        //Define el ancho de las columnas de la tabla
        defineAnchoTabla();
        //Llenar el control cboCiclo
        llenaCombo();
        //Permite mostrar los registros contenido en el archivo
        //Estudiantes.txt
        listar();
        btnGrabarI.setVisible(false);
        btnGrabarM.setVisible(false);
        habilitaCajas(false);
    }
```

El siguiente script muestra los métodos de la clase frmMantenimiento.

```
    //Metodo que define los anchos de la tabla tAlumnos
     void defineAnchoTabla(){
         TableColumn columna;
         columna=tAlumnos.getColumnModel().getColumn(0);
         columna.setPreferredWidth(50);
         columna=tAlumnos.getColumnModel().getColumn(1);
         columna.setPreferredWidth(150);
         columna=tAlumnos.getColumnModel().getColumn(2);
         columna.setPreferredWidth(30);
         columna=tAlumnos.getColumnModel().getColumn(3);
         columna.setPreferredWidth(40);
         tAlumnos.getTableHeader().setReorderingAllowed(false);
         tAlumnos.getTableHeader().setResizingAllowed(false);
     }
```

```
//Metodo que llena los ciclos en el control cboCiclo
void llenaCombo(){
    cboCiclo.addItem(.1");
    cboCiclo.addItem(.2");
    cboCiclo.addItem(.3");
    cboCiclo.addItem(.4");
    cboCiclo.addItem(.5");
    cboCiclo.addItem(.6");
}

//Metodo que permite habilitar o ionhabilitar el acceso
//a los controles JTextField
void habilitaCajas(boolean opcion){
    txtCodigo.setEditable(opcion);
    txtNombres.setEditable(opcion);
    txtPension.setEditable(opcion);
    cboCiclo.setEnabled(opcion);
}

//Metodo que permite limpiar los controles JTextField
void limpiaCajas(){
    txtCodigo.setText(.");
    txtNombres.setText(.");
    txtPension.setText(.");
    cboCiclo.setSelectedIndex(0);
}

//Metodo que permite limpiar la tabla tAlumnos
void limpiaMatriz(){
    for(int i=0;i<10;i++){
        tAlumnos.setValueAt(., i, 0);
        tAlumnos.setValueAt(., i, 1);
        tAlumnos.setValueAt(., i, 2);
        tAlumnos.setValueAt(., i, 3);
    }
}

//Metodo que permite autogenerar un codigo del estudiante
//con un maximo de 4 cifras por numero autogenerado.
public int autogeneraCodigo(){
    int cod=(int)((9999-1000+1)*Math.random()+1000);
    return cod;
}

//Metodo que permite listar los registros de los estudiantes
//al control tAlumnos
void listar(){
    if (a.getTamaño()>0){
        for(int i=0;i<a.getTamaño();i++){
            Estudiante e=a.obtener(i);
            tAlumnos.setValueAt(e.getCodigo(), i, 0);
            tAlumnos.setValueAt(e.getNombre(), i, 1);
            tAlumnos.setValueAt(e.getCiclo(), i, 2);
            tAlumnos.setValueAt(e.getPension(), i, 3);
        }
    }
}

//Metodos Get
public int getCodigo(){
    return Integer.parseInt(txtCodigo.getText());
}
```

```
    public String getNombre(){
        return txtNombres.getText();
    }
    public double getPension(){
        return Double.parseDouble(txtPension.getText());
    }
    public int getCiclo(){
        return cboCiclo.getSelectedIndex()+1;
    }
```

El siguiente script muestra las instrucciones del botón **btnConsultar**.

```
private void btnConsultarActionPerformed(….)
{
  try{
     limpiaCajas();
     limpiaMatriz();

     int codigo=Integer.parseInt(JOptionPane.showInputDialog(this,
                                  .Ingrese codigo a Buscar:"));

     Estudiante e=a.buscar(codigo);
     if (e!=null){
        tAlumnos.setValueAt(e.getCodigo(), 0, 0);
        tAlumnos.setValueAt(e.getNombre(), 0, 1);
        tAlumnos.setValueAt(e.getCiclo(), 0, 2);
        tAlumnos.setValueAt(e.getPension(), 0, 3);
     } else
        JOptionPane.showMessageDialog(this,
                                    .Codigo de estudiante NO Existe");
   } catch(Exception ex){
       JOptionPane.showMessageDialog(this,"Error de Datos");
 }
}
```

El siguiente script muestra las instrucciones del botón **btnIngresar**.

```
private void btnIngresarActionPerformed(java.awt.event.ActionEvent evt) {
   int codigo=autogeneraCodigo();
   txtCodigo.setText("."+codigo);
   habilitaCajas(true);
   txtCodigo.setEditable(false);
   txtNombres.requestFocus();
   btnGrabarI.setVisible(true);
   btnIngresar.setVisible(false);
}
```

El siguiente script muestra las instrucciones del botón **btnModificar**.

```
private void btnModificarActionPerformed(…)
{
   int codigo=Integer.parseInt(JOptionPane.showInputDialog(this,
                        .Ingrese codigo de Estudiante a Modificar:"));
   habilitaCajas(true);
   limpiaMatriz();
   txtCodigo.setEditable(false);

   Estudiante e=a.buscar(codigo);
   if (e!=null){
      txtCodigo.setText("."+e.getCodigo());
      txtNombres.setText(e.getNombre());
      cboCiclo.setSelectedIndex(e.getCiclo()-1);
      txtPension.setText("."+e.getPension());

      txtNombres.requestFocus();
      btnGrabarM.setVisible(true);
      btnModificar.setVisible(false);
   } else
      JOptionPane.showMessageDialog(this,
                                    .Codigo de estudiante NO Existe");
}
```

El siguiente script muestra las instrucciones del botón **btnEliminar**.

```
private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
 try{
    limpiaCajas();
    limpiaMatriz();

    int codigo=Integer.parseInt(JOptionPane.showInputDialog(this,
                                .Ingrese codigo a Eliminar:"));
    Estudiante e=a.buscar(codigo);

    if (e!=null){
         a.eliminar(e);
         a.grabar();
         listar();
    }
    }catch(Exception ex){
       JOptionPane.showMessageDialog(this,
                                    .Codigo de estudiante NO Existe");
    }
}
```

El siguiente script muestra las instrucciones del botón **btnListar**.

```
private void btnListarActionPerformed(java.awt.event.ActionEvent evt) {
   listar();
}
```

El siguiente script muestra las instrucciones del botón **btnGrabarM**.

```
private void btnGrabarMActionPerformed(java.awt.event.ActionEvent evt) {
   Estudiante e=a.buscar(getCodigo());
   if (e != null){
       e.setNombre(getNombre());
       e.setCiclo(getCiclo());
       e.setPension(getPension());

       a.grabar();
       listar();
   }
   limpiaCajas();
   habilitaCajas(false);

   btnModificar.setVisible(true);
   btnGrabarM.setVisible(false);
}
```

El siguiente script muestra las instrucciones del botón **btnGrabarI**.

```
private void btnGrabarIActionPerformed(java.awt.event.ActionEvent evt) {
    Estudiante e=a.buscar(getCodigo());
    if (e == null){
         e=new Estudiante(getCodigo(),getNombre(),
                          getCiclo(),getPension());

         a.adicionar(e);
         a.grabar();
         listar();
    }
    limpiaCajas();
    habilitaCajas(false);

    btnIngresar.setVisible(true);
    btnGrabarI.setVisible(false);
}
```

Nota:

El archivo de texto Estudiantes.txt debe encontrarse dentro de la carpeta del proyecto como lo muestra la siguiente imagen.

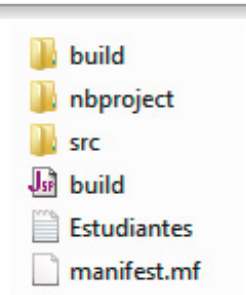


Fig. 12.11

Impreso en los Talleres Gráficos de

Surquillo
☎ 719-9700