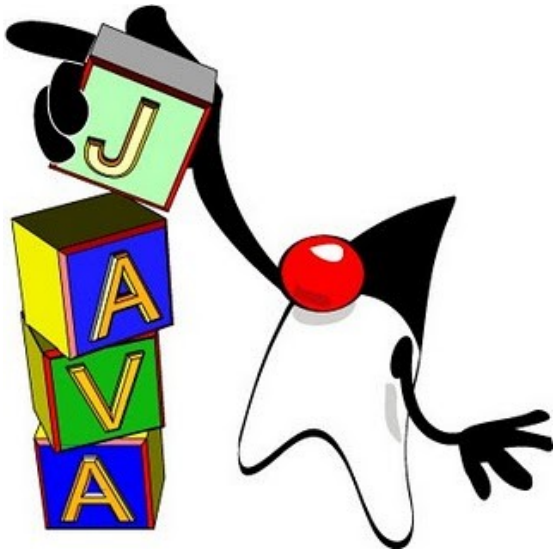
**UNIVERSIDAD NACIONAL DEL SANTA**

**FACULTAD DE INGENIERIA**

**E.A.P INGENIERIA DE SISTEMAS E INFORMATICA**




# MANUAL DE PROGRAMACION VISUAL CON JAVA

## PARTE - 1

*Ing. Mirko Manrique Ronceros*

**Primera Edición**

# INDICE

# INTRODUCCION

El presente documento tiene como objetivo fundamental servir como guía didáctica para la programación visual en java para los alumnos de la Escuela Académica Profesional de ingeniería de Sistemas e Informática de la Universidad Nacional del Santa.

Programación Visual está orientada al diseño de aplicaciones bajo entorno visual comúnmente a través del uso del formulario. En este manual se hará una presentación y estudio de las interfaces visuales a través del entorno de desarrollo denominado NetBeans, es decir, utilizaremos el lenguaje de programación Visual Java. El presente documento se encuentra dividida en partes: la primera es "Manipulación de controles", la segunda es "Diseño de formularios y Menús" y la tercera es "Sistemas de Aplicación".

En la primera parte se estudiará a los objetos de control básicos como son JLabel, JTextField, JButton, JRadioButton, JCheckBox, JList, JComboBox y Jtable; en la segunda parte se tendrá en cuenta el diseño de aplicaciones a partir de casos, lo cual implica el uso de los objetos de control básicos estudiados en la primera parte como también la construcción o diseño de menús; y en la tercera se abarcará la construcción o diseño de sistemas de aplicaciones haciendo uso de las sentencias selectivas, repetitivas y el uso de arreglos como también se verá el uso de interfaces multimedia y paquetes.

# Capítulo I

## RESUMEN

Por mucho tiempo los desarrolladores de software han hecho su trabajo usando lenguajes textuales de programación, pero eso está a punto de cambiar. en este artículo se presenta el paradigma de la programación visual y los lenguajes visuales de programación como una alternativa para mejorar la producción de aplicaciones de software.

## ¿Qué es Programación Visual?

El concepto de programación visual es un poco confuso ya que actualmente se le considera programación visual a los lenguajes de programación textual que tienen una interfaz gráfica para poder visualizar lo que uno esta desarrollando. Este concepto en programación visual es erróneo ya que este es aquel que por medio de iconos puedes ir creando programas sin tener un lenguaje textual atrás de el.

La **programación visual** (visual programming) se refiere al desarrollo de software donde las notaciones gráficas y los componentes de software manipulables interactivamente son usados principalmente para definir y componer programas.

La programación visual se define comúnmente como el uso de expresiones visuales (tales como gráficos, animación o iconos) en el proceso de la programación, pueden ser utilizadas para formar la sintaxis de los nuevos lenguajes de programación visuales que conducen a los nuevos paradigmas tales como programación por la demostración; o pueden ser utilizadas en las presentaciones gráficas del comportamiento o de la estructura de un programa.

El objetivo de la programación visual es mejorar la comprensión de los programas y simplificar la programación en sí. Más allá, la programación visual deberá fomentar a los usuarios finales a construir sus propios programas, que de otra forma deben ser escritos por programadores profesionales.

La programación visual brinda los conocimientos necesarios para diseñar y desarrollar aplicaciones con un entorno visual amigable y fácil de utilizar por el usuario. Los lenguajes de programación visual, como Visual Java, hacen sencilla la tarea de los programadores porque antes constituía una gran demora tiempo en el diseño de ventanas o formularios.

## Programación orientada a Objetos

En el caso del lenguaje de programación, Java aplica la programación orientada a objetos (POO). La POO define a los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (datos), comportamiento (procedimientos o métodos) e identidad (propiedad o atributo del objeto) que lo diferencia de los demás. La POO expresa un programa como un conjunto de objetos, que colaboran entre ellos para realizar tareas.

Podríamos decir que que las características de los objetos son:

- Los objetos se pueden agrupar para formar las clases.
- El estado de los objetos está determinado por los datos.
- Pueden heredar propiedades o atributos de otros objetos.
- Usando Mensajes un objeto se puede comunicar con otro objeto.
- Los métodos definen el comportamiento de los objetos.

## Bibliotecas de Clases en Java

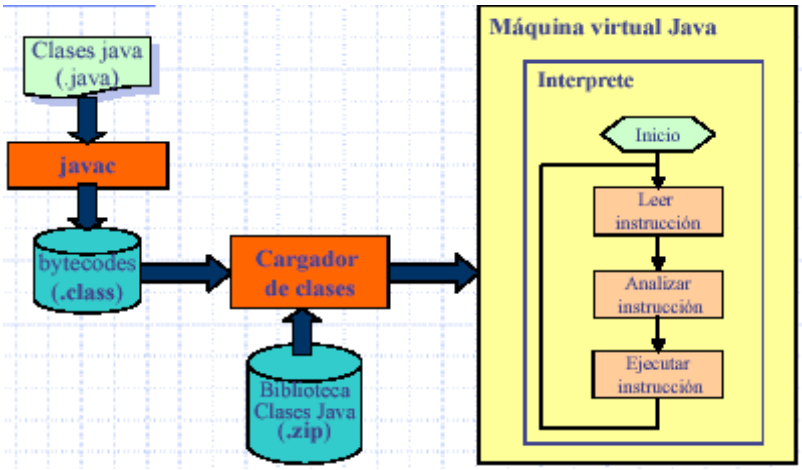
Sabemos que Java es un lenguaje de programación con un entorno de ejecución de aplicaciones como también entorno de ejecución de despliegue de aplicaciones. Es utilizado para desarrollar applets como aplicaciones.

Java está compuesto de bibliotecas de clases (package) siendo las más importantes:

- **Package Lang:** compuesta por clases con funcionalidades básicas, arrays, cadenas de caracteres, entrada/salidad, excepciones, etc. Este paquete debes haberlo utilizado en el curso de Fundamentos de Programación.
- **Package Util:** compuesta por clases para utilizadas como números aleatorios, vectores, propiedades del sistema, etc.
- **Package net:** compuesta por clases, es usada para la conectividad y trabajo con redes, URL, etc.

- **Package Applet**: compuesta por clases, es usada para el desarrollo de aplicaciones ejecutables en navegadores.
- **Package Awt y Swing:** compuesta por clases para el desarrollo de interfaces gráficas de usuario. El paquete swing es el paquete por excelencia para el desarrollo de los temas del presente curso.

Observa la siguiente figura:

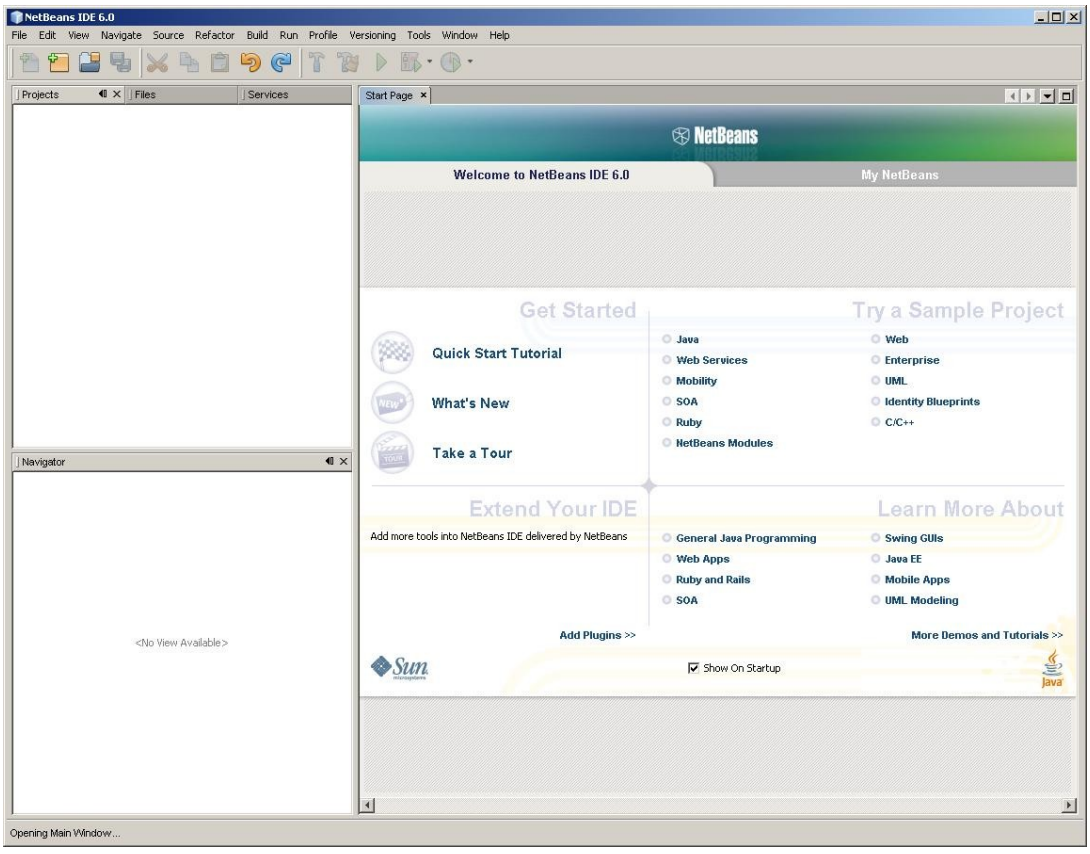



Cualquier programa hecho en Java lleva a definir un archivo de extensión .java. El programa debe pasar por un proceso de compilación que consiste en convertir tu programa fuente (el archivo de extensión .java) en un archivo de extensión .class y conjuntamente con la biblioteca de clases se logra interpretar lo programado, es así cuando ya se puede ejecutar el programa y ver los resultados en la pantalla del computador. Para este curso, en el que veremos programación visual y por lo tanto el uso de formularios, se creará un archivo adicional con extensión.frm que guardará la estructura o diseño del formulario.

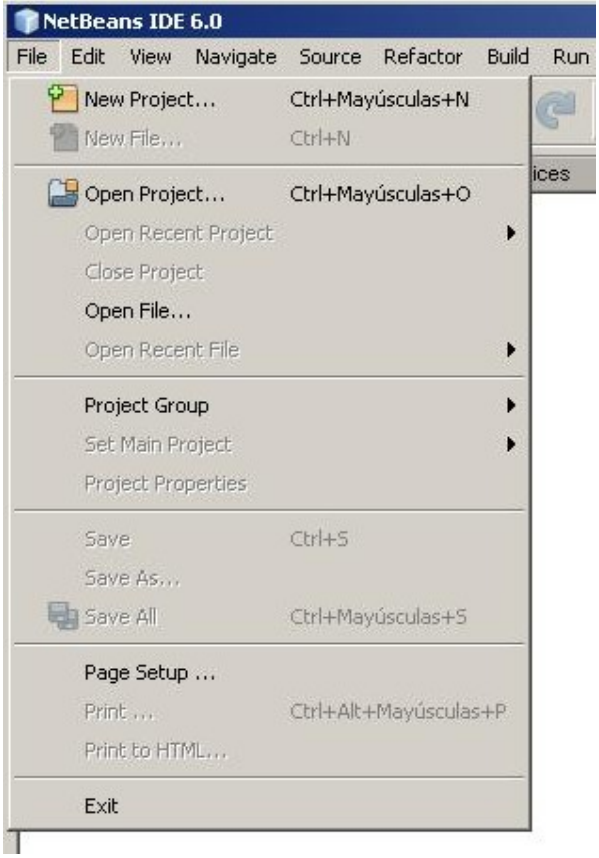
## NetBeans IDE en la Programación Visual

El NetBeans es un entorno de desarrollo integrado que permite crear aplicaciones de escritorio, aplicaciones Web y aplicaciones móviles utilizando las últimas tecnologías para los desarrolladores de software de Java. El IDE de NetBeans es un producto gratuito y sin restricciones de uso pudiendo escribir, compilar, depurar e implementar programas en Java. NetBeans es un proyecto open source de desarrollo escrito en Java. La plataforma NetBeans da soporte para escritura de servlets, ayuda on-line y ayudas con el código.

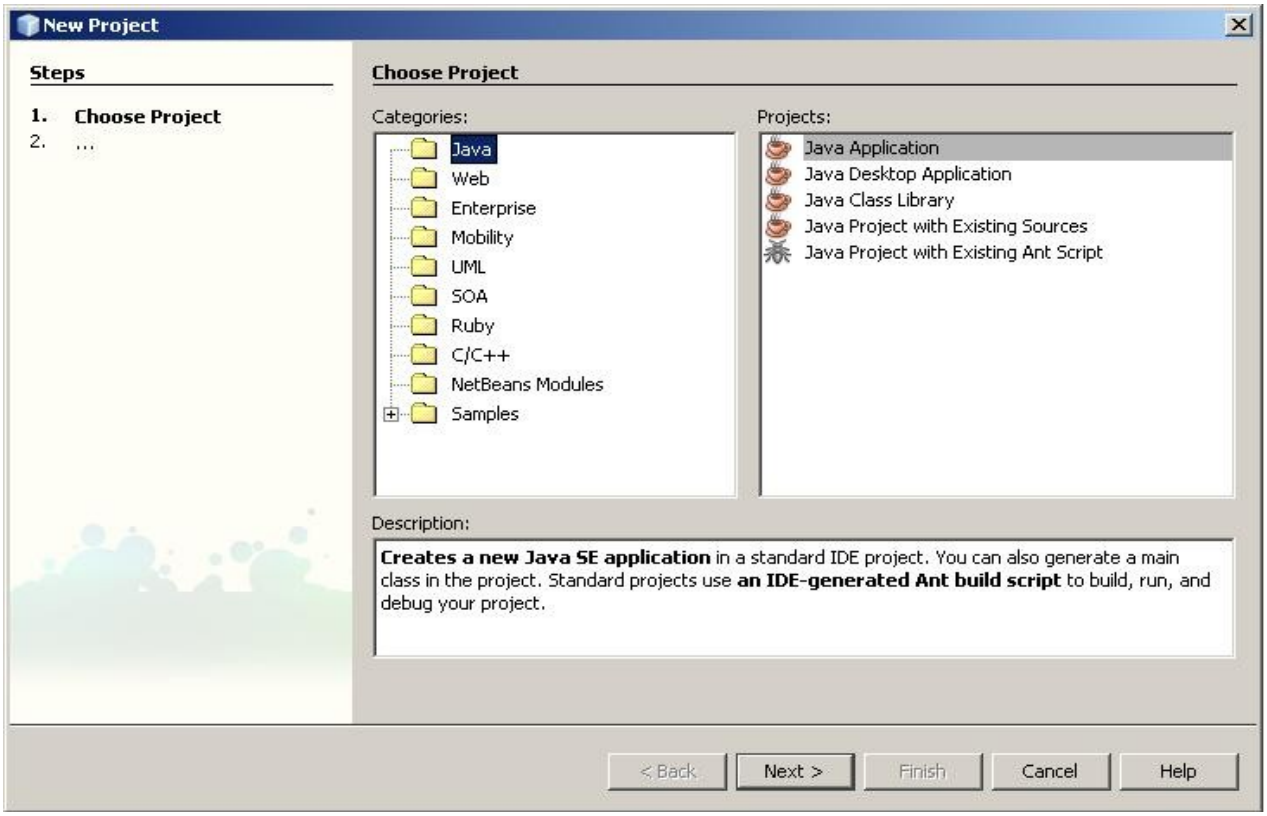
Usaremos la versión 6.0 de NetBeans para la construcción y diseño de las aplicaciones. Una vez que ingresas al entorno de desarrollo de NetBeans se observa:



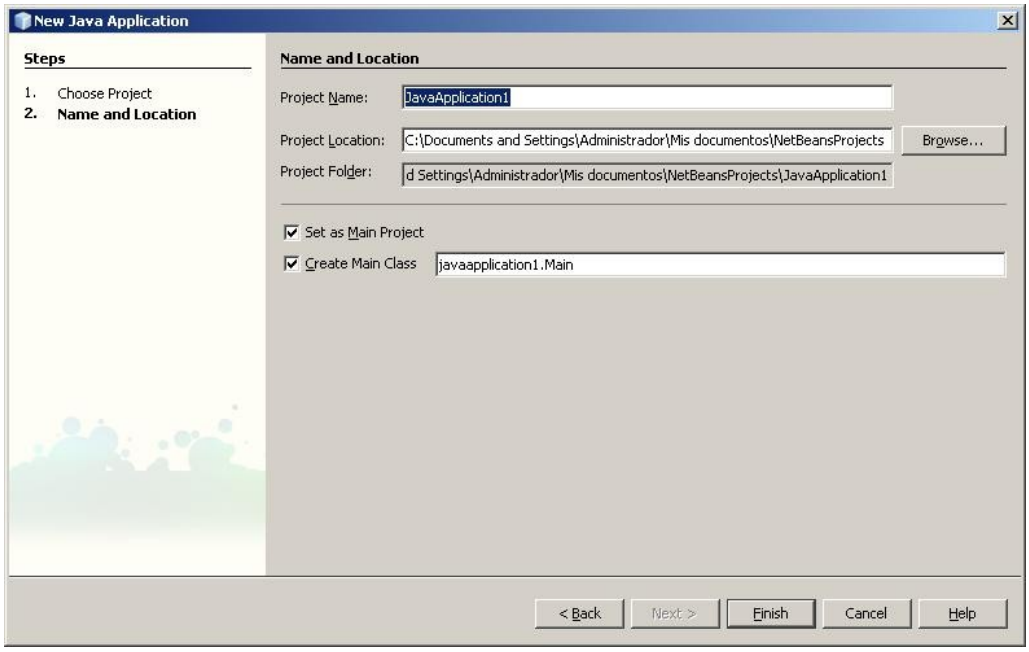

Para dar inicio a una aplicación de Java bajo el entorno de Netbeans se debe definir un proyecto, para ello, seleccionas la opción del menú denominada ***File.*** Se muestra inmediatamente un menú flotante cuya primera opción indica ***New Proyect***, está opción la seleccionas.




Al momento de seleccionar ***New Proyect*** se visualiza la ventana siguiente:

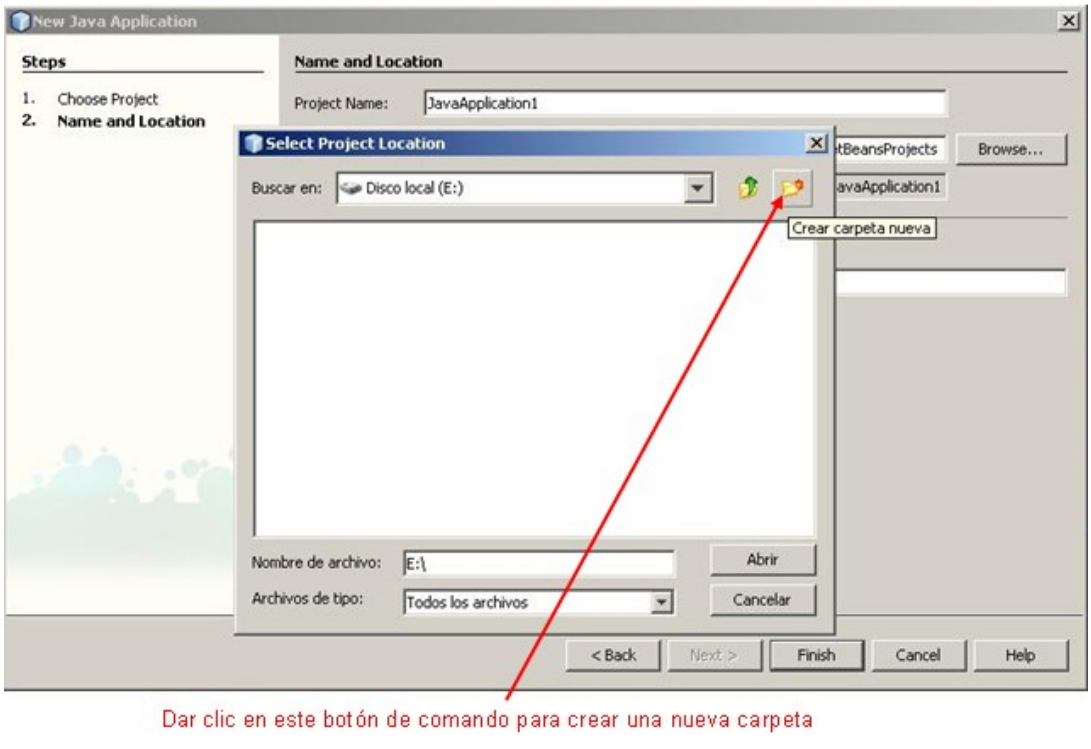


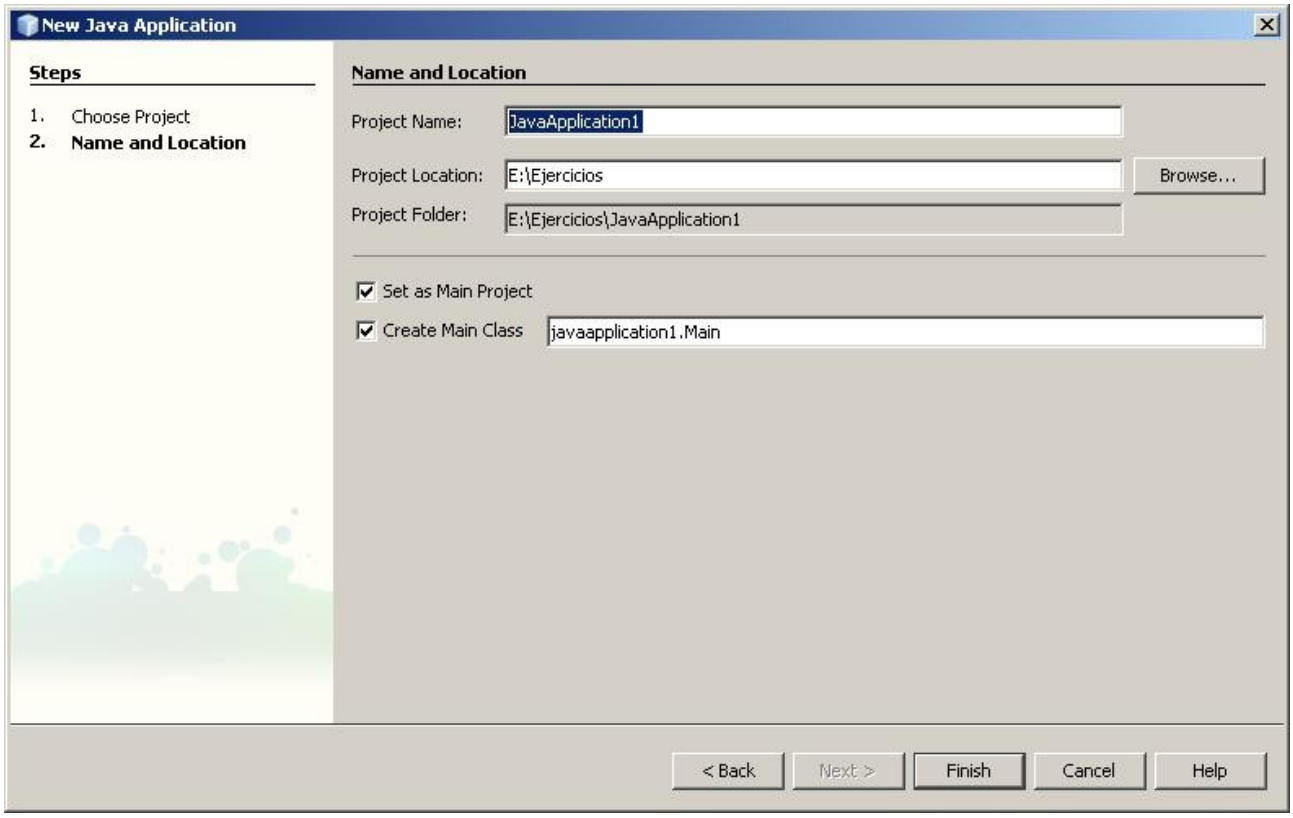

Dado que nuestras aplicaciones van ser desarrolladas en entorno visual en Categories seleccionas la carpeta Java y en Proyects seleccionas Java Application. Luego hacer click en el botón de comando ***Next*** que mostrará la siguiente ventana*:*



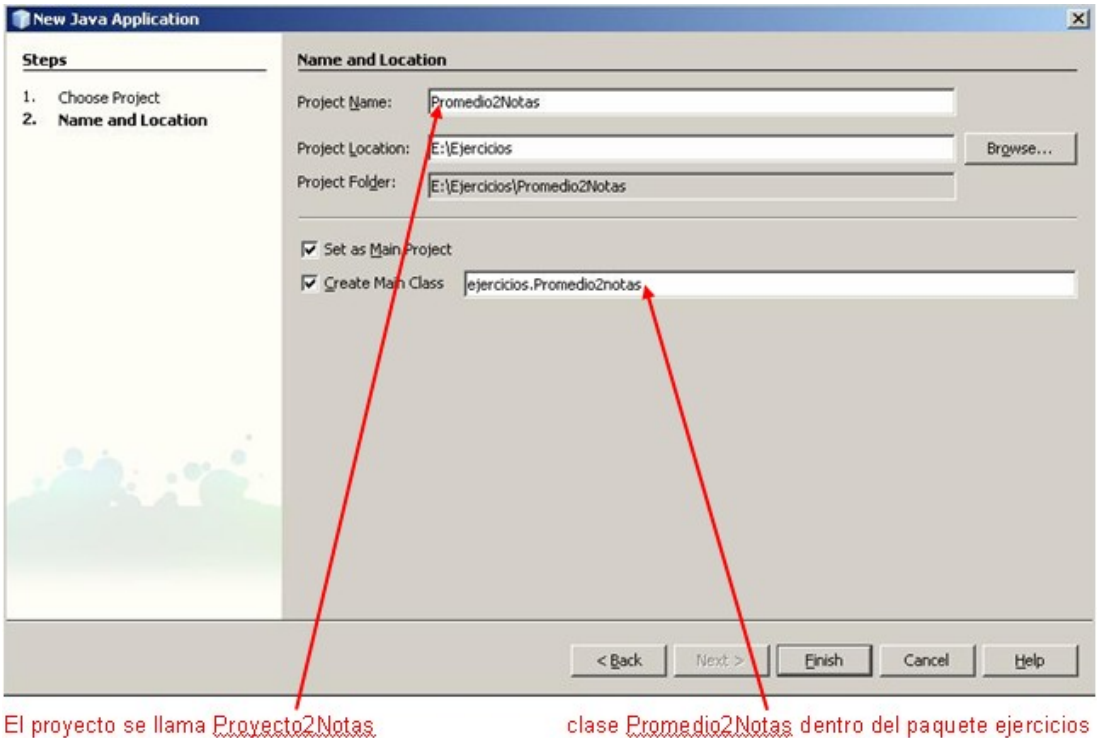

Es conveniente que uno mismo cree su carpeta de destino de los archivos que se generan para la construcción de una aplicación. Supongamos que la carpetas que necesitamos crear se llama Ejercicios y la creamos en la unidad E, para ello es necesario dar click en el botón de comando ***Browse***.

Una vez indicada la nueva carpeta Ejercicios, procede a dar click en el botón de comando ***Abrir*** quedando la ventana **New Java Application** de la siguiente forma:

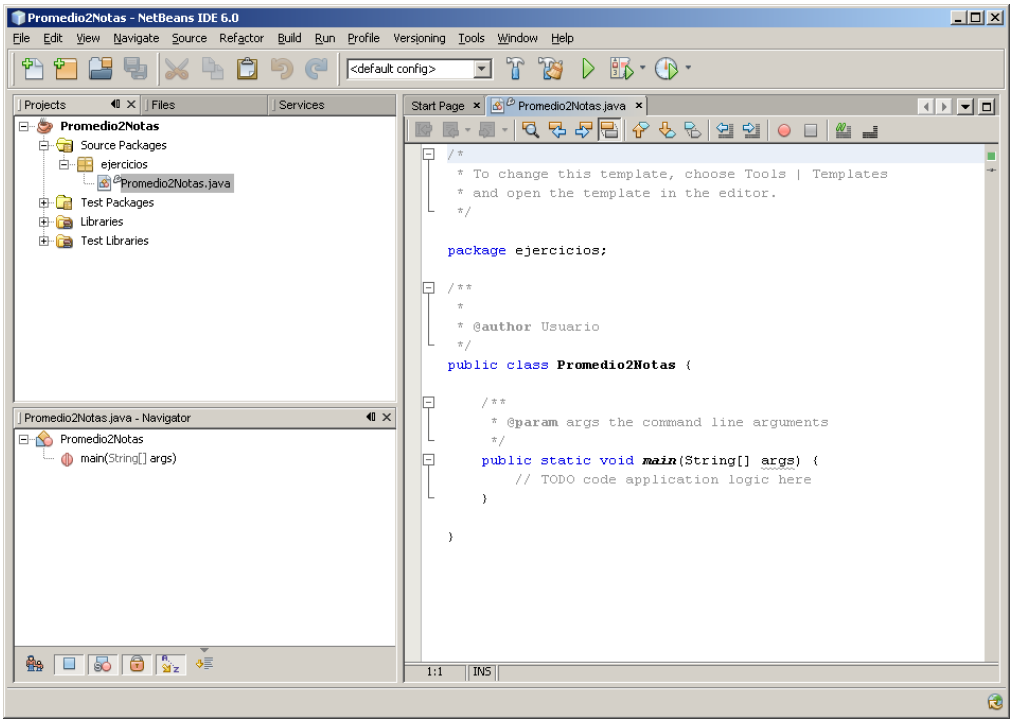




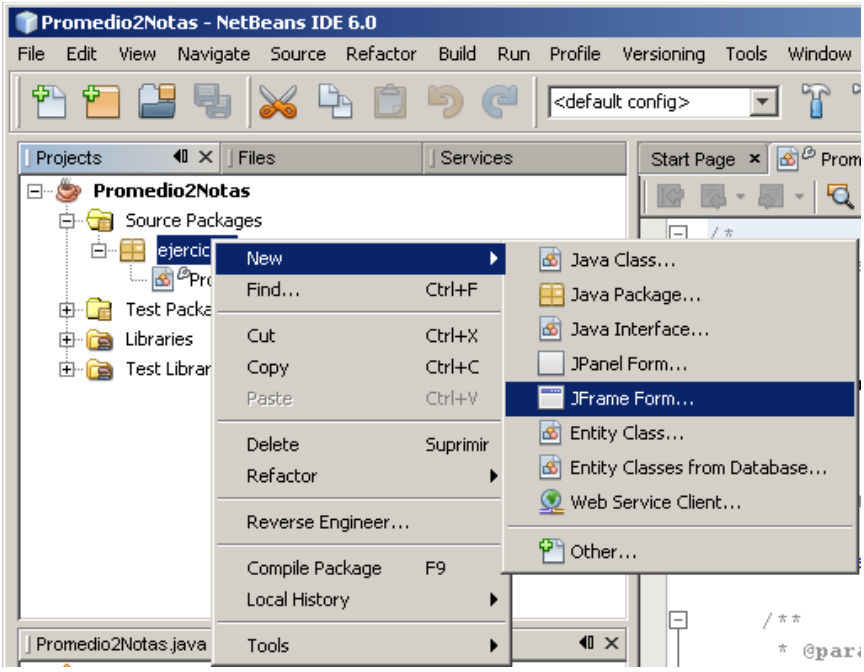

Se observa que en **Proyect Location** se muestra la carpeta destino del proyecto. Vamos a suponer que se quiere construir un programa que calcule el promedio de dos notas, entonces la ventana debería quedar de la siguiente forma:



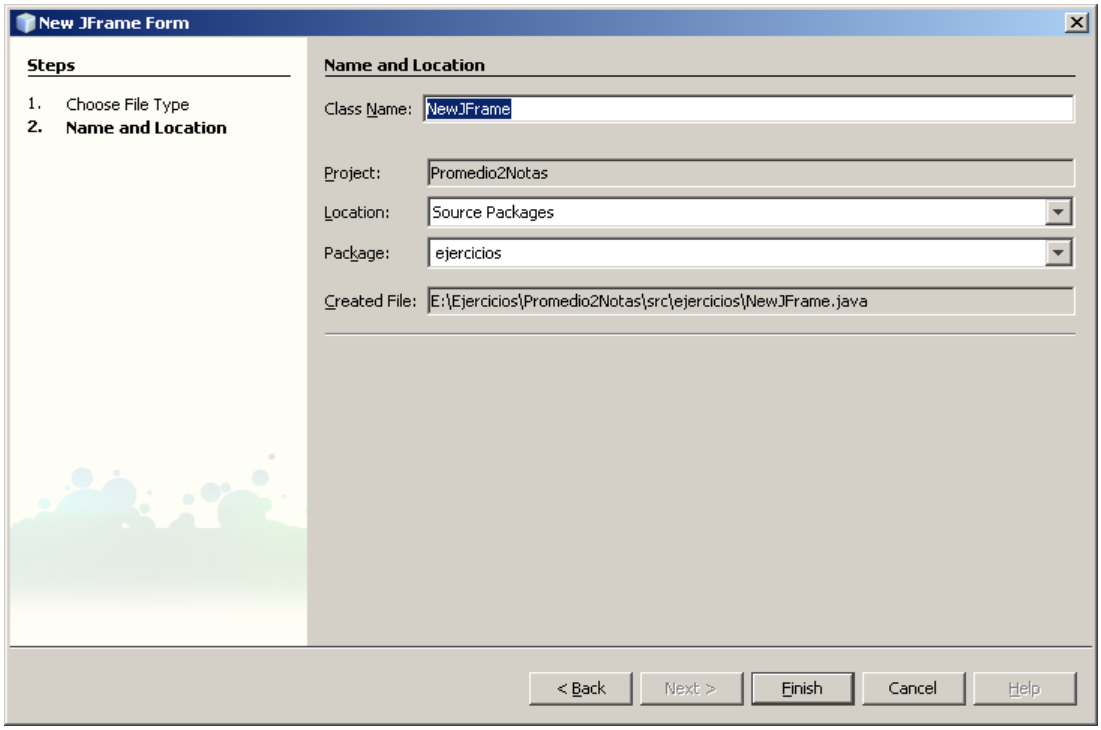
Al dar click en ***Finish*** se mostrará el entorno de desarrollo de NetBeans listo para dar inicio a la construcción de la aplicación.




Cuando se pretenda construir una aplicación de entorno visual tendríamos que usar plantillas que el mismo NetBeans IDE te ofrece. Ahora, seleccionemos el paquete ejercicios y luego elijamos la opción ***New*** y a continuación seleccionemos ***JFrameForm***



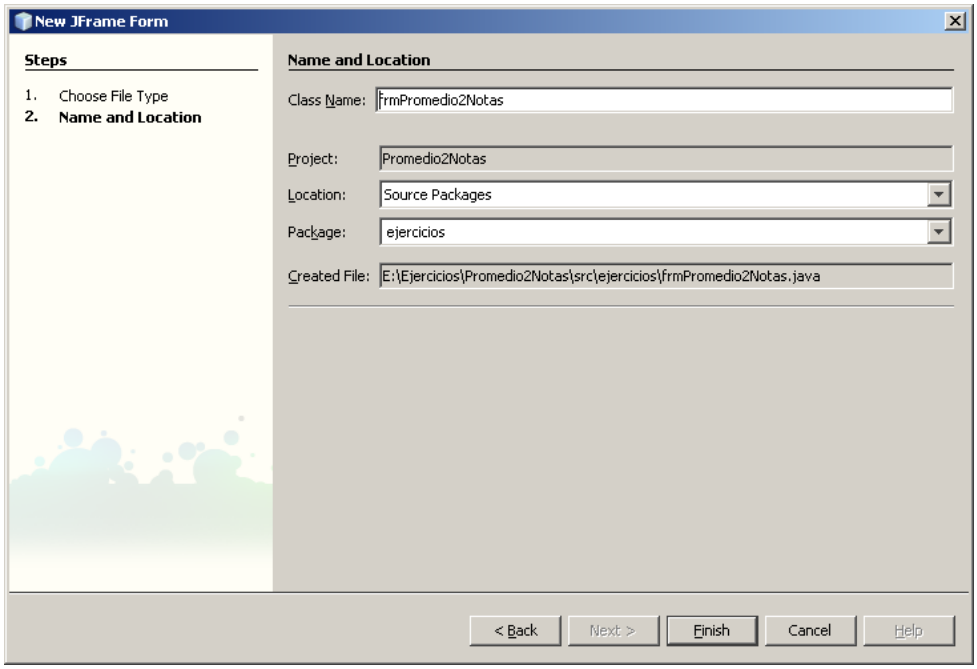
Una vez seleccionada la opción JframeForm se muestra la ventana **New Jframe Form** para definir el nombre de clase.



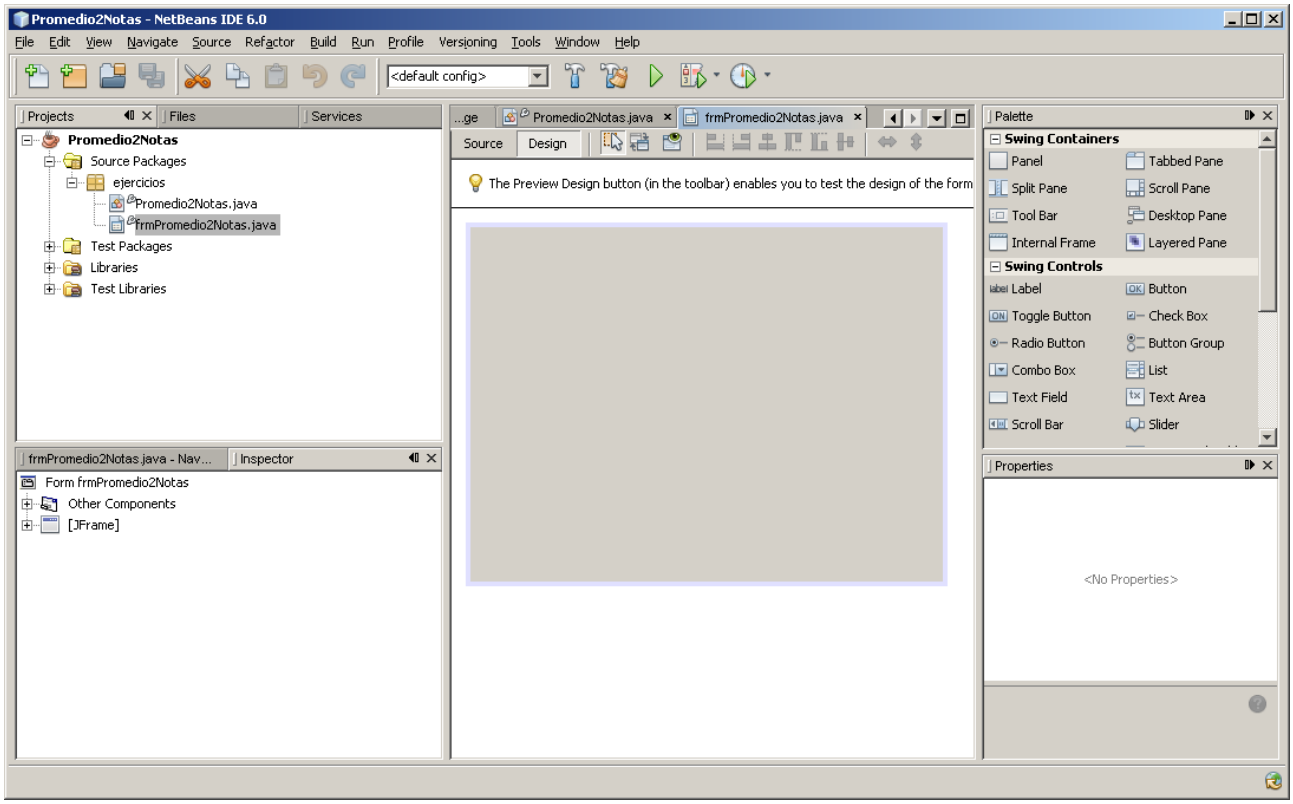

Como nombre de clase le pondremos ***frmPromedio2Notas*** esto generará un archivo de extensión .java dentro de la ruta:

***E:\Ejercicios\Promedio2Notas\src\ejercicios\frmPromedio2Notas.java***

y como veremos todo se encuentra dentro de la carpeta Ejercicios creada inicialmente en la unidad E



Al dar click en el botón de comando ***Finish*** se mostrará el entorno de desarrollo de NetBeans listo para dar inicio al diseño de un formulario y por lo tanto dar comienzo a una aplicación en un entorno visual.



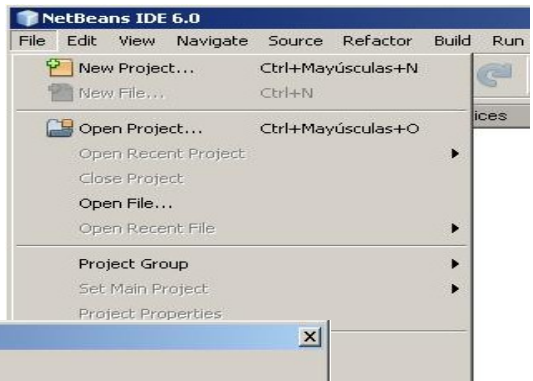
# USO DE LOS OBJETOS JLABEL, JTEXTFIELD Y JBUTTON

## Una aplicación sin usar Formulario

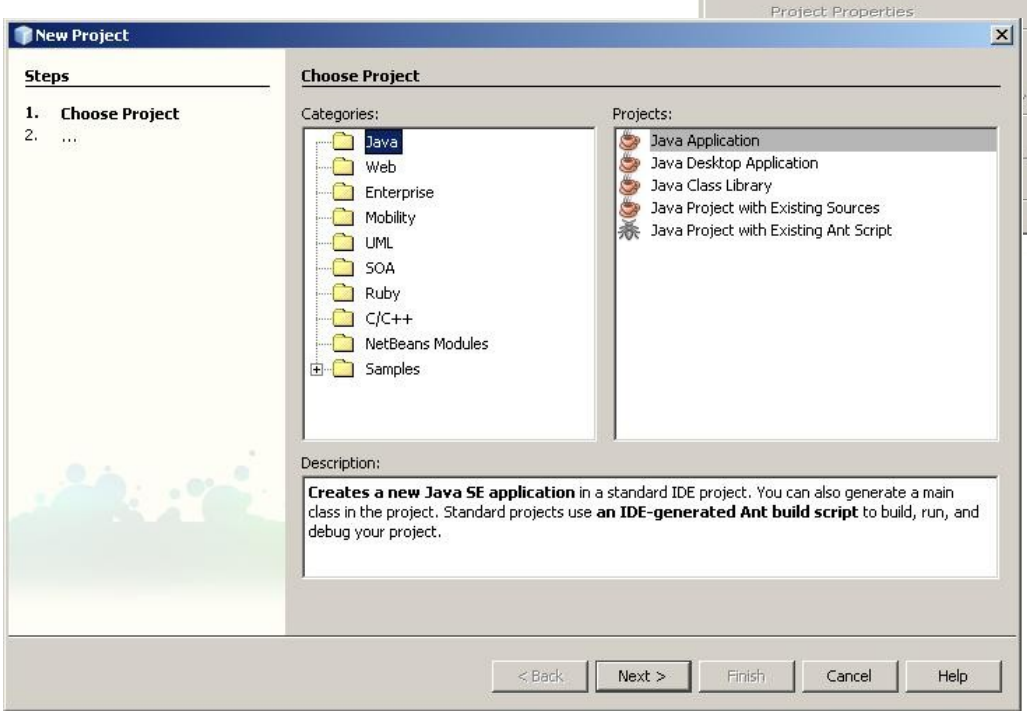
A continuación vamos a desarrollar una aplicación sencilla que permita calcular el área del triángulo dado los valores de la base y la altura. No se utilizará formulario, pero sí las clases del paquete swing para ingresos y salida de datos.

## Solución:

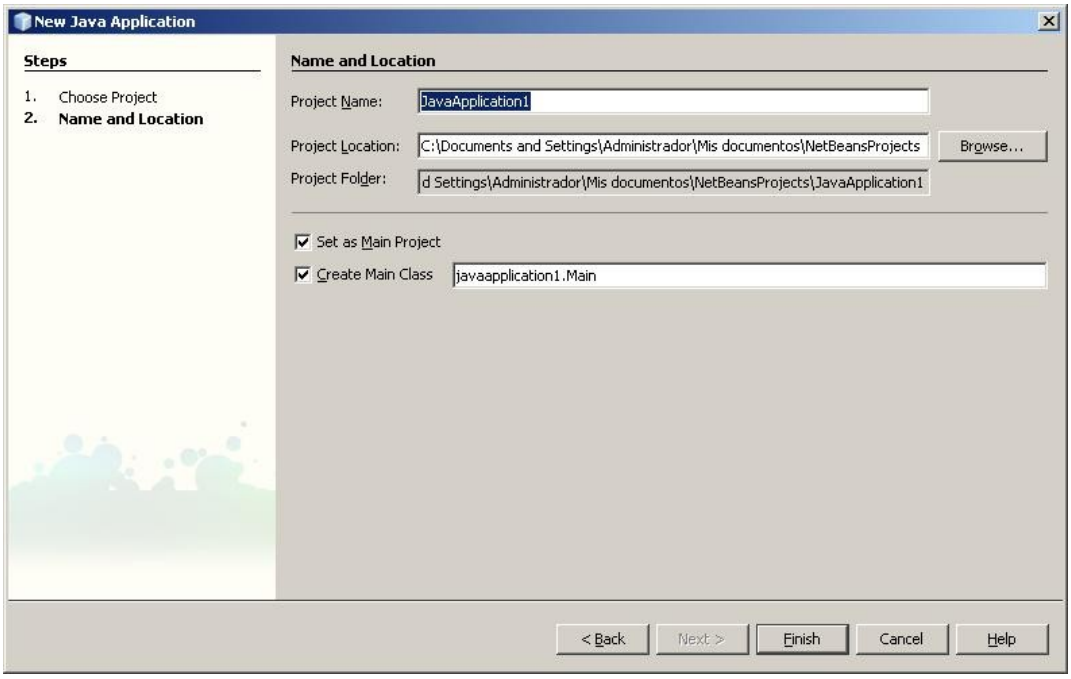
- Estando en el entorno de desarrollo NetBeans seleccionamos la opción del menú y luego la opción ***New Proyect***.



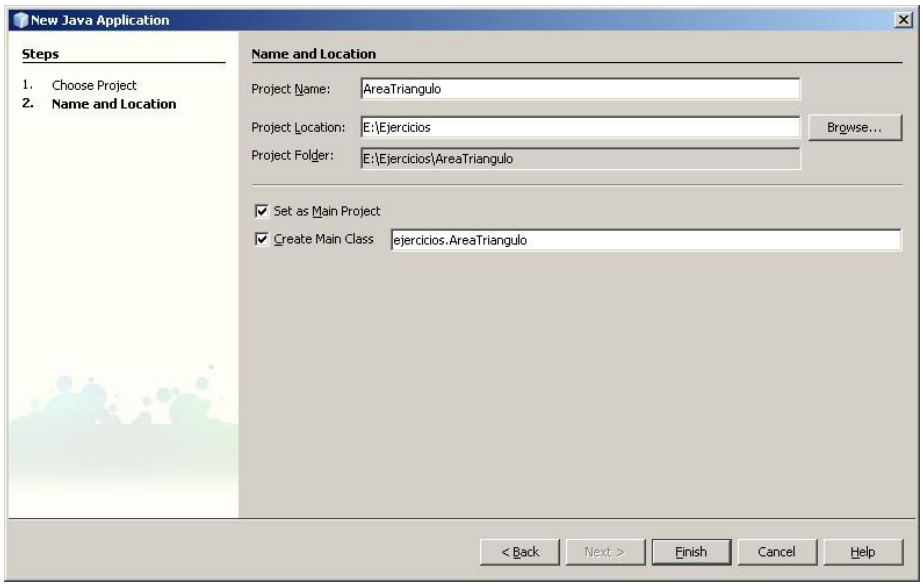

- Al momento de seleccionar New Proyect se visualiza la siguiente ventana:



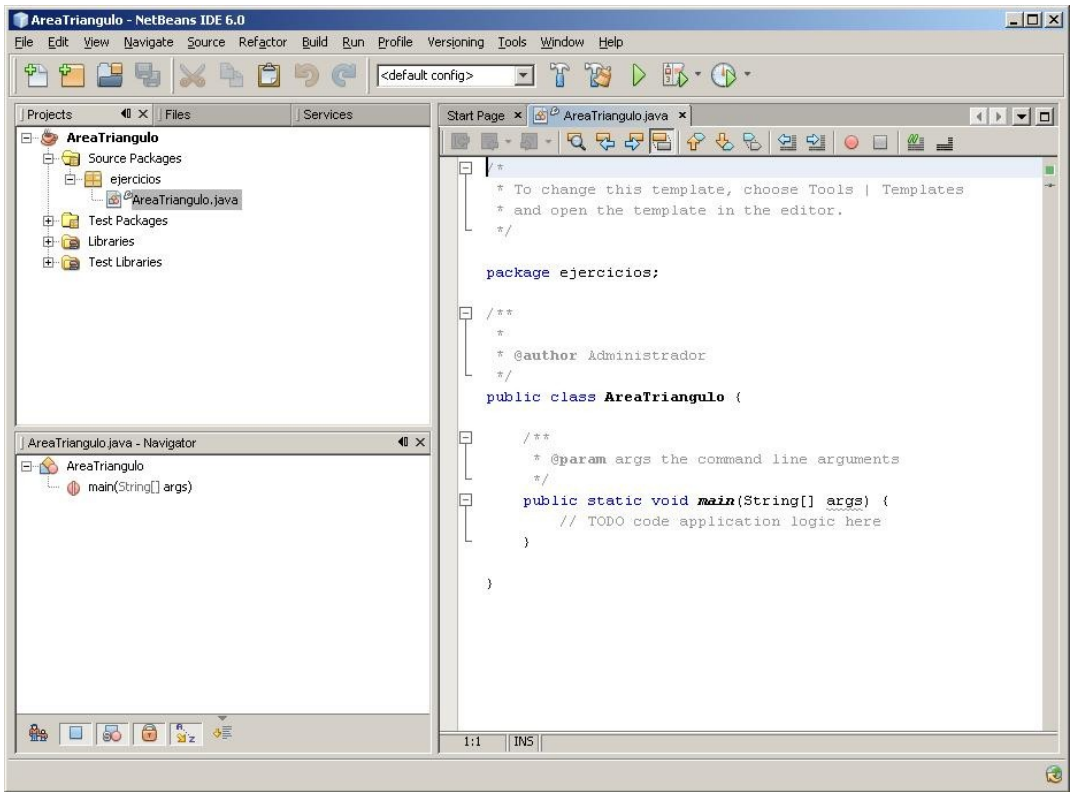
- Seleccionamos para ***Categories*** Java y para Proyects ***Java Application*** y luego damos click en el botón de comando ***Next***.




Usamos el botón de comando ***Browse*** para crear una carpeta denominada Ejercicios en la unidad E. En ***Proyect Name*** colocamos AreaTriangulo y en el cuadro de texto referido a ***Create Main Class*** colocamos *ejercicios.AreaTriangulo*, lo cual permitirá crear un paquete denominado *ejercicios* y como primera clase AreaTriangulo, es decir, se crea el archivo AreaTriangulo.java que pertenecerá al paquete ***ejercicios***.



- Sabemos que en Java una clase tiene comúnmente al método Main que es el método que se ejecuta cuando se aplica ***Run*** a la aplicación construida. En el entorno de NetBeans, una clase creada, crea un método que tiene el mismo nombre de la clase a la que se denomina *método constructor* y toda programación hecha en éste método se ejecutará primero antes que el método ***main***. NetBeans generará un paquete (Package) denominado ejercicios y dentro de ella se mostrará la clase de acceso público *AreaTriangulo*. El entorno de desarrollo de NetBeans después de dar click en el botón ***Finish*** queda así:




- Agregamos al código de la programación generada, por debajo del package ejercicios, lo siguiente:

```
import java.io.*;
import javax.swing.*;
```

El paquete javax.swing.*; permitirá poder crear ingresos y salida de datos a través de cajas de mensaje.

- Agregamos al código en el método Main de la clase AreaTriangulo como se aprecia a continuación (lo escrito en azul).

```
package ejercicios;
import java.io.*;
import javax.swing.*;

/**
*
* @author Administrador
*/
public class AreaTriangulo {

/**
* @param args the command line arguments
*/
public static void main(String[] args) throws IOException
{
int base, altura;
double area;
base=Integer.parseInt(JOptionPane.showInputDialog(null,"Ingrese la base: "));
altura=Integer.parseInt(JOptionPane.showInputDialog(null,"Ingrese la altura"));
area=base*altura/2;
JOptionPane.showMessageDialog(null,"El area del triangulo es: "+area);
}
```

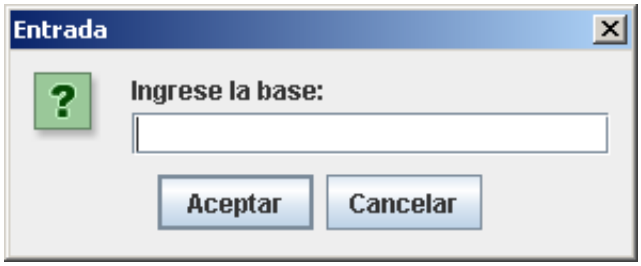
}

En el método main observamos que para leer el dato para la variable **base** utilizamos el método *showInputDialog* de la clase *JOptionPane* que a su vez pertenece al paquete *swing,* que mostrará un caja de mensaje para la lectura de datos. De igual manera se trabajará para leer el dato de altura.

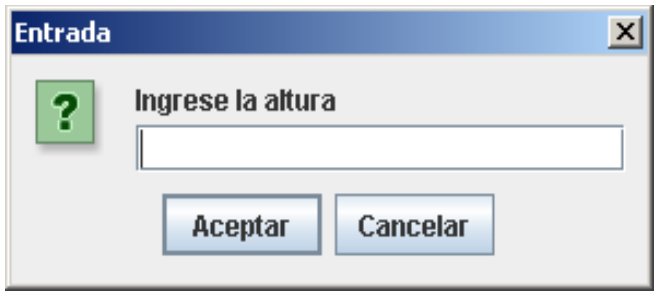
También se observa el método showMessageDialog de la clase JOptionPane que permite mostrar en una caja de mensaje el resultado del cálculo del área.

- Seleccionamos AreaTriangulo en el entorno de desarrollo de NetBeans y luego damos click botón derecho del Mouse. Se visualiza un menú flotante, se elige la opción ***Run File*** y se procederá a ejecutar el programa.

- Cuando la aplicación es ejecutada se visualizará la siguiente ventana:




Nos pide el ingreso del valor de la base y luego de dar click en el botón de comando ***Aceptar*** se mostrará la siguiente ventana:




Posteriormente se mostrará la ventana del resultado del cálculo del área.

## Objeto de control JLabel

Un objeto de control Jlabel permite dibujar en el formulario una etiqueta, entendiéndose como etiqueta una expresión estática que se quiere colocar. También es usado para mostrar los resultados de un proceso.

**Propiedades más usadas:**

- Text: Contiene el valor que se visualizará en el formulario.
- Font: Permite establecer el tipo de letra de la expresión a mostrar en el formulario.
- Border: Para establecer el tipo de borde de la etiqueta.
- Enabled: Para habilitar o inhabilitar el uso del objeto de control.

## Objeto de control JTextField

Un objeto de control JTextField permite dibujar en el formulario un cuadro de texto, es decir, una caja que permite la introducción de un dato o valor. Este objeto es utilizado para el ingreso de datos.

**Propiedades más usadas:**

- Text: Contiene el valor o dato introducido en el cuadro de texto.
- Font: Permite establecer el tipo de letra del texto en la caja.
- Border: Para establecer el tipo de borde del cuadro de texto.
- Enabled: Para habilitar o inhabilitar el uso del objeto de control.

**Métodos más usados:**

- getText(): Permite obtener el texto introducido en el cuadro de texto.
- setText(): Permite colocar un texto en el objeto JTextField.
- requestFocus(): permite asignar el cursor al objeto de control

## Objeto de control JButton

Un objeto de control JButton permite dibujar en el formulario un objeto que contiene un proceso a ejecutar. Se utiliza comúnmente para llevar a cabo procesos específicos según la naturaleza de la aplicación.

**Propiedades más usadas:**

- Text: Contiene el valor o dato introducido en el cuadro de texto.
- Font: Permite establecer el tipo de letra del texto en la caja.
- Enabled: Para habilitar o inhabilitar el uso del objeto de control.

**Evento más usado:**

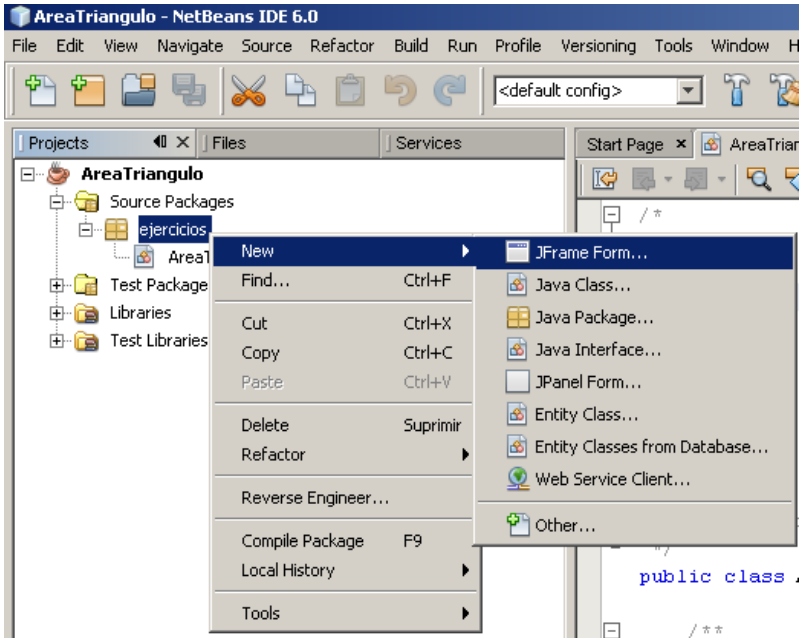
- ActionPerformed: Este evento se lleva a cabo cuando el usuario da click sobre el objeto de control JButton.

## Una aplicación usando los objetos de control

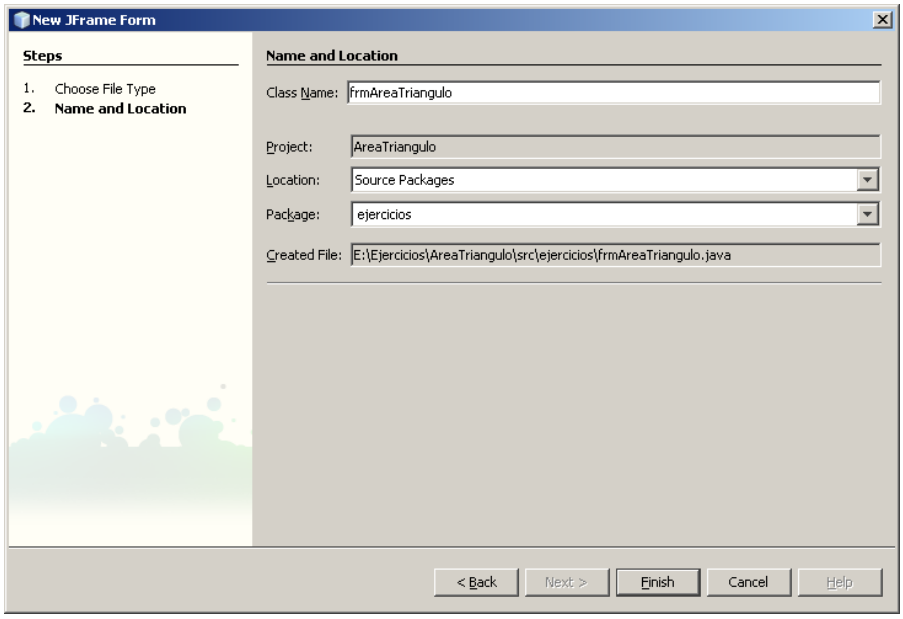
Ahora procedamos a desarrollar la misma aplicación usando como interfase un formulario y los objetos de control antes mencionado.

**Solución:**

- Seleccionamos el paquete ejercicios y damos click botón derecho del mouse y eligimos la opción ***New*** y posteriormente ***JframeForm***.

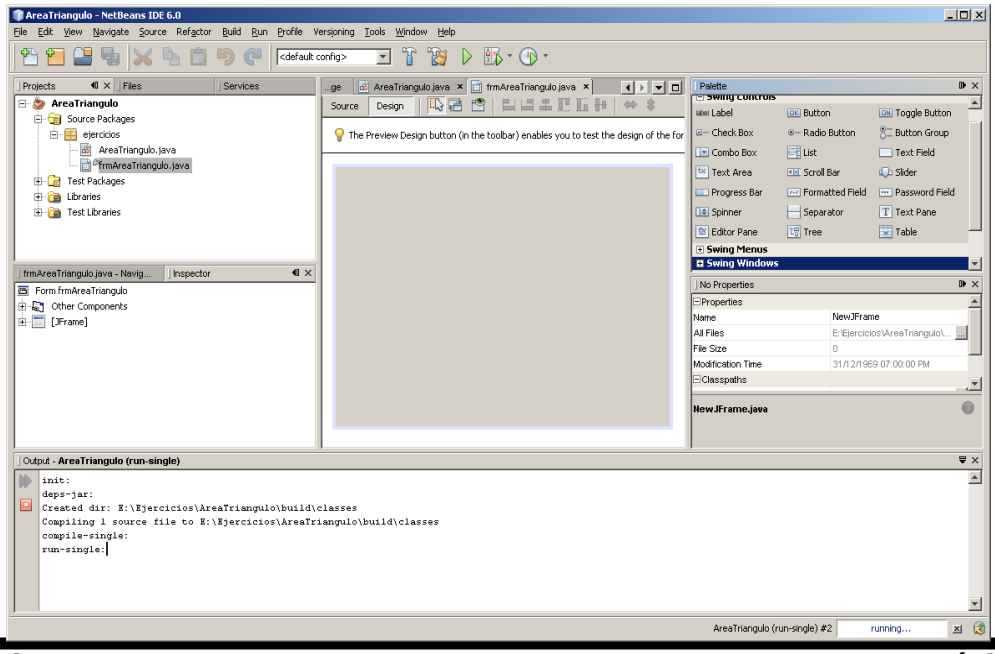



- Se muestra la ventana ***New JFrame Form*** y colocamos en Class Name el nombre del formulario: **frmAreaTriangulo**.

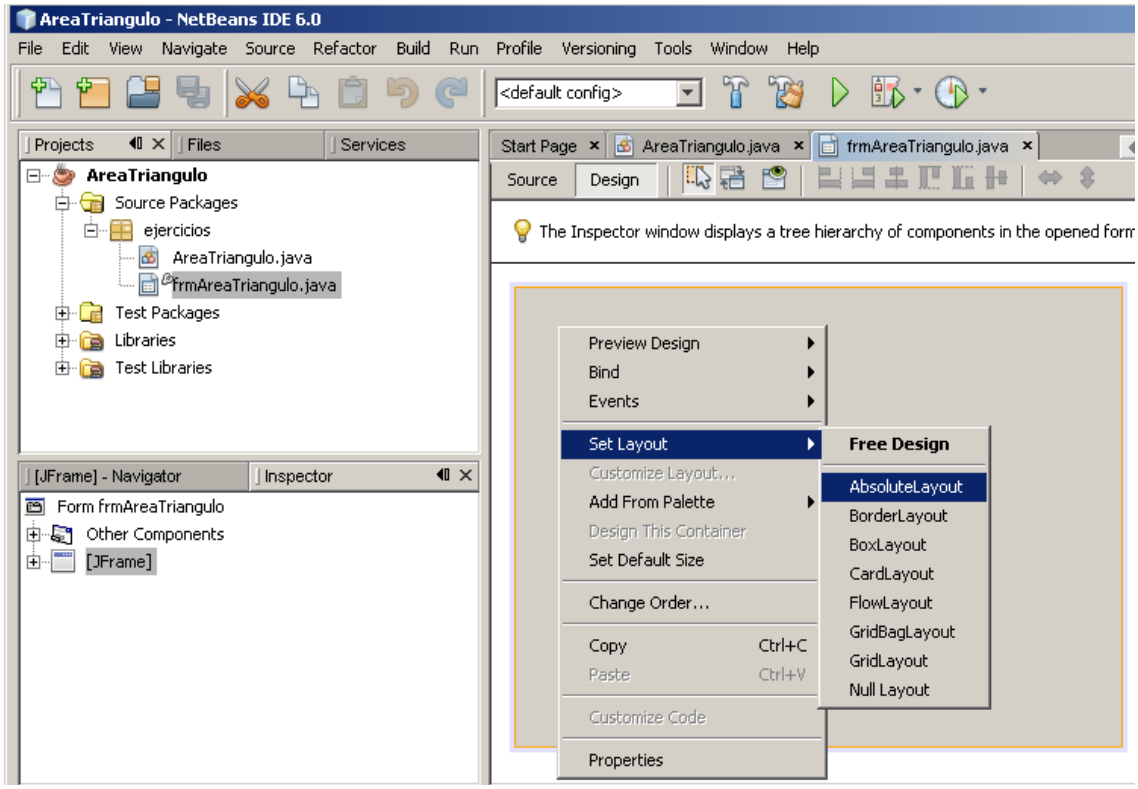



En esta ventana se observa que la clase denominada ***frmAreaTriangulo*** generará un archivo de extensión .java denominado *frmAreaTriangulo* que se almacenará dentro de la carpeta *ejercicios* y pertenecerá al paquete ejercicios.

- Al momento de dar click en el botón de comando ***Finish*** se visualizará el entorno de desarrollo NetBeans y al lado derecho se muestra la paleta de los objetos de control (Swing Controls).

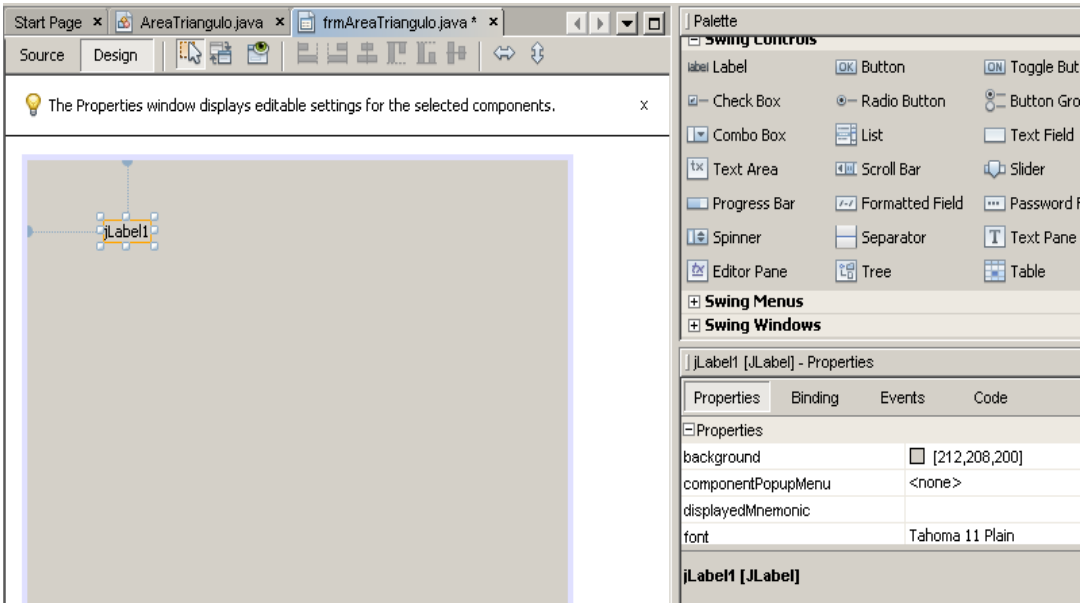


- Sobre el diseño del formulario damos click botón derecho y seleccionamos ***Set Layout*** y posteriormente ***AbsoluteLayout***.

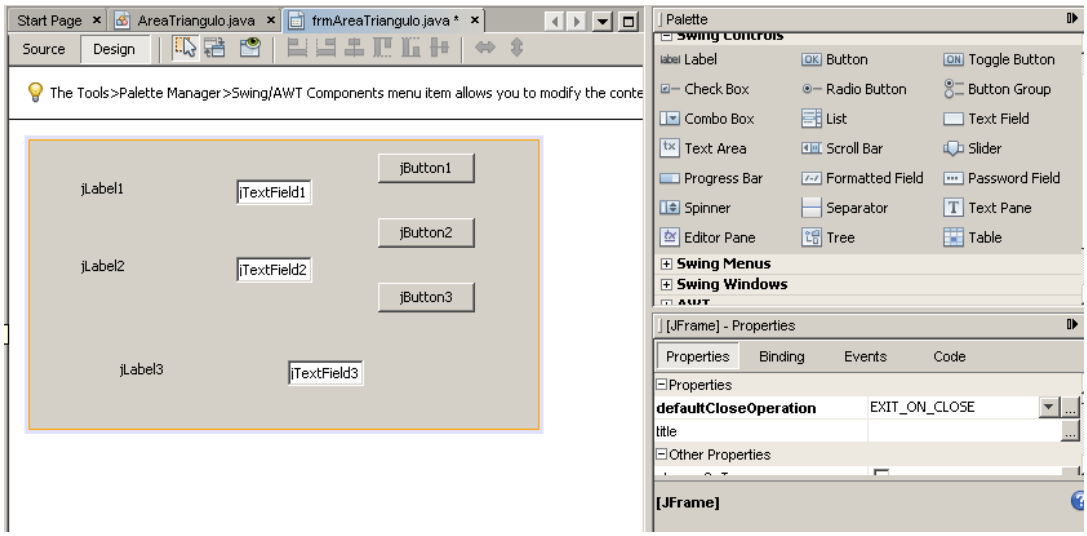



Es necesario usar AbsoluteLayout para que permita dibujar los objetos de control en el lugar donde uno quiere en el formulario.

- Ahora procedamos a colocar un objeto Jlabel seleccionando de la paleta Swing Controls Label y lo arrastramos hacia el diseño del formulario.



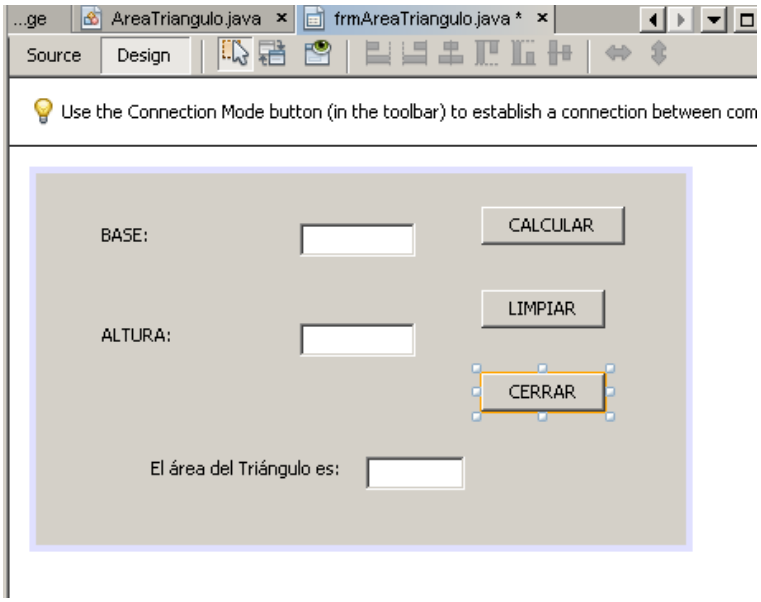
- Continuamos el diseño del formulario, quedando éste de la siguiente manera:



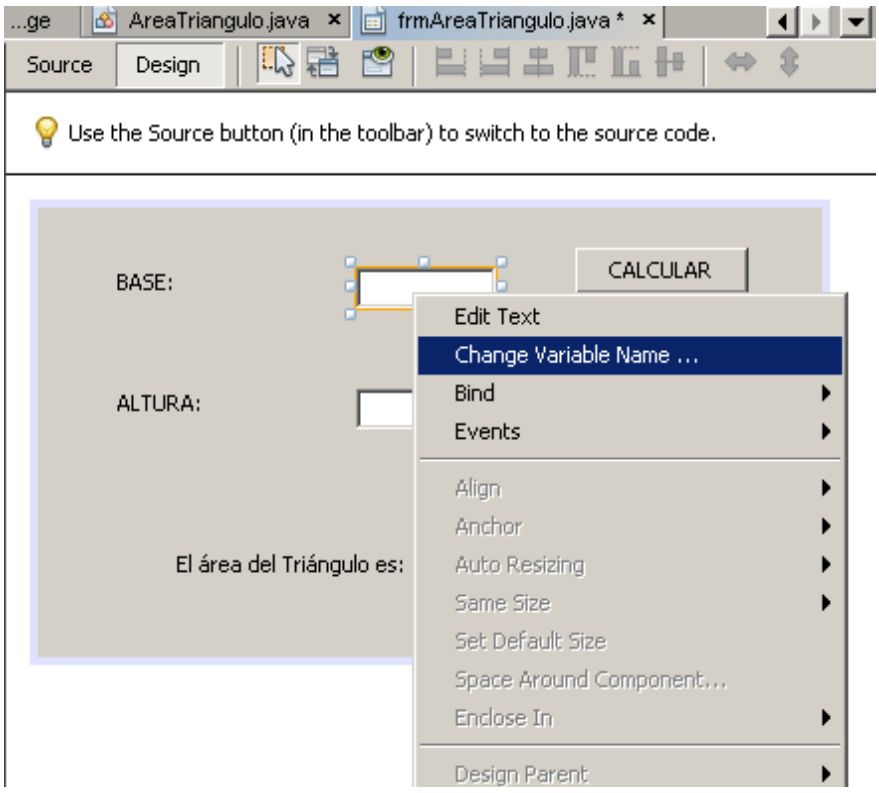

- Ahora procedamos a usar a cambiar los valores de las propiedades de los objetos de control en la ventana de propiedades:

| Objeto de Control | Propiedad | Valor de la Propiedad |
|---|---|---|
| JLabel1 | Text | BASE: |
| JLabel2 | Text | ALTURA: |
| JLabel3 | Text | El área del Triángulo es: |
| JTextField1 | Text | *(Vacío o limpiar)* |
| JTextField2 | Text | *(Vacío o limpiar)* |
| JTextField3 | Text | *(Vacío o limpiar)* |
| JButton1 | Text | CALCULAR |
| JButton2 | Text | LIMPIAR |
| JButton3 | Text | CERRAR |

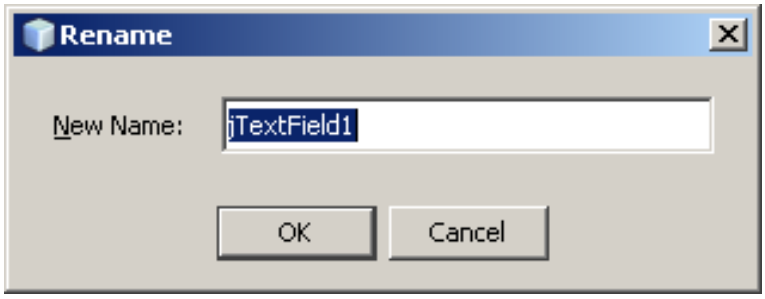
Luego de aplicar los cambios en los valores de propiedades el diseño del formulario debe quedar de la siguiente manera:

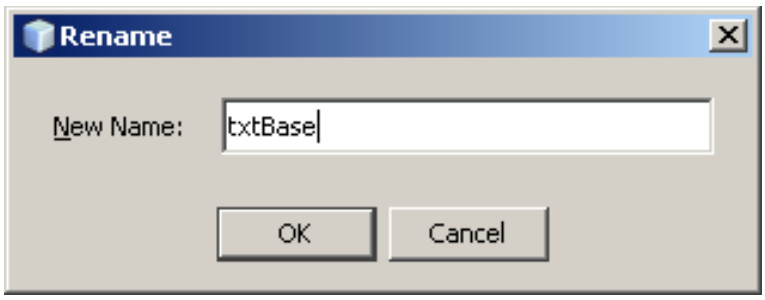



- Es necesario darle un nombre propio a los objetos de control y sobretodo a aquellos que intervienen en la lógica de la programación. Seleccionamos el objeto de control JTextField1 y damos click con el botón derecho del mouse y se visualizará un menú flotante y seleccionamos ***Change Variable Name***.



Colocamos como nombre al objeto JtextField1: **txtBase**




Continuamos con los objetos de control siguientes:

| Objeto de Control | Nombre |
|---|---|
| JTextField2 | txtAltura |
| JTextField3 | txtArea |
| JButton1 | btnCalcular |
| JButton2 | btnLimpiar |
| JButton3 | btnCerrar |

- Ahora procedamos a programar en los botones de comando.
  En el botón CALCULAR (btnCalcular) al darle doble click y escribir el siguiente código: (lo escrito en azul)

```
private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt)
{
int base, altura;
double area;
base=Integer.parseInt(txtBase.getText());
altura=Integer.parseInt(txtAltura.getText());
```

```
area=base*altura/2;
txtArea.setText(String.valueOf(area));
}
```

Una vez declaradas las variables de memoria, a la variable **base** se le asigna el valor introducido en el cuadro de texto **txtBase.** El método ***getText()*** permite obtener el dato introducido y con el método ***parseInt*** de la clase Integer es convertido a numérico entero. Se hace lo mismo para la variable altura. Para mostrar el cálculo de área se utiliza el método setText del cuadro de texto txtArea que permite visualizar el contenido de la variable **area.** A la variable se le aplica el método valueOf de la clase String para convertir el dato área en cadena de caracteres.

En el botón LIMPIAR (btnLimpiar) luego de darle doble click escribimos el siguiente código: (lo escrito en azul)

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt)
{
txtBase.setText("");
txtAltura.setText("");
txtArea.setText("");
txtBase.requestFocus();
      }
```

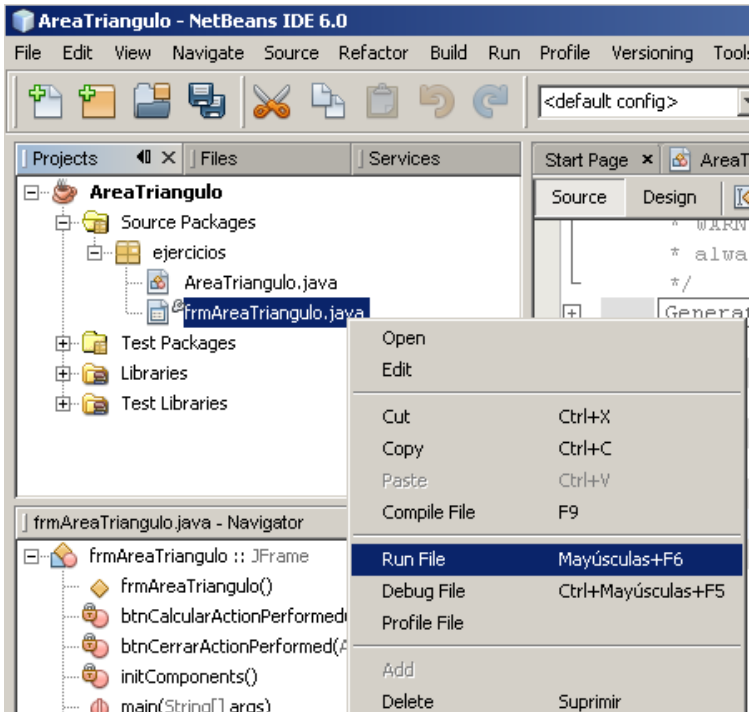
Se limpian los cuadros de textos a través del método setText() y con el método requestFocus() se pasa el cursor al objeto de control txtArea.

En el botón CERRAR (btnCerrar) luego de darle doble click, escribimos el siguiente código: (lo escrito en azul)

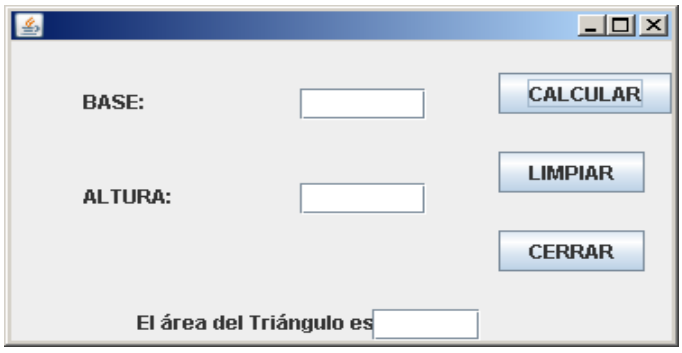
```
private void btnCerrarActionPerformed(java.awt.event.ActionEvent evt)
{
dispose();
}
```

El método ***dispose()*** permite descargar el formulario y terminar la ejecución de la aplicación.

- Procedemos a ejecutar la aplicación seleccionado frmAreaTriangulo y al dar clic botón derecho eligimos ***Run File***.




Se muestra el formulario diseñado en etapa de ejecución.



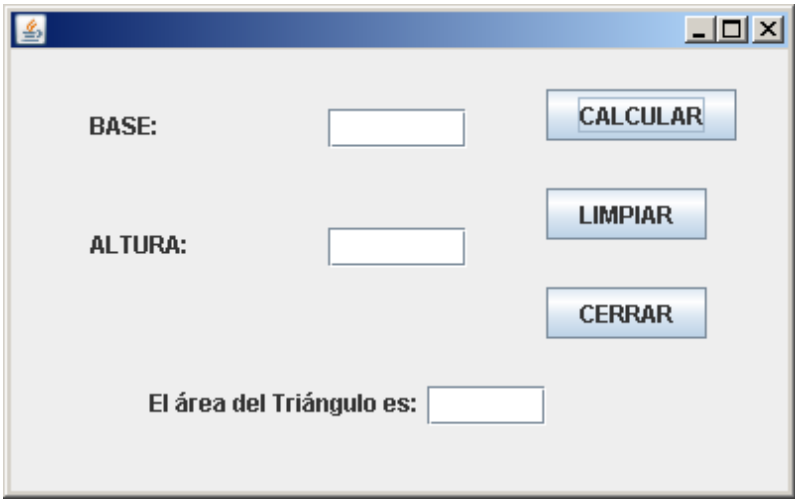

- Podemos observar que el formulario sale con los objetos de control cercano a los bordes del formulario y el mismo formulario se visualiza pegado en la parte superior izquierda de la pantalla del computador. Vamos a proceder a corregir estos defectos agregando dos líneas de código en el método constructor de la clase frmAreaTriangulo. (escribe lo que está en azul)

```
public class frmAreaTriangulo extends javax.swing.JFrame
{
/** Creates new form frmAreaTriangulo */
```

```
public frmAreaTriangulo()
{
initComponents();
setSize(400,250);
setLocation(250,250);
        }
```

El método setSize() permite establecer el tamaño del formulario y el método setLocation() permite ubicar el formulario dentro de la pantalla. Los métodos antes mencionados pertenecen al formulario frmAreaTriabgulo. Otros métodos del formulario se verán más adelante.

- Procedemos nuevamente a ejecutar el formulario frmAreaTriangulo y se mostrará de la siguiente manera:



# USO DE LOS OBJETOS JRADIOBUTTON Y JCHECKBOX

## Objeto de Control JRadioButton

Un objeto de control JRadioButton permite dibujar en el formulario una opción que puede ser seleccionada, es decir, es un objeto que define una opción o alternativa para ser elegida. Este objeto debe mostrarse más de una vez en el diseño del formulario para que exista la alternativa de seleccionar una opción de un grupo de opciones. Los objetos son mutuamente excluyentes con respeto a la selección. Se tiene la siguiente figura:


En la figura anterior se observan tres objetos JRadioButton y para poder seleccionar sola una alternativa se tendrá que usar el objeto ButtonGroup que es un elemento que no se llega a dibujar en el formulario pero permite agrupar objetos JRadioButton y una vez agrupados permite la selección de una opción cuando la aplicación se encuentre en ejecución. En las dos aplicaciones que veremos más adelante se hará hincapié en este asunto.

### Propiedades más usadas:

- Text: Permite establecer la expresión de la opción.
- Font: Permite establecer el tipo de letra en el objeto de control.
- Enabled: Para habilitar o inhabilitar el uso del objeto de control.

### Método más usado:

- isSelected(): retorna el valor de verdadero si el objeto se encuentra seleccionado y falso en caso contrario.

## Objeto de Control JCheckBox

Un objeto de control JCheckBox permite dibujar en el formulario una opción que puede ser seleccionada, es decir, es un objeto que define una opción o alternativa para ser elegida. La diferencia con respecto al objeto de control JRadioButton es que si se tienen dos o más objetos JCheckBox se puede seleccionar más de una opción o simplemente no seleccionar ninguna, por lo que no son mutuamente excluyentes. Se tiene la siguiente figura:


En la figura anterior se observa un objeto JCheckBox que expresa como opción Contado, si es seleccionado significa que la forma de pago es al contado y si se deja como no seleccionado significa que la forma de pago no es al contado por lo que se puede asumir que es al crédito.

**Propiedades más usadas:**

- Text: Permite establecer la expresión de la opción.
- Font: Permite establecer el tipo de letra en el objeto de control.
- Enabled: Para habilitar o inhabilitar el uso del objeto de control.

**Método más usado:**

- isSelected(): retorna el valor de verdadero si el objeto se encuentra seleccionado y falso en caso contrario.

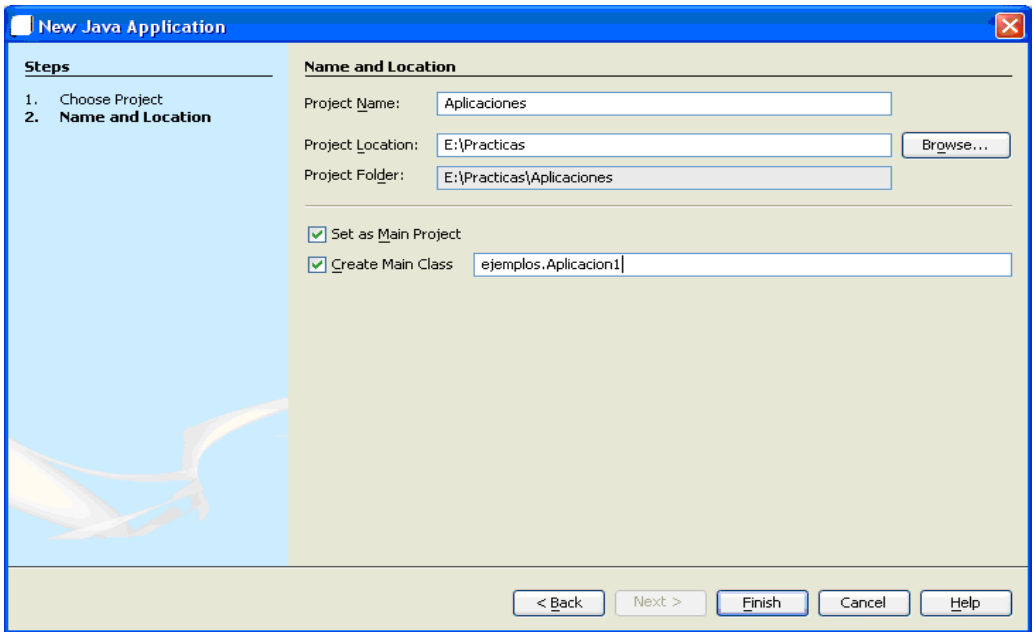
### Aplicación 1

Vamos a construir una aplicación en entorno visual que permita ingresar del costo de un artefacto, del tipo de artefacto y la forma de pago, para calcular lo siguiente:

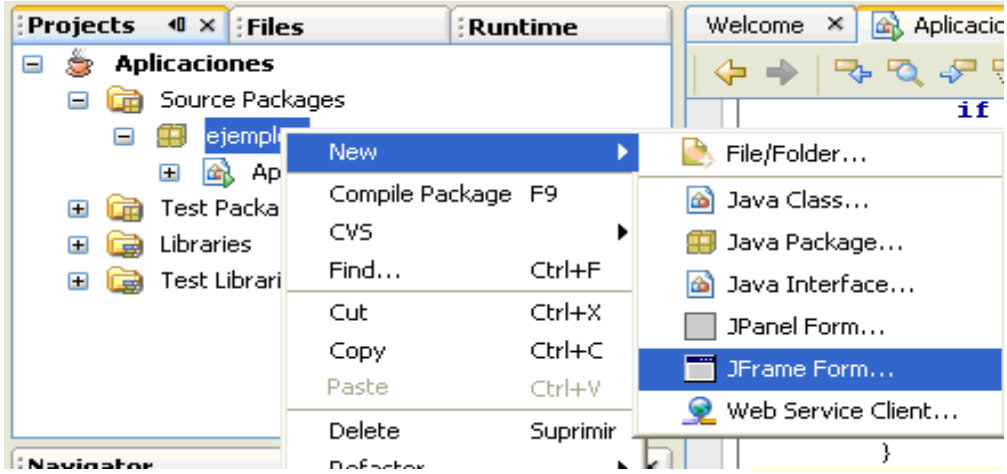
a. Si el pago es al contado hay un descuento del 6% del costo del artefacto si el tipo de artefacto es Audio, 8% si es Video y 5% si es Línea Blanca.
b. Si el pago es al crédito hay un incremento del 7% sobre el costo del artefacto si es el tipo de artefacto es Audio, 9% si es Video y 10% si es Línea Blanca.
c. El monto del IGV es del 19% sobre el costo del artefacto luego de aplicar el descuento o el incremento.
d. El monto a pagar que es el costo del artefacto (descontado o incrementado) más el monto del IGV.

**Solución:**

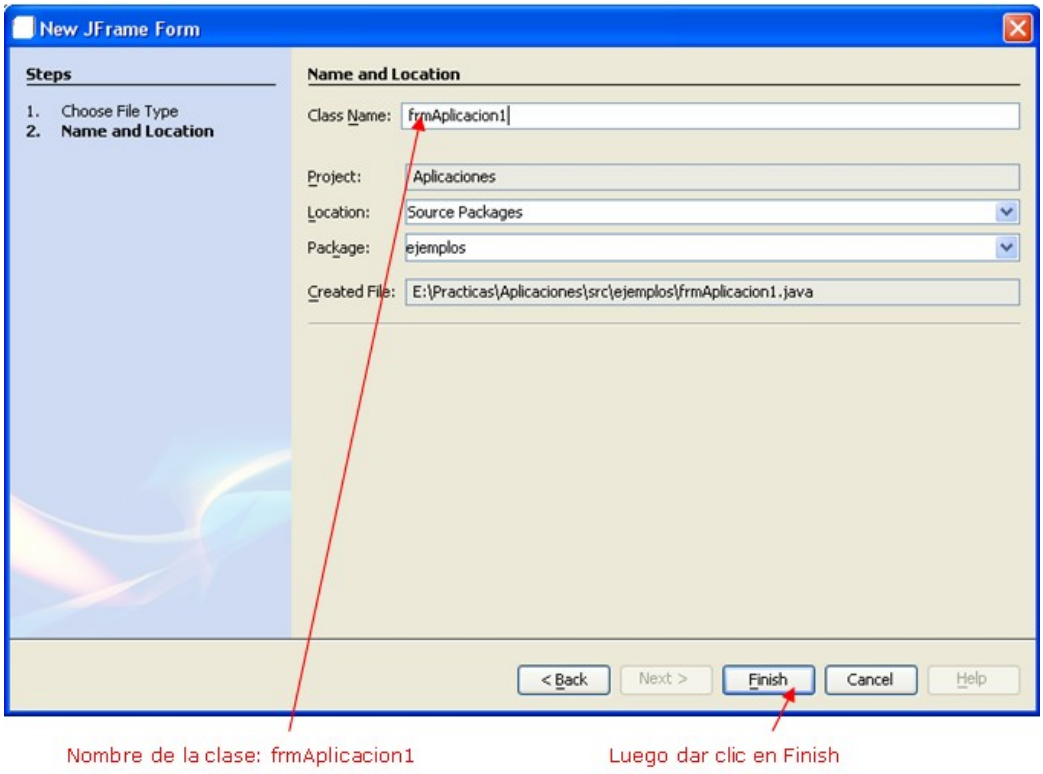
- Procedamos a crear un proyecto denominado **Aplicaciones** dentro de una carpeta llamada **Practicas** y como clase ***Aplicación1*** que pertenece al paquete de ***ejemplos***. Luego damos click en el botón de comando ***Finish***.



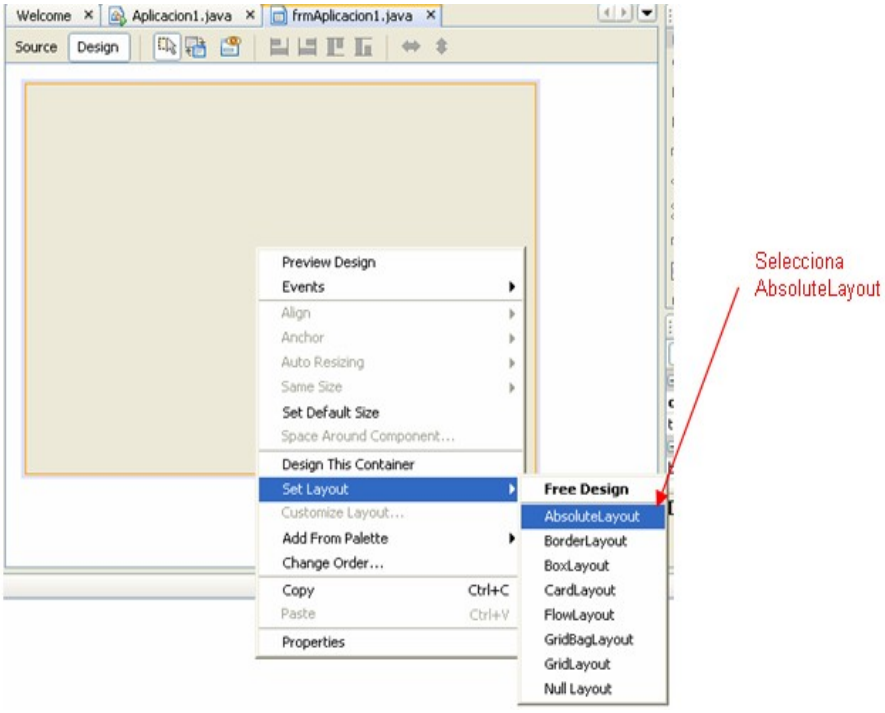

Seleccionamos la carpeta de ejemplos, damos click con el botón derecho del mouse y elegimos la opción ***New*** y posteriormente ***JframeForm***.



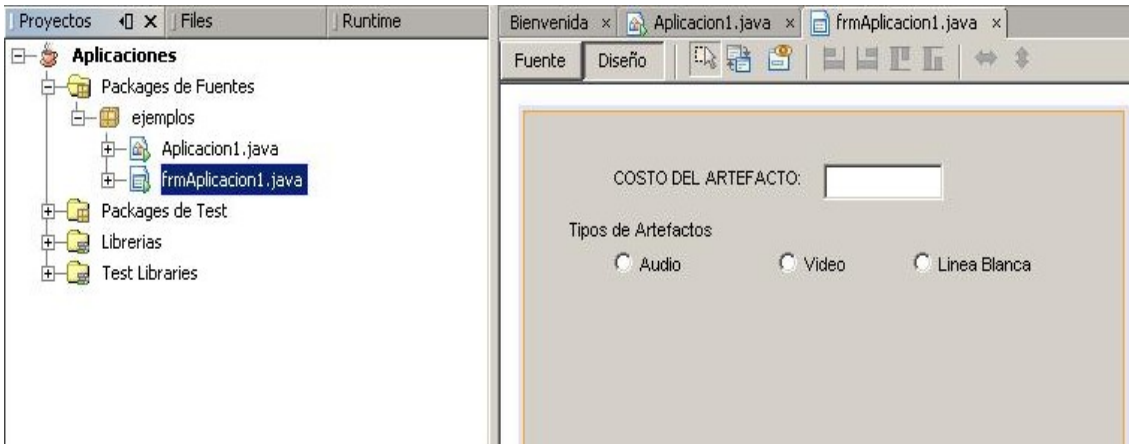

- A continuación, se visualiza la siguiente ventana y cambiamos el nombre de la clase indicado por la flecha:



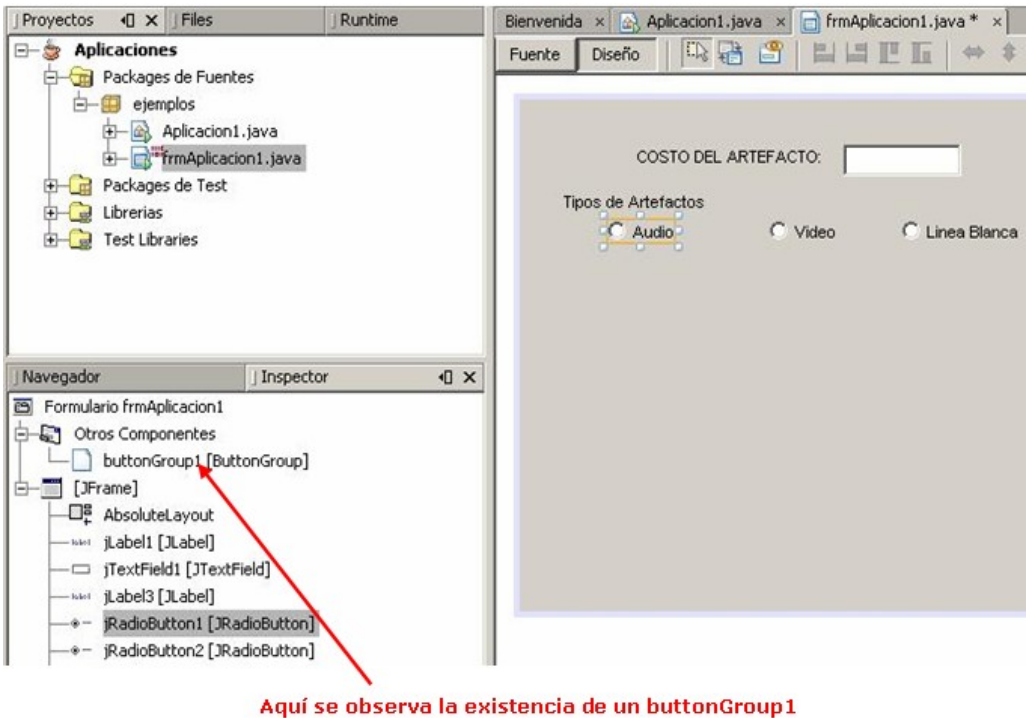

- Ahora procedemos a diseñar el formulario, donde se hará énfasis en el manejo de los nuevos objetos de control. No olvidemos que cada vez que usemos un formulario su ***Layout*** debe ser cambiado a ***Absolute Layout*** como se aprecia en la siguiente figura:



- Dibujamos como etiqueta (usando un JLabel) la expresión: "Costo del Artefacto" y el cuadro de texto (Usando un JtextField). También a través de una etiqueta colocamos la expresión "Tipo de Artefacto" y luego 3 objetos JRadioButton. Los objetos JRadioButton deben expresar Audio, Video y Línea Blanca. Debemos recordar que se tendrá que hacer uso de la propiedad Text para cambiar las expresiones.



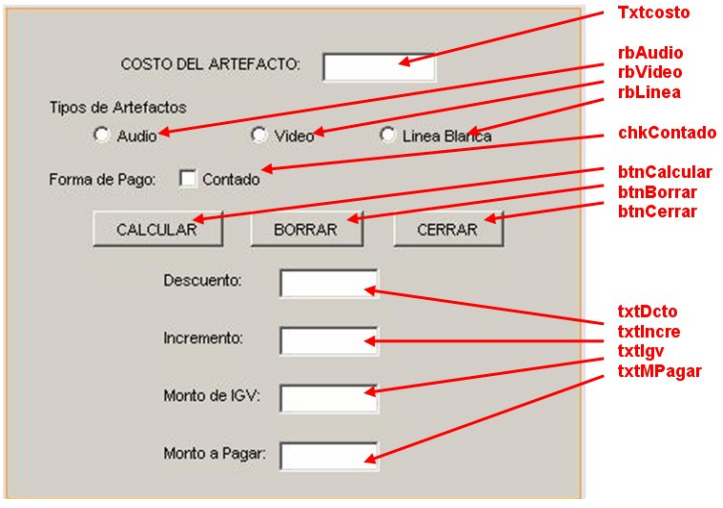

- De la paleta SwingControls seleccionamos ButtonGroup y lo arrastramos hacia el formulario. Esto ocasionará que se cree un objeto ButtonGroup1 tal como se aprecia en el navegador (lado izquierdo del diseño del formulario).



- Seleccionamos el objeto JRadioButton referido a Audio y luego buscamos en la ventana de propiedades, la propiedad ***buttongroup*** y luego eligimos ***buttongroup1***.


Lo mismo hacemos para los tipos de artefactos Video y Línea Blanca. Por lo tanto, los tres objetos JRadioButton deben pertenecer a ***buttongroup1***. Si pretendemos en estos momentos ejecutar el formulario podremos seleccionar uno de los tres tipos de artefactos.

- A continuación, agregamos un objeto JCheckBox para indicar la forma de pago que solo puede ser de dos posibilidades: Contado o Crédito. Seleccionado significa al Contado, no seleccionado significa al crédito. Agregamos los demás objetos de control que se visualiza en el diseño del formulario y que fueron estudiados en la sesión anterior. En el diseño del formulario se indica los nombres de los objetos y debemos recordar que para asignar un nombre a un objeto de control hay que seleccionar al objeto y dando click con el botón derecho del mouse se procede a seleccionar la opción ***Change Variable Name***



- Procedamos a programar en los botones de comando:

  En el botón de comando CALCULAR (btnCalcular), al darle doble click, escribimos el siguiente código: (lo escrito en azul)

```
private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt)
{
        double costo, dcto=0, incre=0, igv, mpagar;
        costo=Double.parseDouble(txtCosto.getText());
        if (chkContado.isSelected())
        {
                if (rbAudio.isSelected())
                dcto=costo*0.06;
        if (rbVideo.isSelected())
        dcto=costo*0.08;
        if (rbLinea.isSelected())
        dcto=costo*0.05;
}
else
{
if (rbAudio.isSelected())
                        incre=costo*0.07;
                if (rbVideo.isSelected())
                incre=costo*0.09;
                if (rbLinea.isSelected())
incre=costo*0.1;
}
igv=(costo-dcto+incre)*0.19;
mpagar=(costo-dcto+incre)+igv;
txtDcto.setText(String.valueOf(dcto));
txtIncre.setText(String.valueOf(incre));
txtIgv.setText(String.valueOf(igv));
txtMPagar.setText(String.valueOf(mpagar));
}
```

  Una vez declaradas las variables de memoria, en la variable **costo** se asigna el valor introducido en el cuadro de texto ***txtCosto*** gracias al método ***getText()*** que logra obtener el dato colocado en el objeto de control. Con la sentencia if se evalúa si está seleccionada la opción al contado y, si es así, se procede a evaluar cuál de los tipos de artefactos está seleccionado para aplicar el cálculo del descuento que será asignando a la variable de memoria **dcto**. En caso que no esté seleccionada la opción al Contado entonces se asume que la forma de pago es al crédito y se procede a evaluar cuál de los tipos de artefactos está

seleccionado para aplicar el cálculo del incremento que será asignado a la variable **incre**. A continuación, se calcula el IGV y el monto a pagar. Luego, los objetos de control ***txtDcto***, ***txtIncre***, ***txtIgv*** y ***txtMPagar*** reciben valores a través de las variables de memoria **dcto**, **incre**, **igv** y **mpagar** en sus cuadros de textos gracias al método ***setText()***, por supuesto previamente se tiene que convertir a cadena de texto los valores numéricos de las variables usando el método ***valueOf()*** de la clase ***String***.

En el botón de comando BORRAR (btnBorrar), luego de darle doble click, escribimos el siguiente código: (lo escrito en azul)

```
private void btnBorrarActionPerformed(java.awt.event.ActionEvent evt)
{
        txtCosto.setText("");
        txtDcto.setText("");
        txtIncre.setText("");
        txtIgv.setText("");
        txtMPagar.setText("");
        rbAudio.setSelected(false);
        rbVideo.setSelected(false);
        rbLinea.setSelected(false);
        chkContado.setSelected(false);
        txtCosto.requestFocus();
}
```

Se limpian los cuadros de textos con sólo poner "" en el método ***setText()*** y a los objetos botón de radio (JRadioButton) y el objeto de caja verificación (JCheckBox) se les aplica el método setSelected() para lograr quitar la selección de estos objetos. Lo más importante es que los cuadros de textos estén limpios para poder permitir el ingreso de nuevos datos.

En el botón de comando CERRAR (btnCerrar), luego de darle doble click, escribimos el siguiente código: (lo escrito en azul)

```
private void btnCerrarActionPerformed(java.awt.event.ActionEvent evt)
{
dispose();
}
```

El método ***dispose()*** permite descargar el formulario y terminar la ejecución de la aplicación.

- Luego procedemos a ejecutar la aplicación seleccionando frmAplicacion1 en la página o pestaña Proyects (se encuentra al lado izquierdo del diseño del formulario) y al dar click botón derecho elegimos ***Run File***.

# USO DEL OBJETO JLIST

## Objeto de Control JList

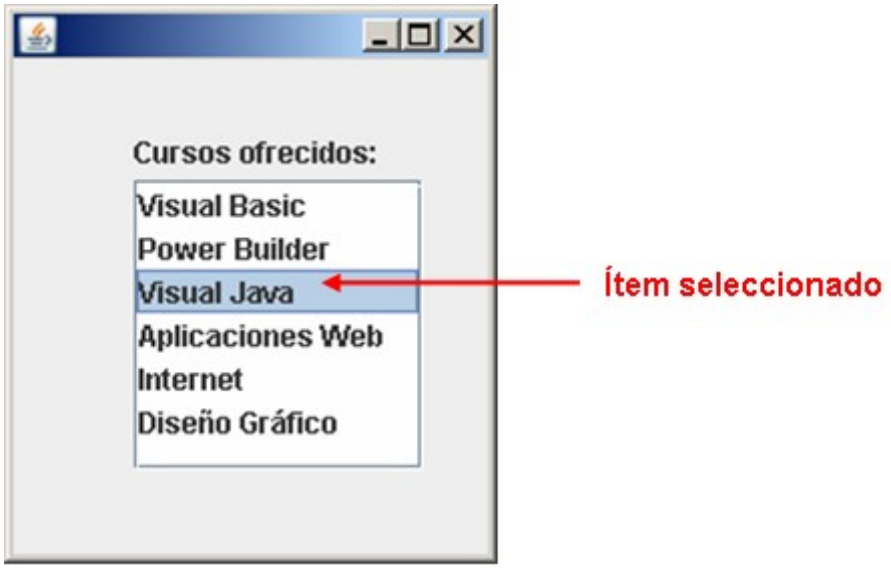
Un objeto de control Jlist permite dibujar en el formulario una caja de lista de opciones (ítems). Cuando el formulario se encuentra en la etapa de ejecución se pueden seleccionar sus ítems. Pero para trabajar con este objeto es necesario usar un objeto de la categoría de Swing Containers denominado JScrollPane. El objeto JScrollPane permite hacer que el objeto JList tenga barra de desplazamiento que es necesaria cuando el número de ítems es grande y no puede ser visto a simple vista en el objeto de control JList. Cabe señalar que los objetos que pertenecen a Swing Containers serán estudiados con mayor detalle en la segunda unidad de aprendizaje del curso, pero el uso de Jlist nos obliga utilizar el objeto contenedor JScrollPane.

### Propiedades más usadas:

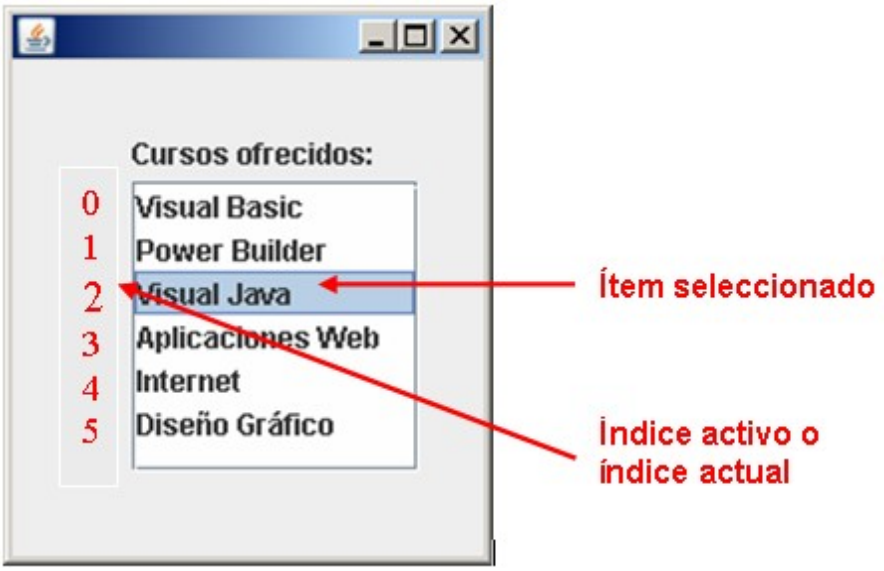
- Model: Permite establecer los ítems de la caja de lista.
- Font: Permite establecer el tipo de letra en el objeto de control.
- Enabled: Para habilitar o inhabilitar el uso del objeto de control.

### Métodos más usados:

- setModel(): Permite vincular una variable objeto de tipo model a un objeto de control JList.
- getSelectedValue(): Contiene el ítem seleccionado de la caja de lista.



- getSelectedIndex(): Contiene el valor del índice activo o índice actual del ítem seleccionado de la caja de lista. El índice es un valor numérico correlativo no visible que va desde 0.




**Evento más usado:**

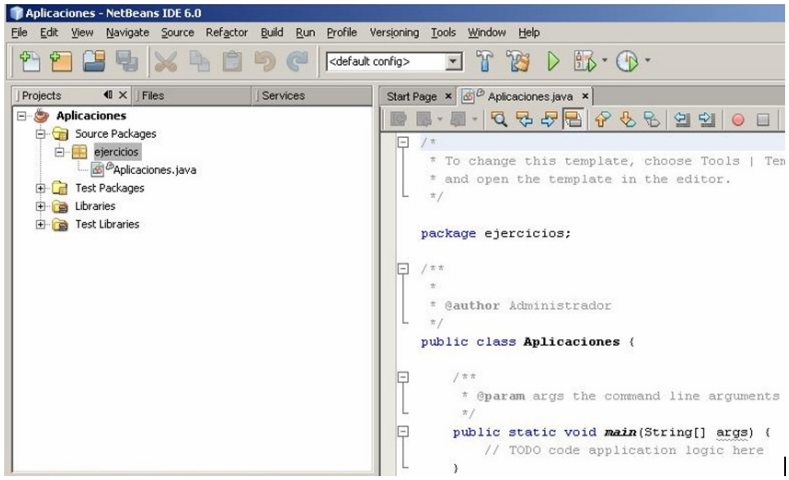
- ValueChanged(): Sucede cuando el usuario selecciona un ítem de la caja de lista.

## Aplicación

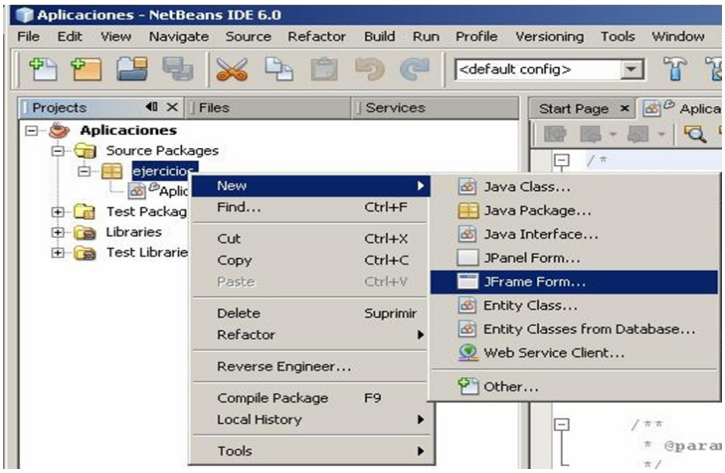
Construir una aplicación que permita el ingreso del nombre del alumno y poder seleccionar uno o varios cursos que éste quisiera llevar. El pago por los cursos seleccionados podrá ser pagado al contado o al crédito. Si el pago es al contado hay un descuento del 5% del costo total de los cursos a llevar y si el pago es al crédito se pagará un incremento del 7% del costo total. La aplicación debe mostrar el descuento, el incremento y el monto a pagar por los seleccionados.

**Solución:**

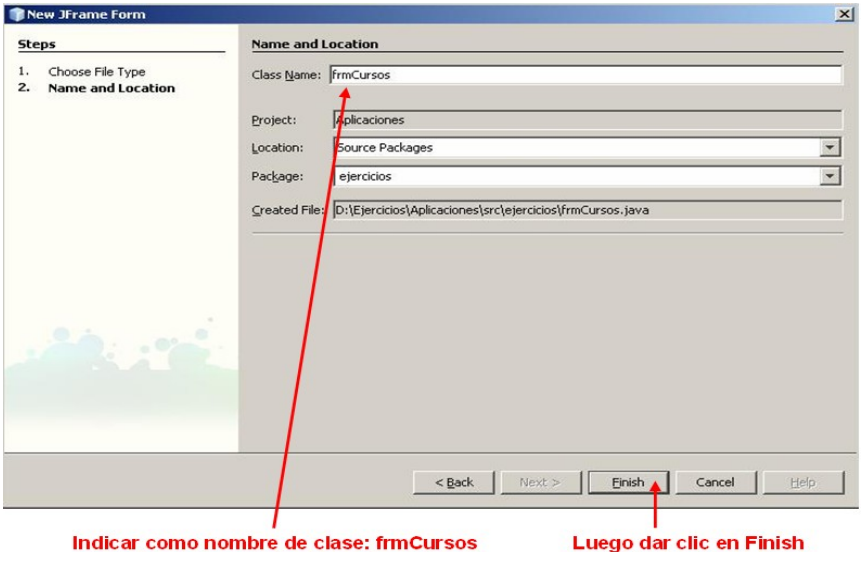
- Iniciamos con la creación de un proyecto denominado Aplicaciones. Seleccionamos del menú, la opción File y luego New Proyect. Aparece la ventana de New Proyect y damos clic en el botón de comando Next. En la ventana ***New Java Application*** indicamos como nombre de proyecto **Aplicaciones** creando como paquete **aplicaciones.** Al dar click en el botón de comando ***Finish*** nos encontramos con el entorno de desarrollo.



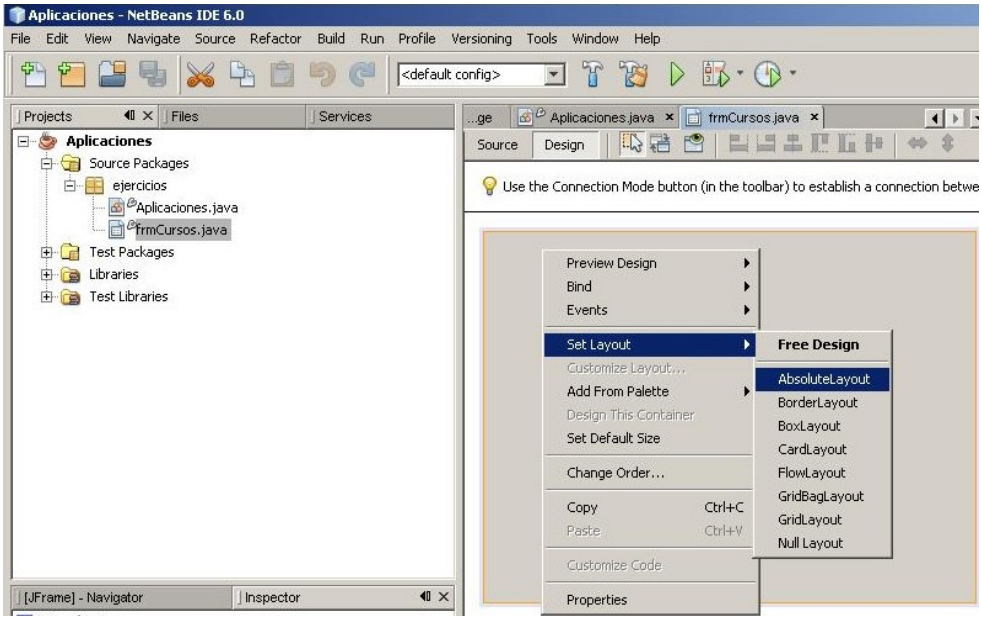

- Seleccionamos el paquete de aplicaciones y al dar click con el botón derecho del mouse elegimos la opción New y luego JFrameForm.



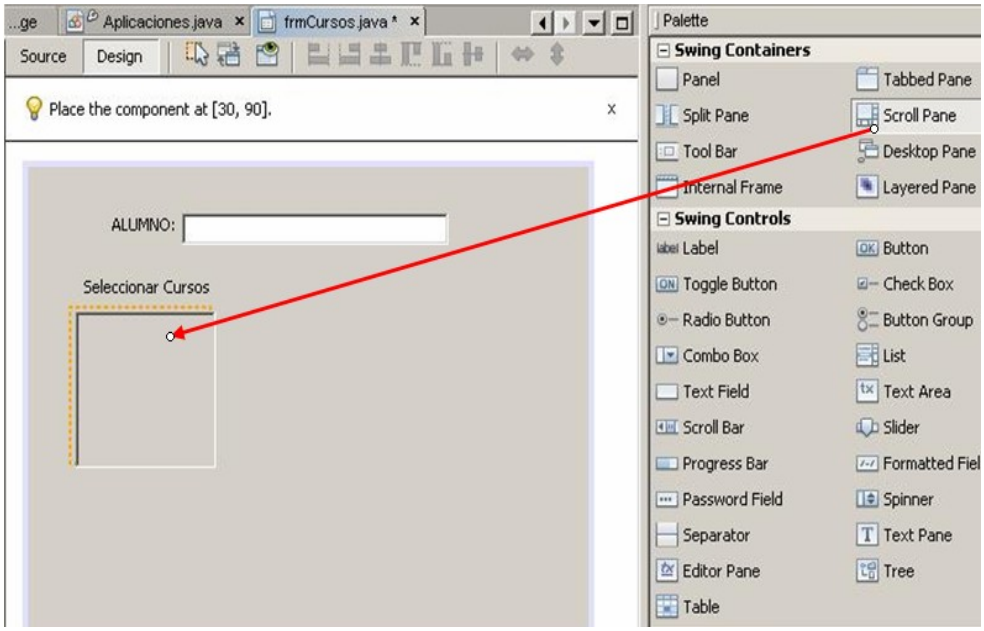

- A continuación se muestra la ventana New Jframe Form que debe quedar así:



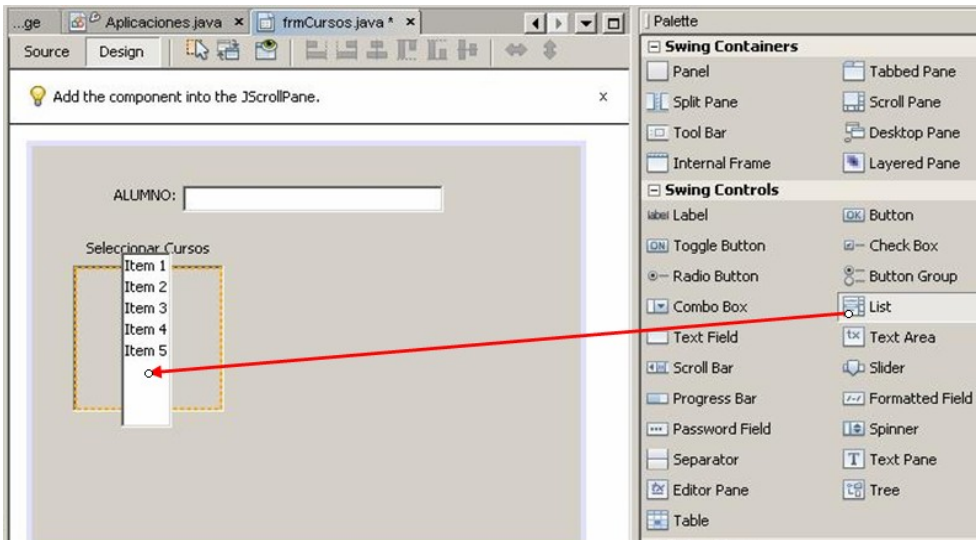
- No olvidemos de dar click botón derecho del mouse sobre el formulario y establecer ***AbsoluteLayout*** en ***Set Layout***.



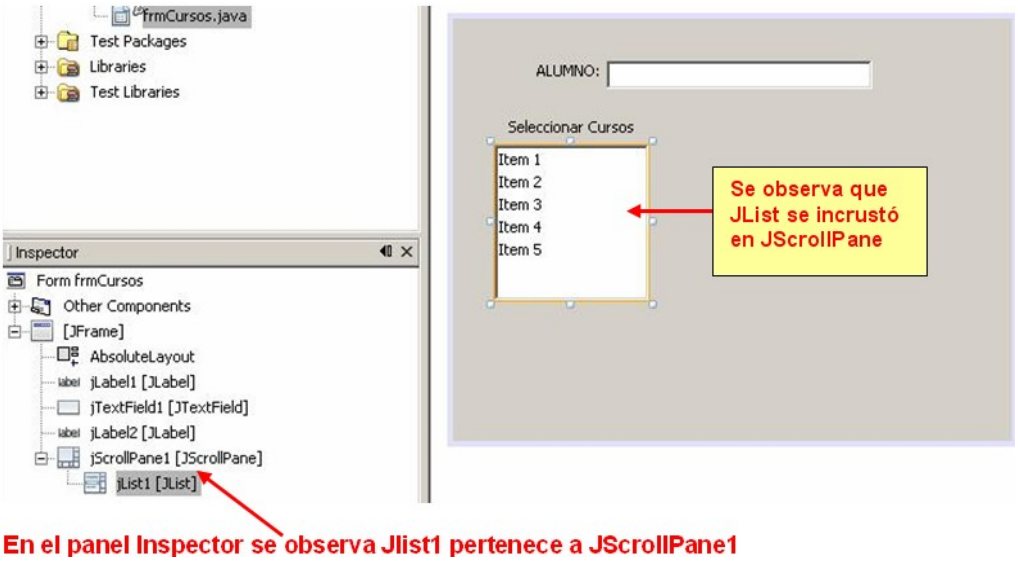

- Procedemos a colocar un objeto Jlabel con la expresión "ALUMNO:" acompañado de un cuadro de texto (JTextField). Luego, colocar un Jlabel que exprese "Seleccionar Cursos" y debajo de esta expresión dibujar un objeto JScrollPane.



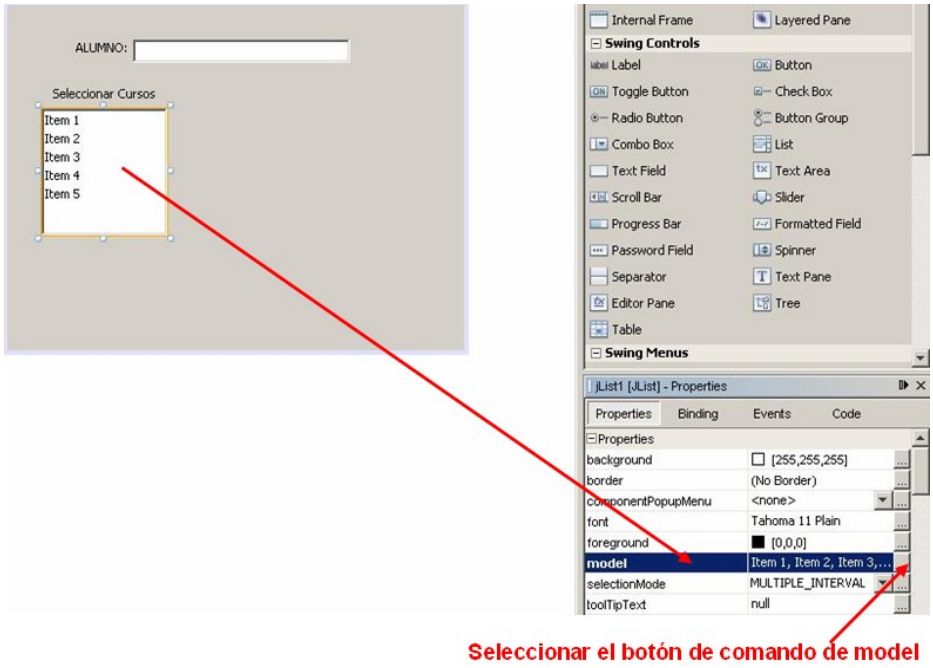
- En el objeto JScrollPane colocamos un objeto JList y al momento de llevarlo al diseño del formulario se muestra de la siguiente manera:



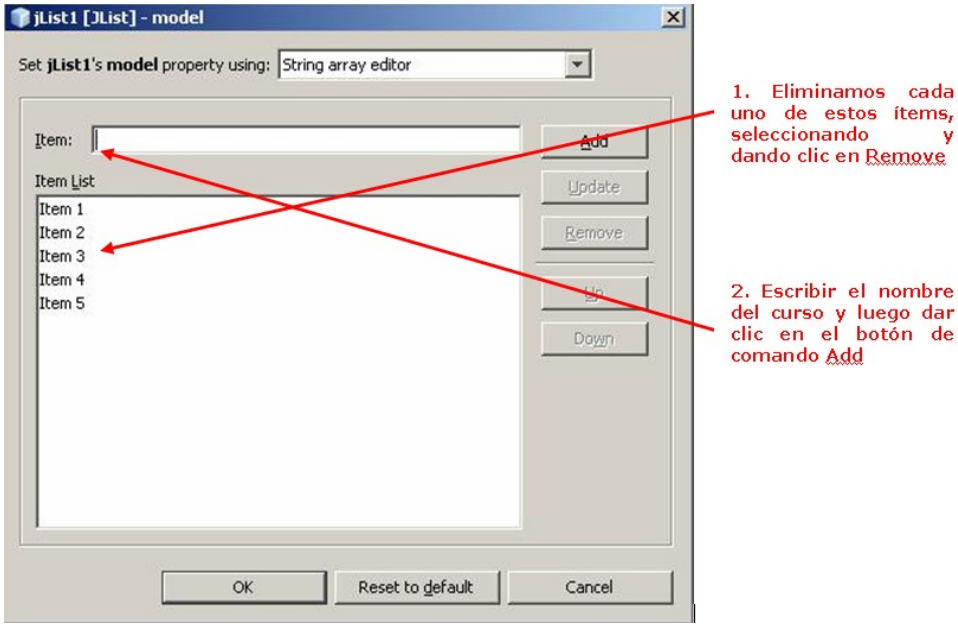

- Luego queda el objeto JList dentro del objeto JScrollPane.




- Seleccionamos el objeto de control Jlist1 y en la ventana de propiedades se tiene una propiedad llamada **model** que permite colocar los ítems dentro de la caja de lista Jlist1.



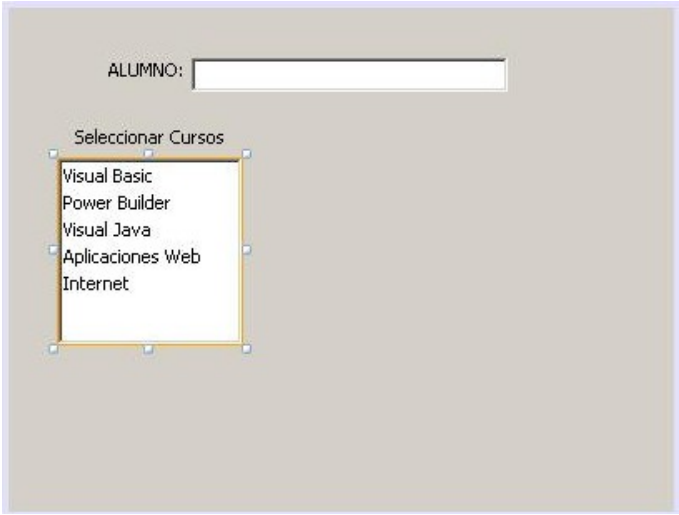

- Luego de seleccionar el botón de comando referido a la propiedad **model** se muestra la siguiente ventana:



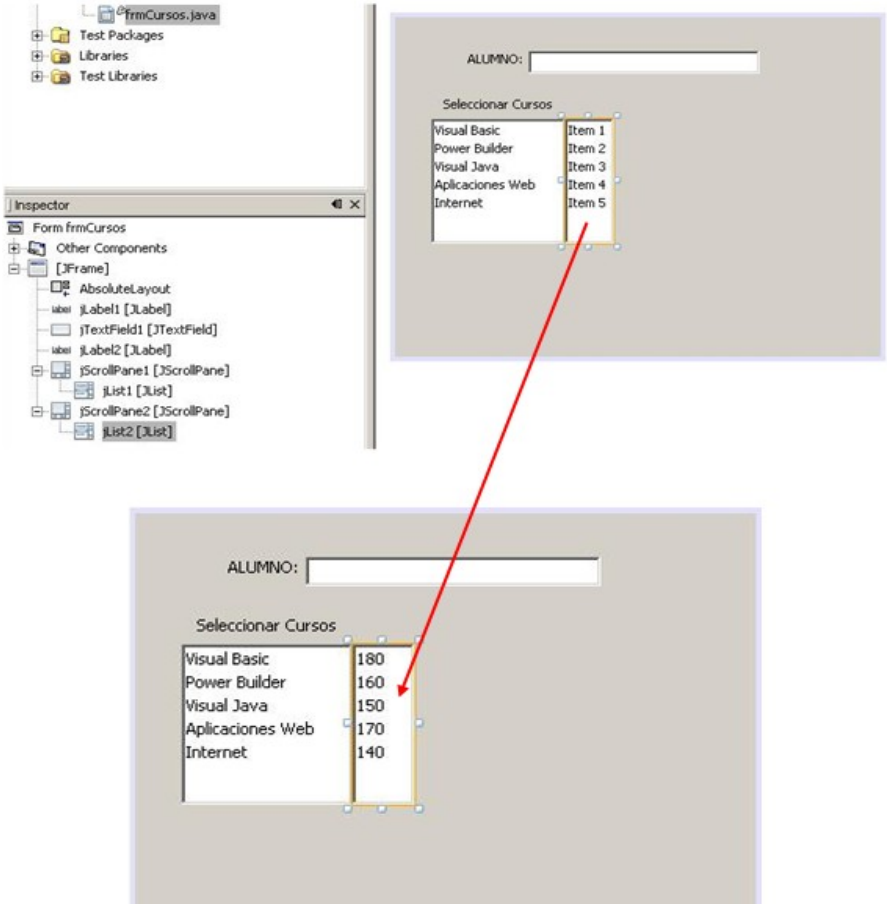

- Ingresemos los nombres de los cursos tal como se muestra en la siguiente ventana:


Luego dar clic en el botón de comando ***OK***.

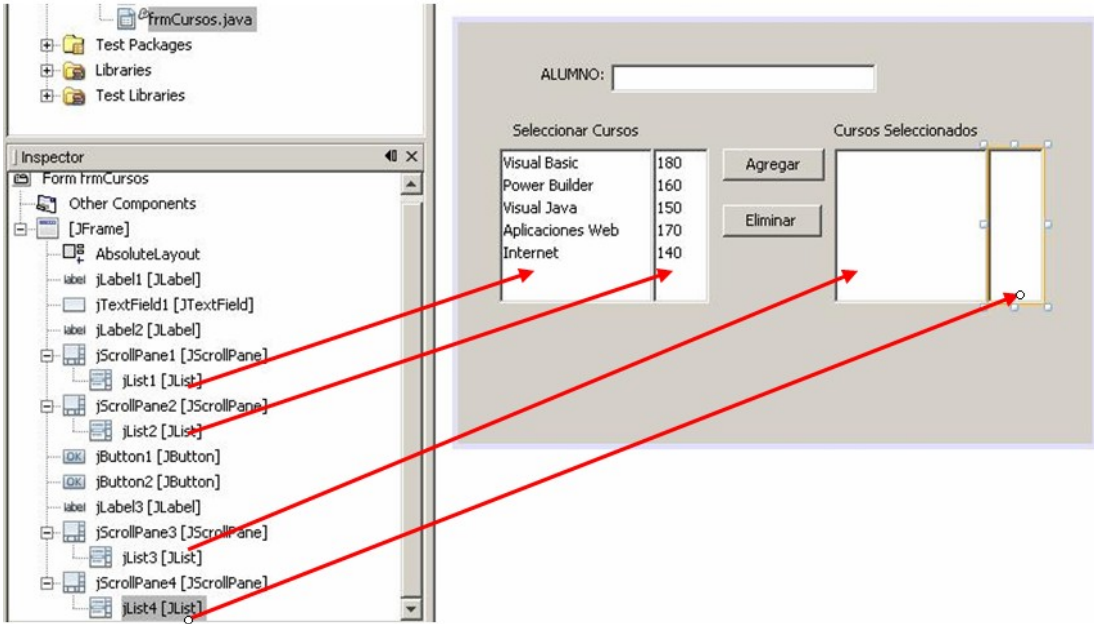
- El diseño del formulario debe quedar así:



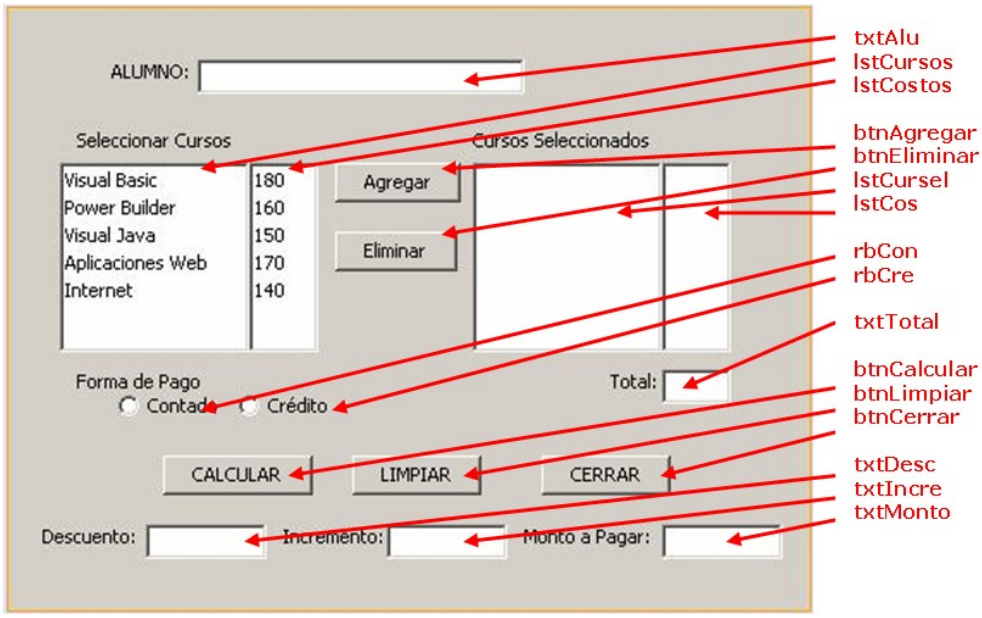

- Luego vamos a colocar en el diseño del formulario otro objeto JScrollPane junto a JList1 para luego poner un objeto JList. En esta lista colocaremos los costos de cada curso.




Adicionalmente pondremos en el diseño del formulario dos botones de comando referido a Agregar y Eliminar y junto a ellos dos objetos Jlist, por supuesto previamente debemos usar dos objetos JScrollPane.



- Posteriormente agregamos las formas de pago con dos objetos JRadioButton, los botones de comando Calcular, Limpiar y Cerrar. Finalmente, los objetos que mostrarán el descuento, el incremento y el monto a pagar por los cursos seleccionados. El diseño del formulario debe quedar así:




- Ahora bien, si observamos en el diseño del formulario de las cuatros objetos Jlist, dos de ellos ya tienen ítems como lo son lstCursos y lstCostos. En cambio, los objetos lstCursel y lstCos se llenarán en función a lo seleccionado y agregado con el botón de comando Agregar. Por lo tanto, debemos definir un modelo (**model**) para aquellas listas que se llenarán en tiempo de ejecución. Por esto debemos definir las variables modelo1 y modelo2 como DefaultListModel(), como se muestra a continuación (escribe lo que indica las flechas de color rojo):

```
public class frmCursos extends javax.swing.JFrame {
    // definir un modelo que es necesario para manejo de listas
    private DefaultListModel modelo1=new DefaultListModel();
    private DefaultListModel modelo2=new DefaultListModel();
```

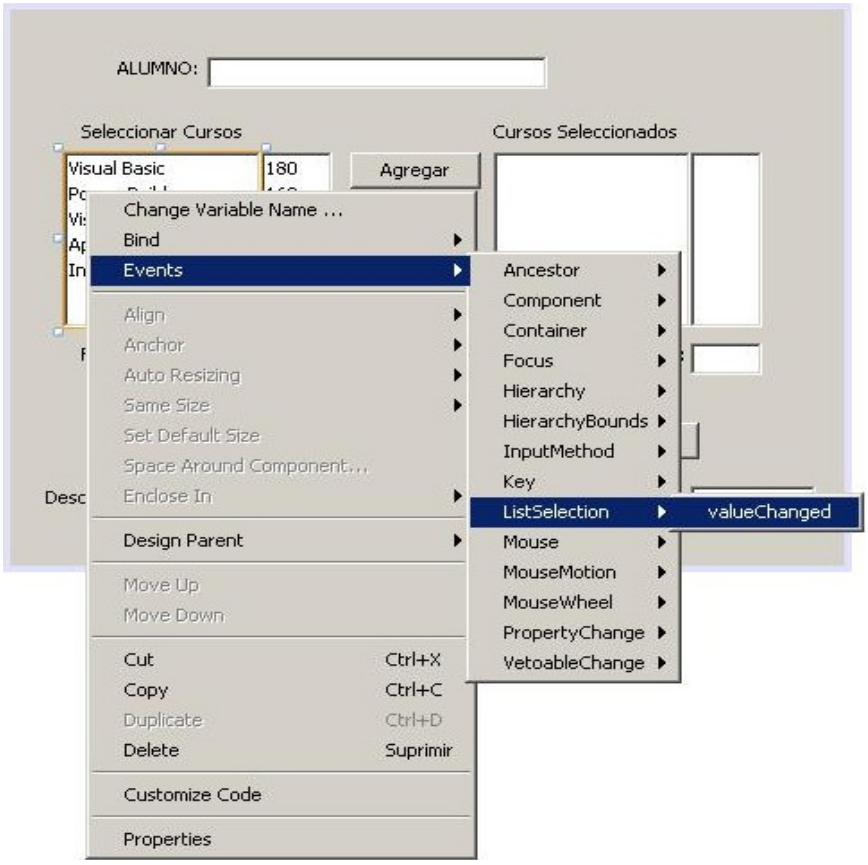
Para que funcione correctamente la clase ***DefaultListModel*** es necesario agregar el paquete import javax.swing.*; después del paquete Aplicaciones.

- Una vez definidas las variables modelo1 y modelo2, en el **método constructor** se debe indicar que la variable **modelo1** es para la caja de lista **lstCursel** y la variable **modelo2** es para la caja de lista **lstCos**, todo esto se podrá hacer con el método **setModel()**. También hacemos que los botones de comando Agregar y Eliminar se inhabiliten desde la ejecución de la aplicación.

```
public frmCursos() {
    initComponents();
    lstCursel.setModel(modelo1);
    lstCos.setModel(modelo2);
    btnAgregar.setEnabled(false);
    btnEliminar.setEnabled(false);
}
```

Agrega éstas cuatro líneas de código

- Bien, ahora debemos programar sobre el objeto lsrCursos, para que el usuario al momento de seleccionar un curso se marque simultáneamente el costo y se habilite el botón de comando Agregar. Para esto se debe seleccionar un evento de la caja de lista lstCursos llamado ValueChanged perteneciente a ListSelection.



En el evento mencionado programa lo siguiente:

```
    private void lstCursosValueChanged(javax.swing.event.ListSelectionEvent evt) {
// TODO add your handling code here:
        int indice;
        indice=lstCursos.getSelectedIndex();
        lstCostos.setSelectedIndex(indice);
        btnAgregar.setEnabled(true);
    }
```

Agrega éstas cuatro líneas de código

Se declara una variable entera llamada índice, esta variable recibe el valor del índice del ítem seleccionado gracias al método **getSelectedIndex()**. Por ejemplo, si de la caja de lista **lstCursos** estuviera seleccionado *Power Builder*, éste método devolvería el valor de 1. El valor 0 lo tiene Visual Basic, el valor de 2 lo tiene Visual Java y así sucesivamente. Lo que se quiere es seleccionar el ítem de la caja de lista de lstCostos que tenga el mismo índice que **lstCursos**, para ello se usa el método ***setSelectedIndex(indice)*** para dar el mismo índice a la caja de lista **lstCostos**. Finalmente, hacemos que el botón de comando Agregar se habilite con el método ***setEnabled()***.

- A continuación, escribimos el siguiente código en el botón de comando **Agregar** (sólo se escribe lo que señala la llave de color rojo):

```
    private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        String curso,costo;
        int cuenta, i, total=0;
        curso=(String)lstCursos.getSelectedValue();
        costo=(String)lstCostos.getSelectedValue();
        modelo1.addElement(curso);
        modelo2.addElement(costo);
        cuenta=modelo2.size();
        for(i=0;i<cuenta;i++)
            total=total+Integer.parseInt((String)modelo2.elementAt(i));
        txtTotal.setText(String.valueOf(total));
        btnAgregar.setEnabled(false);
    }
```

Aquí declaramos dos variables de tipo *String* llamados **curso** y **costo** y las variables enteras **cuenta**, **i** y **total**. En la variable **curso** se almacena el curso seleccionado en la lista **lstCursos**, el método ***getSelectionValue()*** trae consigo el ítem seleccionado pero lo trae como objeto y al poner ***(String)*** hacemos que se convierta en cadena para que pueda ser asignada a la variable **curso**. De igual manera se hace con la variable **costo**. Para agregar un ítem a una caja de lista se usa el método ***addElement*** perteneciente al objeto variable **modelo1** o **modelo2**. Con la variable **cuenta** se almacena el total de ítems que hay en la caja de lista **lstCos** pero a través de la variable objeto **modelo2**. Con la sentencia repetitiva for se busca extraer cada uno de los ítems de la caja de lista **lstCos** e ir sumando en cada interacción para poder encontrar el costo total de los cursos seleccionados, para esto usamos el método ***elementAt()*** que devuelve un ítem de una caja de lista con sólo indicar el valor del índice. Finalmente en el objeto **txtTotal** se visualiza el contenido de la variable **total** e inhabilitamos el botón de comando **Agregar**.

- Ahora programamos en la caja de lista lstCursel el evento ***ValueChanged*** cuando querramos seleccionar un curso para luego eliminarlo (sólo escribe lo que señala la llave de color rojo).

```
    private void lstCurselValueChanged(javax.swing.event.ListSelectionEvent evt) {
// TODO add your handling code here:
        int indice;
        indice=lstCursel.getSelectedIndex();
        lstCos.setSelectedIndex(indice);
        btnEliminar.setEnabled(true);
    }
```

La idea es la misma que se aplicó en la programación sobre el evento ***ValueChanged*** del objeto **lstCursos**. La diferencia está en que esta vez se habilita el botón de comando **Eliminar**.

- En el botón de comando Eliminar colocamos la siguiente programación (sólo se escribe lo que señala la llave de color rojo):

```
    private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        int indice, cuenta, i, total=0;
        indice=lstCursel.getSelectedIndex();
        modelo1.remove(indice);
        modelo2.remove(indice);
        cuenta=modelo2.size();
        for(i=0;i<cuenta;i++)
            total=total+Integer.parseInt((String)modelo2.elementAt(i));
        txtTotal.setText(String.valueOf(total));
        btnEliminar.setEnabled(false);
    }
```

Lo novedoso de esta programación es la presencia del método ***remove()*** que elimina un ítem de la lista a través de la variable objeto **modelo1** ó **modelo2** dado el valor del índice. Al final de la programación se vuelve a calcular el costo total de los cursos seleccionados y se inhabilita el botón de comando **Eliminar**.

- Ahora procedemos a programar en el botón de comando **Calcular** (sólo se escribe lo que señala la llave de color rojo):

```
    private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        double desc=0, incre=0, monto;
        int total;
        total=Integer.parseInt(txtTotal.getText());
        if (rbCon.isSelected())
            desc=total*0.05;
        if (rbCre.isSelected())
            incre=total*0.07;
        monto=total-desc+incre;
        txtDesc.setText(String.valueOf(desc));
        txtIncre.setText(String.valueOf(incre));
        txtMonto.setText(String.valueOf(monto));
    }
```

- A continuación procedemos a programar en el botón de comando Limpiar (sólo se escribe lo que señala la llave de color rojo):

```
    private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        txtAlu.setText("");
        modelo1.clear();
        modelo2.clear();
        txtTotal.setText("");
        txtDesc.setText("");
        txtIncre.setText("");
        txtMonto.setText("");
        lstCursos.setSelectedIndex(5);
        lstCostos.setSelectedIndex(5);
        txtAlu.requestFocus();
    }
```

En esta programación, la novedad está en que para limpiar totalmente una caja de lista se hace con el método ***clear()*** perteneciente a las variables objeto **modelo1** y **modelo2**, con lo cual también se hace la limpieza a los objetos **lstCursel** y **lstCos**. También, hacemos el uso del método ***setSelectedIndex()*** dando el valor de 5, ya que dicho índice no existe en la caja de lista, lo que hace que se pierda lo seleccionado.

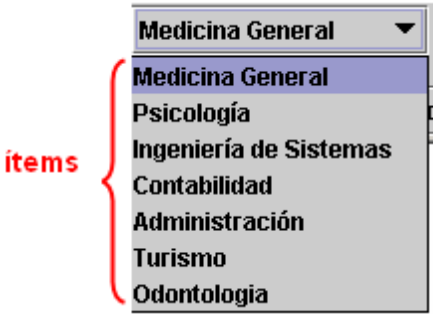
- Finalmente programamos en el botón de comando Cerrar:

# USO DEL OBJETO JCOMBOBOX

## Objeto de Control JComboBox

Un objeto de control JComboBox permite dibujar en el formulario una lista desplegadle, la cual contiene opciones (ítems). ComboBox significa "cuadro combinado" porque combina un cuadro de texto con una caja de lista, es como si fuera un JtextField mezclado o combinado con un Jlist. Tiene la particularidad de que se debe seleccionar un botón de comando de despliegue y luego seleccionar la opción o ítem.


Una vez dado clic en el botón de despliegue se muestra las opciones o ítems del objeto




**Propiedades más usadas:**

- Model: Permite establecer los ítems de la caja de lista.
- Font: Permite establecer el tipo de letra en el objeto de control.
- Enabled: Para habilitar o inhabilitar el uso del objeto de control.
- getSelectedIndex: Contiene el índice del ítem seleccionado
- setSelectedItem: Contiene el ítem seleccionado

**Métodos más usados:**

- setModel(): Permite vincular una variable objeto de tipo model a un objeto de control JList.
- getItemAt(): Devuelve el ítem que está en el índice que se especifica.

- getSelectedIndex(): Contiene el valor del índice activo o índice actual del ítem seleccionado de la caja de lista. El índice es un valor numérico correlativo no visible que va desde 0.

**Evento más usado:**

- ValueChanged(): Sucede cuando el usuario selecciona un ítem de la caja de lista.

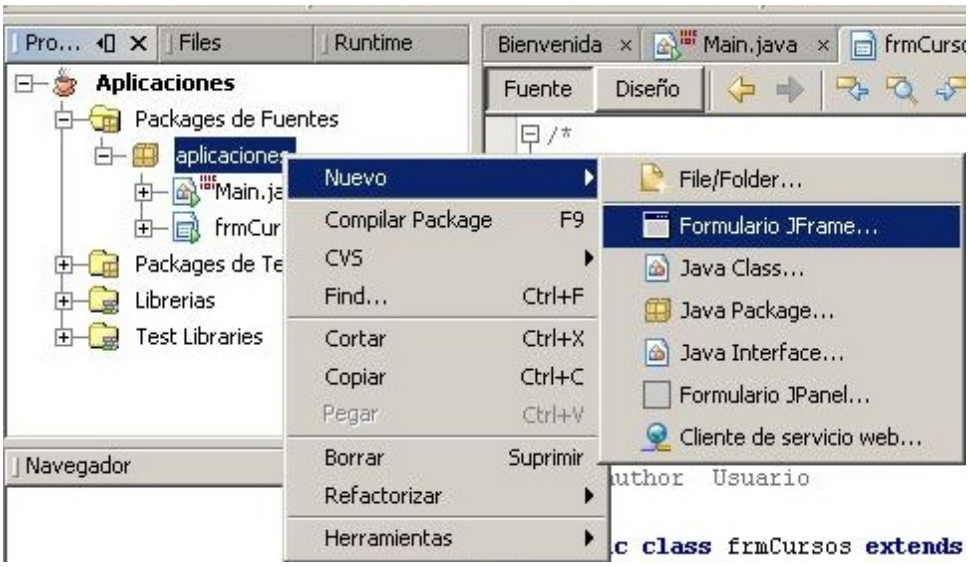
## Aplicación

Construir una aplicación que permita el ingreso del nombre del alumno y poder seleccionar de una lista desplegadle una categoría de los cursos. Al momento de seleccionar la categoría se debe mostrar los cursos con sus respectivos costos en las cajas de listas (los JList que se muestran al lado izquierdo del diseño del formulario). Una vez visualizado los cursos el usuario puede seleccionar y agregar en las cajas de listas (los Jlist que se muestran al lado derecho del diseño del formulario) los cursos solicitados por el alumno. El pago por el servicio de enseñanza se establece de la siguiente manera:

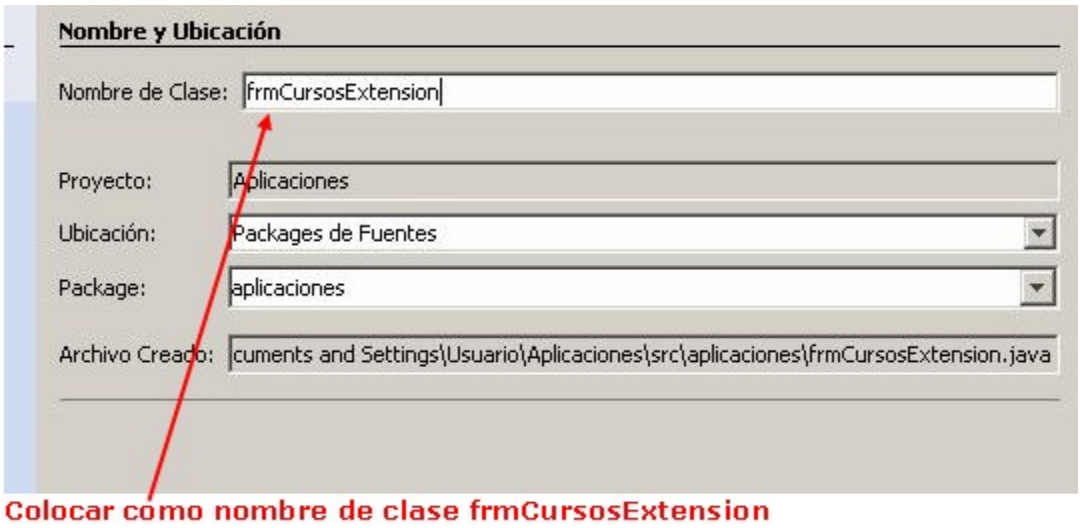
- Existe un pago por matrícula del 80% del costo total (suma de los costos de los cursos escogidos) siempre y cuando quiera el alumno llevar un solo curso, 60% del costo total si lleva dos cursos y 50% del costo total si lleva 3 o más cursos.
- El costo total tiene un descuento del 10% si la forma de pago es al contado y un incremento del 10% si es al crédito.
- Existe un pago mensual cuando la forma de pago es al crédito y es equivalente al costo total incrementado dividido en 4 cuotas.

**Solución:**

- Usarás el mismo proyecto utilizado en la sesión anterior y sólo agregarás un formulario (Jframe).

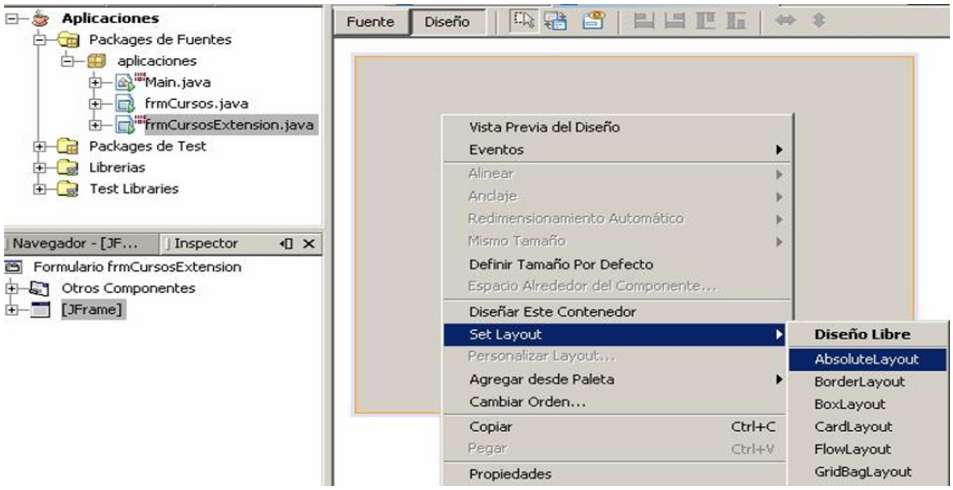



Inmediatamente se muestra la siguiente ventana:

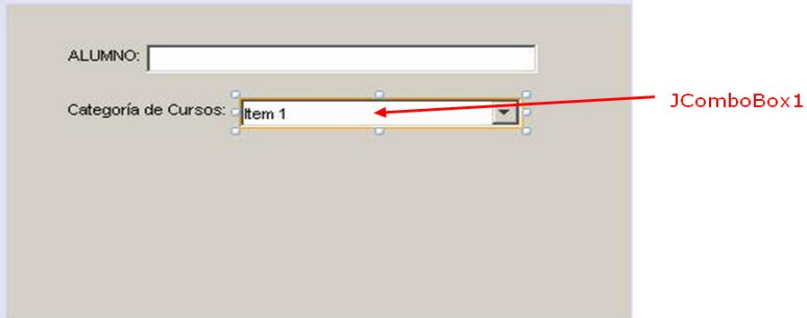



Luego dar clic en el botón de comando ***Finish***.

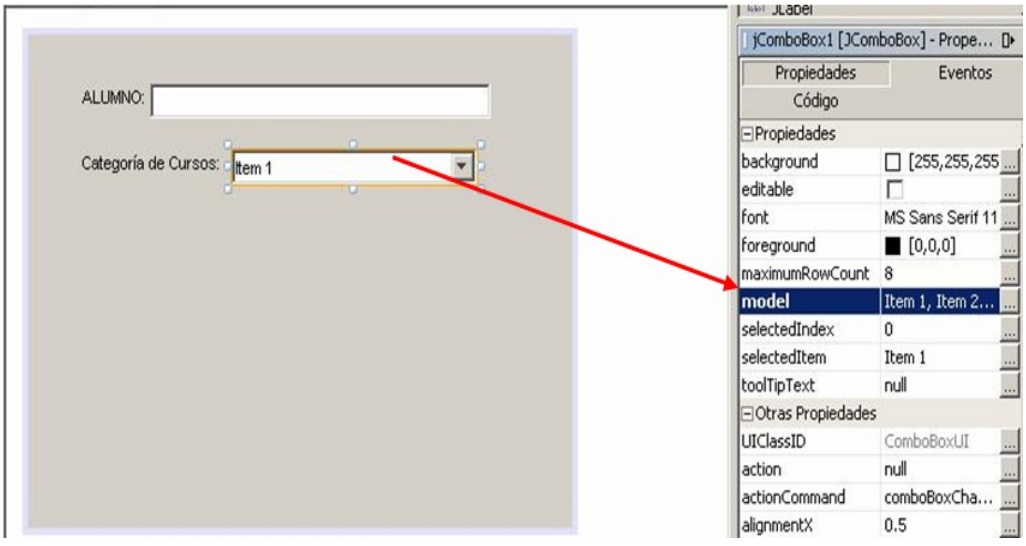
- A continuación se muestra el entorno de desarrollo de NetBeans y no olvides de dar clic en el botón derecho del mouse sobre el formulario y establece ***AbsoluteLayout*** en ***Set Layout***.



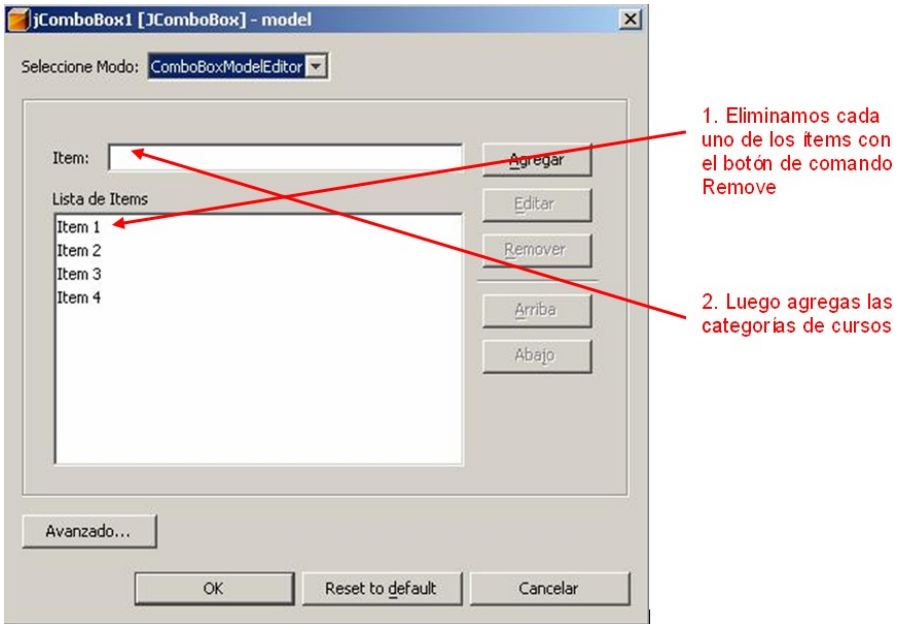

- Procede a colocar un objeto Jlabel con la expresión "ALUMNO:" acompañado de un cuadro de texto (JTextField). Por debajo de "ALUMNO:" colocar un objeto Jlabel que exprese "Categoría de Cursos:" y al lado derecho de éste objeto colocar un objeto JComboBox.




- Selecciona el objeto JComboBox y elige en la ventana de propiedades la propiedad ***model*** que permite colocar los ítems dentro de la caja de lista desplegadle

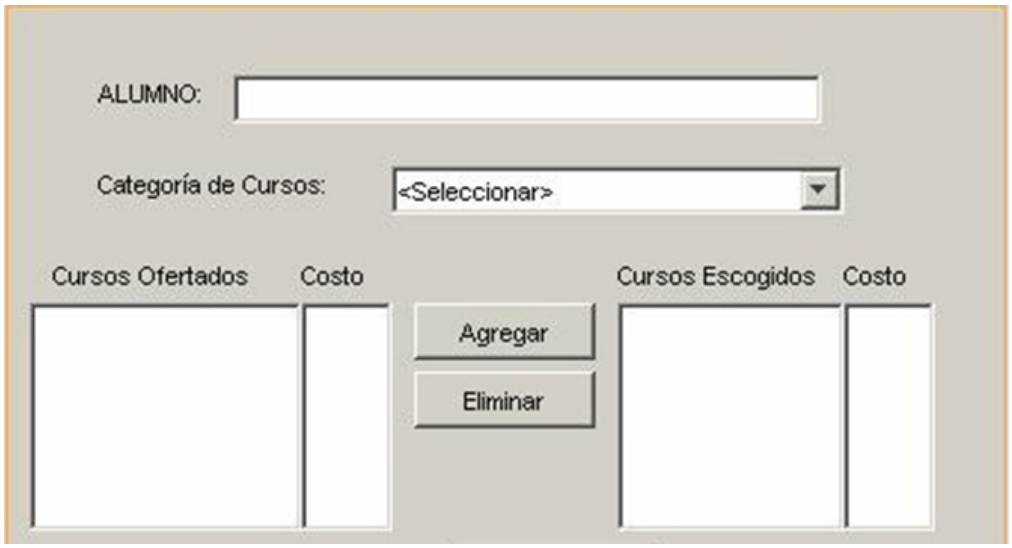


- Luego de seleccionar el botón de comando referido a la propiedad ***model*** se muestra la siguiente ventana:

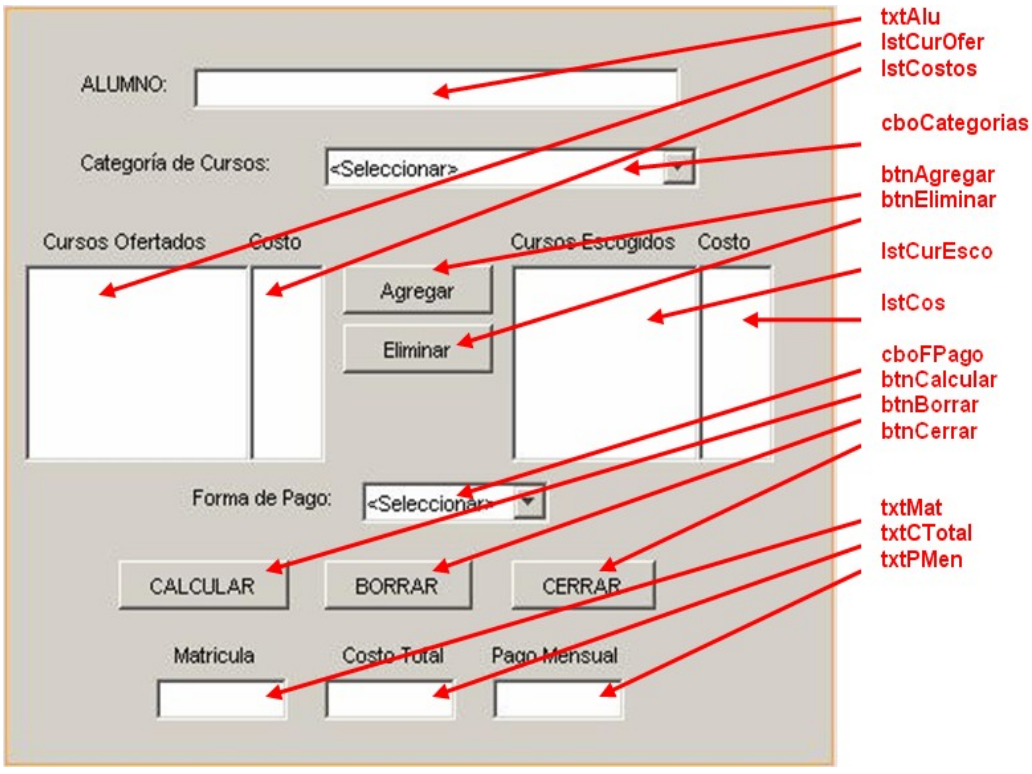



- Ingresas las categorías de cursos como son: <Seleccionar>, Diseño Gráfico, Diseño Web, Ofimática, Lenguajes de Programación y Sistemas Operativos, quedando el diseño del formulario de la siguiente manera:


- Colocarás un objeto Jlabel con la expresión "Cursos Ofertados" y otro objeto Jlabel ubicado al lado derecho de éste último con la expresión "Costo". Añades dos listas debajo de las expresiones de estos dos últimos Jlabel, haciendo que los ítems sean eliminados o removidos a través del uso de la propiedad **model**. Luego agregas dos botones de comando que indiquen **Agregar** y **Eliminar.** Posteriormente, añades dos objetos Jlabel que expresen: "Cursos Escogidos" y "Costo" y debajo de estos objetos Jlabel agregas dos objetos JList siendo también eliminados o removidos los ítem a través del uso de la propiedad **model**.




- A continuación agregas un objeto Jlabel con la expresión "Forma de Pago:" y al lado derecho de éste último objeto colocas un JComboBox que debe contener como ítems: <Seleccionar>, Contado y Crédito. Luego los botones de comando CALCULAR, BORRAR y CERRAR y los objetos que mostrarán el monto de la matrícula, el costo total y el pago mensual. Los nombres de los objetos de control dibujados en el formulario queda de la siguiente manera:



- Ahora bien, si observamos en el diseño del formulario, los cuatros objetos JList no tienen ítems. Los objetos JList referidos a Cursos Ofertados y Costo (objetos ubicados al lado izquierdo del diseño del formulario) se llenarán de ítems de acuerdo a lo seleccionado en la lista desplegadle referido a la Categoría de Cursos. Los objetos Jlist referidos a Cursos Escogidos y Costos se llenarán en la medida que se seleccione un curso ofertado y se agregue con el botón de comando **Agregar**. Los 4 objetos JList deben tener un modelo (**model**) cada uno, para ello debes definir 4 variables: modelo1, modelo2, modelo3 y modelo4 del tipo ***DefaultListModel().*** Para poder hacer uso de la clase ***DefaultListModel*** se debe hacer uso del paquete javax.swing.*; y debe ser escrito después del paquete Aplicaciones.

Escribe esta línea de código →

```
package aplicaciones;
import javax.swing.*;
```

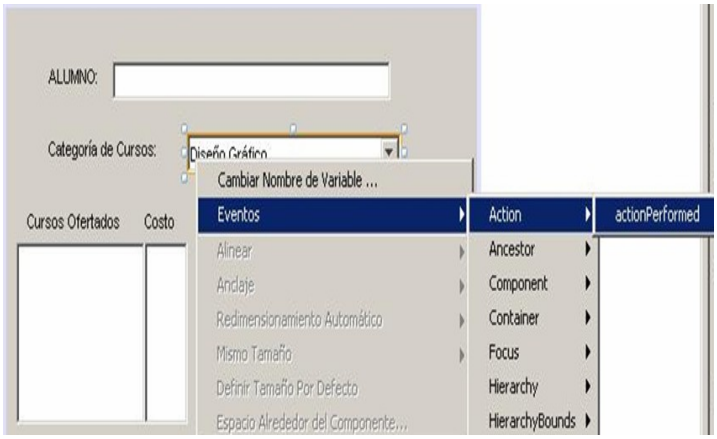
Ahora procede a escribir la definición de las variables del tipo DefaultListModel en la clase frmCursosExtension.

```
public class frmCursosExtension extends javax.swing.JFrame {
    // crear un model para cada uno de las 4 listas
    private DefaultListModel modelo1=new DefaultListModel();
    private DefaultListModel modelo2=new DefaultListModel();
    private DefaultListModel modelo3=new DefaultListModel();
    private DefaultListModel modelo4=new DefaultListModel();
```

Agrega éstas cuatro líneas de código

- Una vez definido las 4 variables de memoria del tipo ***DefaultListModel***, en el **método constructor** se debe indicar que la variable **modelo1** es para la caja de lista **lstCurOfer**, la variable **modelo2** para la caja de lista **lstCostos**, la variable **modelo3** es para la caja de lista **lstCurEsco** y la variable **modelo4** es para la caja de lista **lstCos**. También hacemos que los botones de comando Agregar y Eliminar se inhabilite su uso desde la ejecución de la aplicación. Además le indicamos una localización dentro de la pantalla y el tamaño del formulario (sólo escribe lo que señala la llave de color rojo).

```
public frmCursosExtension()
{
    initComponents();
    lstCurOfer.setModel(modelo1);
    lstCostos.setModel(modelo2);
    lstCurEsco.setModel(modelo3);
    lstCos.setModel(modelo4);
    btnAgregar.setEnabled(false);
    btnEliminar.setEnabled(false);
    setSize(450,400);
    setLocation(250,250);
}
```



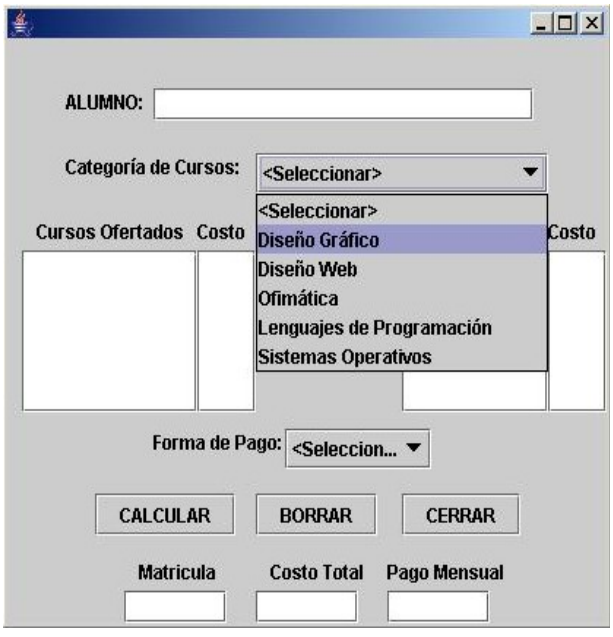

- Ahora le toca el turno en la programación al objeto JComboBox denominado cboCategorias en el evento ActionPerformed, para ello debes seleccionar el objeto mencionado y dando clic botón derecho del mouse eliges Events y luego como Actions seleccionas ActionPerformed.

En el evento mencionado programa lo siguiente (sólo escribe lo que señala la llave de color rojo):

```
private void cboCategoriasActionPerformed(java.awt.event.ActionEvent evt) {
    int indice;
    indice=cboCategorias.getSelectedIndex();
    switch(indice)
    {
        case 0:
            modelo1.clear(); modelo2.clear();
            break;
        case 1:
            modelo1.clear(); modelo2.clear();
            modelo1.addElement("Corel Draw"); modelo2.addElement("180");
            modelo1.addElement("Photo Shop"); modelo2.addElement("170");
            break;
        case 2:
            modelo1.clear(); modelo2.clear();
            modelo1.addElement("Diseño con HTML"); modelo2.addElement("100");
            modelo1.addElement("JavaScript"); modelo2.addElement("120");
            modelo1.addElement("PHP y MySQL"); modelo2.addElement("180");
            modelo1.addElement("ASP .Net"); modelo2.addElement("180");
            break;
        case 3:
            modelo1.clear(); modelo2.clear();
            modelo1.addElement("Word"); modelo2.addElement("120");
            modelo1.addElement("Excel"); modelo2.addElement("130");
            modelo1.addElement("Power Point"); modelo2.addElement("100");
            modelo1.addElement("Visio"); modelo2.addElement("110");
            break;
        case 4:
            modelo1.clear(); modelo2.clear();
            modelo1.addElement("Visual Basic .Net"); modelo2.addElement("240");
            modelo1.addElement("Visual Java"); modelo2.addElement("290");
            modelo1.addElement("Power Builder"); modelo2.addElement("230");
            modelo1.addElement("Visual C# .Net"); modelo2.addElement("220");
            break;
        case 5:
            modelo1.clear(); modelo2.clear();
            modelo1.addElement("Windows Vista"); modelo2.addElement("160");
            modelo1.addElement("Linux"); modelo2.addElement("180");
            break;
    }
}
```

Se declara una variable de memoria llamada **indice** para que almacene el índice actual del ítem seleccionado del objeto JComboBox llamado **cboCategorias**. Sabiendo el valor del índice actual o activo se hace uso de una sentencia selectiva **switch** que evalúa cuál de los ítems ha sido seleccionado. Se sabe que el primer ítem de la lista desplegadle es <*Seleccionar*> y le corresponde el índice 0, *Diseño Gráfico* el índice 1, *Diseño Web* el índice 2 y así sucesivamente. Cuando sea <*Seleccionar*> solo se procede a limpiar los objetos **lstCurOfer** y **lstCostos** a través del método **clear()** aplicados a las variables **modelo1** y **modelo2**. Si el ítem seleccionado es *Diseño Gráfico* se procede a limpiar los objetos **lstCurOfer** y **lstCostos** y se agrega los nombres de los cursos *Corel Draw* y *Photo Show* con sus respectivos costos a través del uso del método **addElement** aplicados a las variables **modelo1** y **modelo2** que tienen relación directa con los objetos **lstCurOfer** y **lstCostos**. De igual forma se trabaja para los demás ítems del objeto JComboBox llamado **cboCategorias**.

- Si en estos momentos procedes a ejecutar la aplicación se mostrará el formulario de la siguiente manera:





Y si seleccionas el ítem Diseño Gráfico se visualizará los cursos con sus respectivos costos en los JList del lado izquierdo del diseño del formulario.

Salte de la ejecución y continuemos con la programación.

- Selecciona el objeto **lstCurOfer** y ubícate en el evento *ValueChaged* perteneciente a **ListSelection** y éste a su vez pertenece a **Events**. Recuerda que esto se hace seleccionando al objeto **lstCurOfer** y dando clic botón derecho del mouse se muestra un menú flotante. En el evento mencionado programa lo siguiente:

```
private void lstCurOferValueChanged(javax.swing.event.ListSelectionEvent evt) {
    int indice=lstCurOfer.getSelectedIndex();
    lstCostos.setSelectedIndex(indice);
    btnAgregar.setEnabled(true);
}
```

Agrega éstas 3 líneas de código

Se declara una variable entera llamada índice, esta variable recibe el valor del índice del ítem seleccionado gracias al método **getSelectedIndex()**. Lo que se quiere es seleccionar el ítem de la caja de lista de lstCostos que tenga el mismo índice que **lstCurOfer**, para ello se usa el método ***setSelectedIndex(indice)*** para dar el mismo índice a la caja de lista **lstCostos**. Finalmente hacemos que el botón de comando Agregar se habilite con el método ***setEnabled()***

- A continuación escribe el siguiente código en el botón de comando **Agregar** (sólo escribe lo que señala la llave de color rojo):

```
private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
    String curso, costo;
    int total, i;
    curso=(String)lstCurOfer.getSelectedValue();
    costo=(String)lstCostos.getSelectedValue();
    modelo3.addElement(curso);
    modelo4.addElement(costo);
    btnAgregar.setEnabled(false);
}
```

Aquí declaramos dos variables de tipo *String* llamados **curso** y **costo** y las variables enteras **total** y **i**. En la variable **curso** se almacena el curso seleccionado en la lista **lstCurOfer**, el método ***getSelectionValue()*** trae consigo el ítem seleccionado pero lo trae como objeto y al poner ***(String)*** hacemos que se convierta en cadena de caracteres para que pueda ser asignada a la variable **curso**. De igual manera se hace con la variable **costo**. Para agregar un ítem a una caja de lista se usa el método ***addElement*** perteneciente al objeto variable **modelo3** o **modelo4**. Finalmente inhabilitamos el botón de comando **Agregar**.

- Ahora programa en la caja de lista lstCurEsco en el evento ***ValueChanged*** cuando desees seleccionar un curso para luego eliminarlo (sólo escribe lo que señala la llave de color rojo).

```
private void lstCurEscoValueChanged(javax.swing.event.ListSelectionEvent evt) {
    int indice=lstCurEsco.getSelectedIndex();
    lstCos.setSelectedIndex(indice);
    btnEliminar.setEnabled(true);
}
```

La idea es la misma que se aplicó en la programación sobre el evento ***ValueChanged*** del objeto **lstCurOfer**. La diferencia está en que esta vez se habilita el uso del botón de comando **Eliminar**.

- En el botón de comando eliminar colocas la siguiente programación (sólo escribe lo que señala la llave de color rojo):

```
private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
    int indice=lstCurEsco.getSelectedIndex();
    modelo3.remove(indice);
    modelo4.remove(indice);
    btnEliminar.setEnabled(false);
}
```

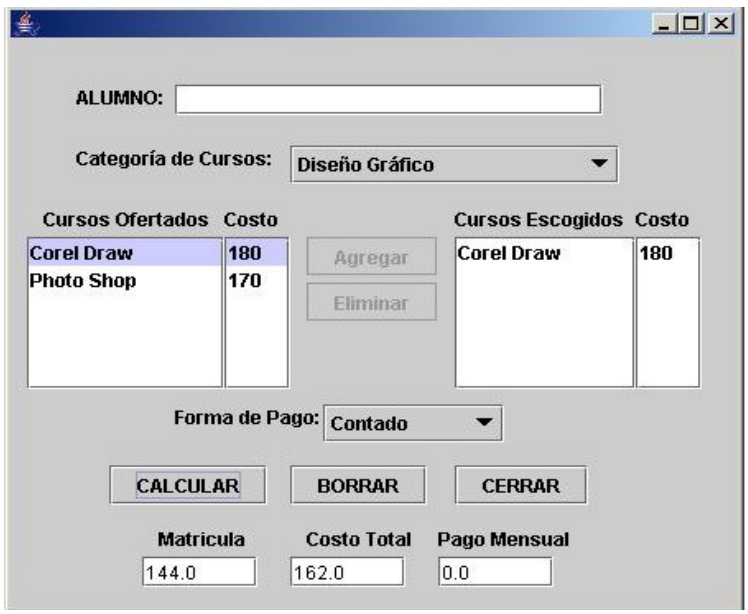
En esta programación se hace uso del método ***remove()*** que elimina un ítem de la lista a través de la variable objeto **modelo3** ó **modelo4** dado el valor del índice. Al final de la programación se inhabilita el botón de comando **Eliminar**.

- Ahora procede a programar en el botón de comando **Calcular** (sólo escribe lo que señala la llave de color rojo):

```
private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt) {
    int cuenta=modelo3.size(), i;
    float mat=0, ctotal=0, pmen=0;
    for (i=0;i<cuenta;i++)
    {
        ctotal=ctotal+Float.parseFloat((String)modelo4.elementAt(i));
    }
    if (ctotal==0)
    {
        JOptionPane.showMessageDialog(null,"Selecciona y agrega al menos un curso");
        return;
    }
    if (cuenta==1)
        mat=ctotal*0.8f;
    if (cuenta==2)
        mat=ctotal*0.6f;
    if (cuenta>=3)
        mat=ctotal*0.5f;
    if (cboFPago.getSelectedItem().equals("<Seleccionar>"))
    {
        JOptionPane.showMessageDialog(null,"Selecciona una forma de pago");
        return;
    }
    if (cboFPago.getSelectedItem().equals("Contado"))
    {
        ctotal=ctotal*0.9f;
        pmen=0;
    }
    if (cboFPago.getSelectedItem().equals("Crédito"))
    {
        ctotal=ctotal*1.1f;
        JOptionPane.showMessageDialog(null,"Ctotal es"+ctotal);
        pmen=ctotal/4;
        JOptionPane.showMessageDialog(null,"Pmen es"+pmen);
    }
    txtMat.setText(String.valueOf(mat));
    txtCTotal.setText(String.valueOf(ctotal));
    txtPMen.setText(String.valueOf(pmen));
}
```

Se declara la variable de memoria cuenta que almacena la cantidad de ítems existentes en la caja de lista **lstCurEsco** a través del uso del método **Size()** aplicado a la variable **modelo3**. También se declara una variable de memoria **i** de tipo entero y tres variables de tipo float para el cálculo de la matrícula, el costo total y el pago mensual. A través de una sentencia ***for*** se procede a obtener los valores de la caja de lista **lstCos** usando la variable **modelo4** con el método **elementAt(),** para que estos valores sean sumados ya acumulados en la variable de memoria **ctotal**. En la primera sentencia ***if*** se procede a averiguar si las cajas de listas lstCurEsco y LstCos tiene ítems, si no tienen ítems se visualiza un mensaje de error indicando la necesidad de seleccionar y agregar cursos y se procede a suspender la ejecución del programa gracias a instrucción *return* (retornar). En los siguientes tres ***if*** se calcula el monto de la matrícula aplicando el porcentaje indicado en el enunciado de la aplicación. En la siguiente sentencia if se evalúa si se seleccionó una forma de pago y si no se logró hacerlo muestra un mensaje de error y suspende la ejecución del programa. Luego con las siguientes sentencias ***if*** se evalúa la forma de pago y se procede hacer los cálculos respectivos. Finalmente se los resultados en las variables de memoria de tipo float se envían a los objetos JTextField par ser visualizados en el formulario.

- La programación en los botones de comando Borrar y Cerrar es de la misma forma como se aplicó en los temas o sesiones anteriores. Cuando procedas a ejecutar tu aplicación se debe visualizar el formulario y una vez interactuado se mostrarán los resultados.



# USO DEL OBJETO JTABLE

## Objeto de Control JTable

Como programadores, sabemos muy bien que la presentación de datos tabulados es una de las tareas más comunes que se presentan al momento de crear interfaces gráficas; desde la simple tabla que permite únicamente mostrar el resultado de una consulta, hasta las que permiten editar directamente el contenido de cada celda, ordenar las columnas, personalizar su apariencia, etc. Todas las tareas antes descritas, y muchas otras, son posibles de realizar utilizando la clase JTable; por supuesto, mientras más complejo sea el requerimiento a cubrir, se requerirá en igual medida utilizar más métodos o recursos de la clase.

Los *modelos de tabla* son objetos que implementan la *interface* TableModel; a través de ellos es posible personalizar mucho más y mejor el comportamiento de los componentes Jtable, permitiendo utilizar al máximo sus potencialidades.

El siguiente gráfico intenta mostrar como cada componente JTable obtiene siempre sus datos desde un *modelo de tabla*.


La clase AbstractTableModel es la que implementa directamente a la interface TableModel, aunque es esta clase la que se recomienda *extender* para utilizarla como *modelo de tabla*, existe un *modelo de tabla* predeterminado que facilita mucho el trabajo con tablas. Este modelo predeterminado es la clase DefaultTableModel.

**Propiedad más usada:**

- Model: Permite definir el número de columnas y filas del objeto como también las expresiones que irán en las columnas.

**Métodos más usados:**

- addColumn(): Añade la columna al final de la matriz de columnas.
- setModel(): Asigna el modelo de datos al objeto JTable.
- GetRowCount(): Devuelve el número de filas en la tabla.

## DefaultTableModel

Esta clase permite construir el modelo para el objeto Table. Los métodos más utilizados son:

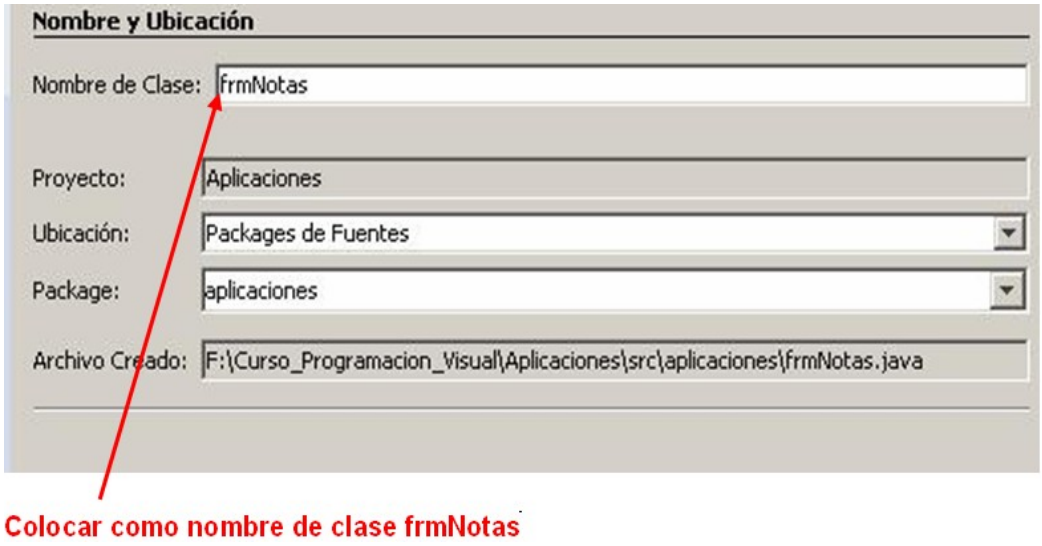
- addColumn(): Añade una columna al modelo.
- AddRow(): Añade una fila al final del modelo.
- getColumnCount(): Devuelve el número de columnas en esta tabla de datos.
- getRowCount(): Devuelve el número de filas en esta tabla de datos.
- getValueAt(): Devuelve un valor de atributo para la celda en la posición row, column.
- insertRow(): Inserta una fila en el modelo.
- RemoveRow(): Elimina del modelo según la posición de la fila indicada.

## Aplicación

Construir una aplicación que permita calcular el promedio de las notas obtenidas en el curso de Programación Visual. La aplicación debe permitir el ingreso del nombre del alumno, la nota de la I Unidad, la nota de la II Unidad y la nota de la III Unidad. Además debe permitir la selección del turno a la que pertenece el alumno. A través de un botón de comando debe agregar los datos en un objeto Jtable, calculando el promedio de las notas; y a través de otro botón de comando debe eliminar la fila seleccionada en el objeto JTable. También se debe mostrar el total de filas agregadas en el objeto JTable.

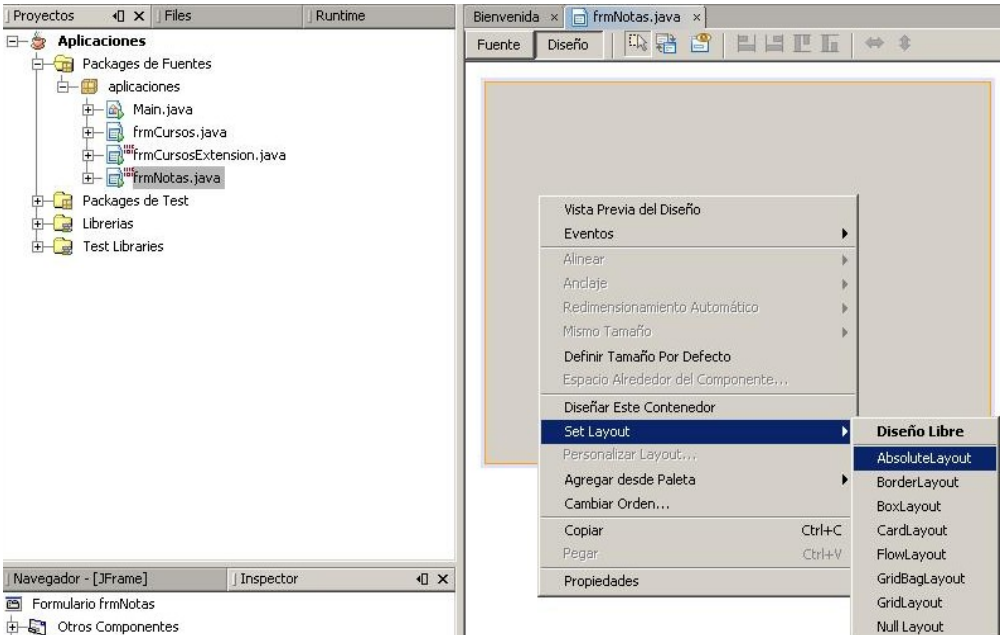
## Solución:

- Usaremos el mismo proyecto utilizado en la sesión anterior y sólo agregarás un formulario (Jframe). → Inmediatamente se muestra la siguiente ventana:



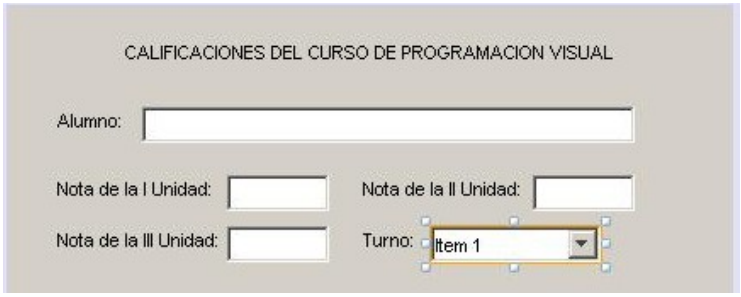

Luego dar clic en el botón de comando ***Finish***.

- A continuación se muestra el entorno de desarrollo de NetBeans y no olvides de dar clic en el botón derecho del mouse sobre el formulario y establece ***AbsoluteLayout*** en ***Set Layout***.

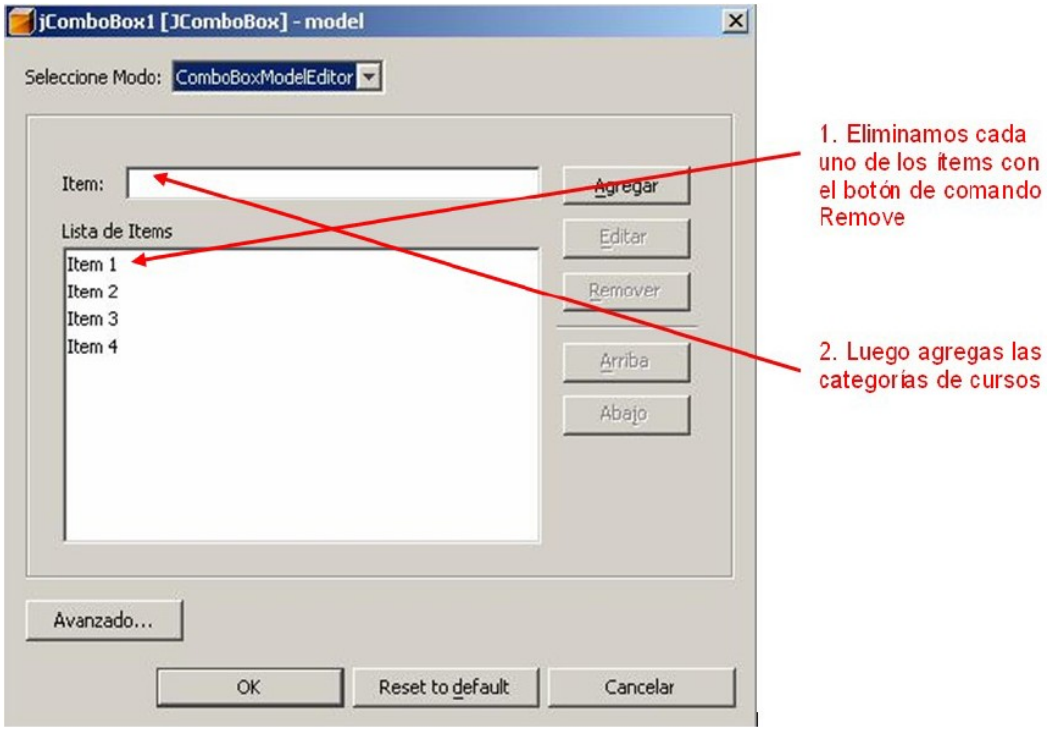



- Procedemos a colocar un objeto Jlabel con la expresión "CALIFICACIONES DEL CURSO DE PROGRAMACION VISUAL". Debajo de éste título ubicar un objeto Jlabel con la expresión "ALUMNO:" acompañado de un cuadro de texto (JTextField) . A continuación, colocar otro Jlabel con la expresión "Nota de la I

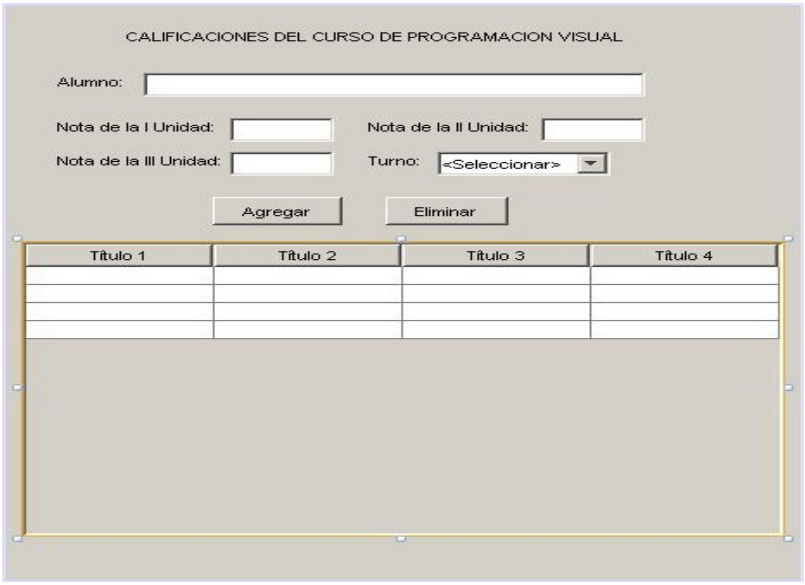
Unidad" acompañado de un cuadro de texto y de igual manera hacerlo para la segunda y tercera unidad. Luego agregamos un objeto JCombobox para seleccionar el turno.



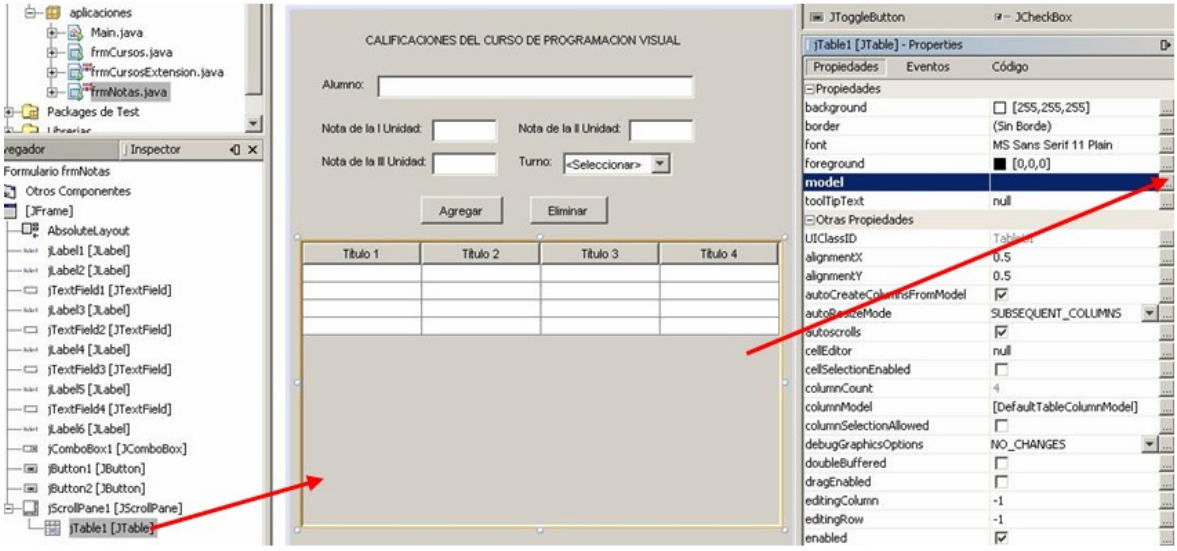

- Seleccionamos el objeto JComboBox y elegimos en la ventana de propiedades, la propiedad ***model*** que permite colocar los ítems dentro de la caja de lista desplegadle. Elegimos el botón de comando referido a la propiedad model se muestra la siguiente ventana:



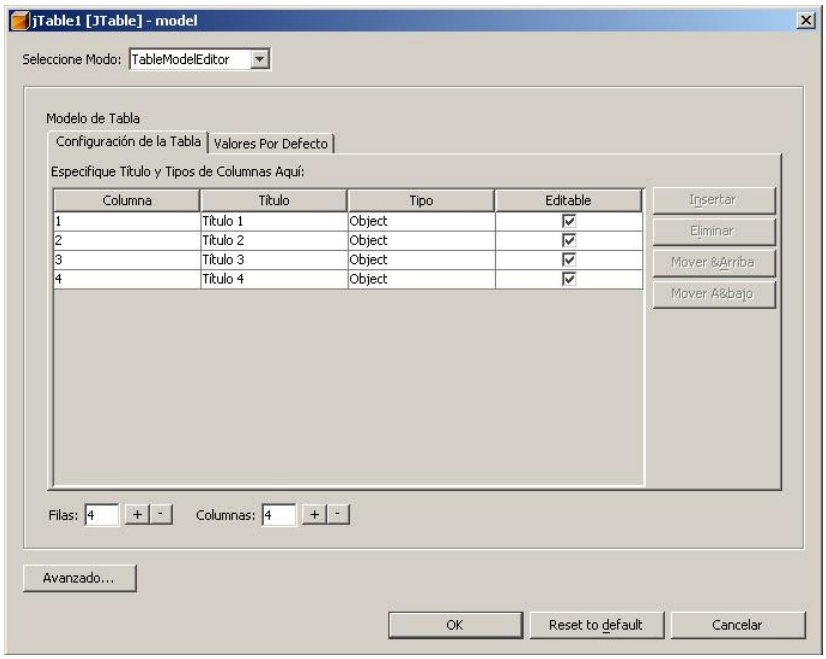

- Ingresamos "<Seleccionar>","Mañana", "Tarde" y "Noche" y luego hacemos click en el botón de comando OK. Continuamos con el diseño del formulario agregando un botón de comando "Agregar" y un botón de comando "Eliminar". Luego procedemos a agregar el objeto JTable.



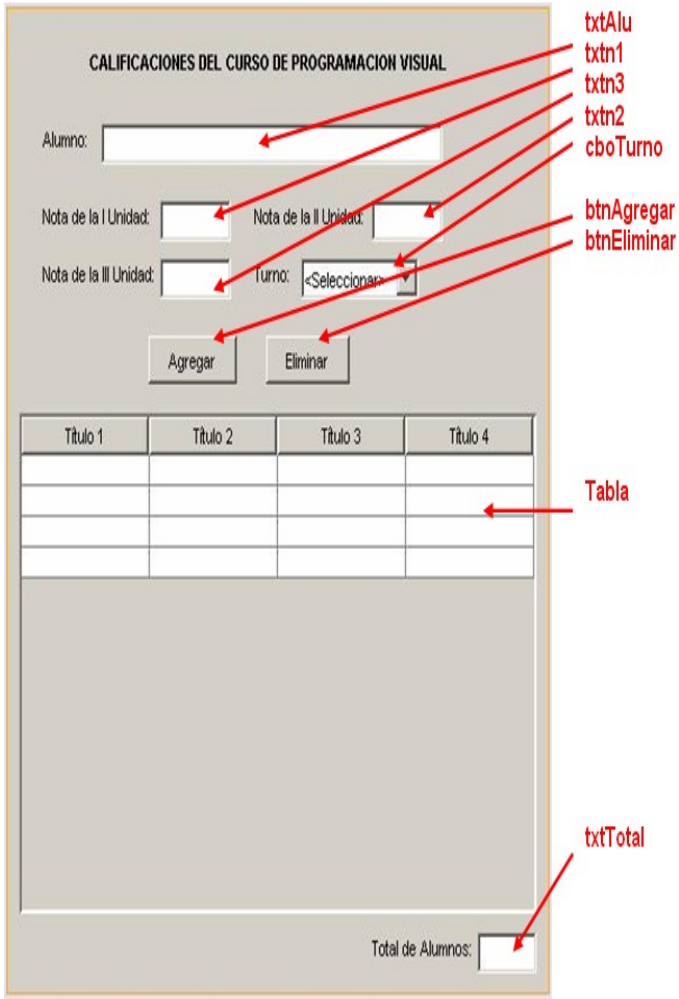

- Al ser dibujado el objeto JTable se observa en el panel de la izquierda que se vincula a un objeto JScrollPane. El objeto Jtable tiene como propiedad principal a ***model.***




- Luego de seleccionar el botón de comando referido a la propiedad ***model*** se muestra la siguiente ventana:




- Observamos en la ventana anterior que por defecto el objeto Table propone la conformación de 4 columnas y 4 filas, dando la posibilidad de aumentar o disminuir el número de columnas y filas. Además podemos establecer los títulos de cada columna. Aquí debemos hacer hincapié que las columnas y las filas son tipo ***Object*** esto quiere decir que un objeto JTable es una matriz de objetos (arreglo bidimensional). Nosotros vamos a establecer el número de columnas a través de la programación y las filas se crearán en la medida que se necesiten.
- Los nombres de los objetos de control dibujados en el formulario queda de la siguiente manera:



- Vamos a proceder a programar. Comenzamos con hacer uso del paquete swing y específicamente a las clases JoptionPane y a la clase JTable.

```
package aplicaciones;
//agregar estos dos lineas de codigo
import javax.swing.JOptionPane;
import javax.swing.table.*;
```

Agregamos estas dos líneas de código

- Luego procedemos a crear un modelo para el objeto JTable llamado ***Tabla*** a través de la clase ***DefaultTableModel.*** Lo hacemos dentro de la clase frmNotas. Usar la clase DefaultTableModel es posible gracias al import javax.swing.table.*;

```
public class frmNotas extends javax.swing.JFrame {
    DefaultTableModel dtm = new DefaultTableModel();
```

Declaramos y creamos una variable de memoria ***dtm*** del tipo DefaultTableModel.

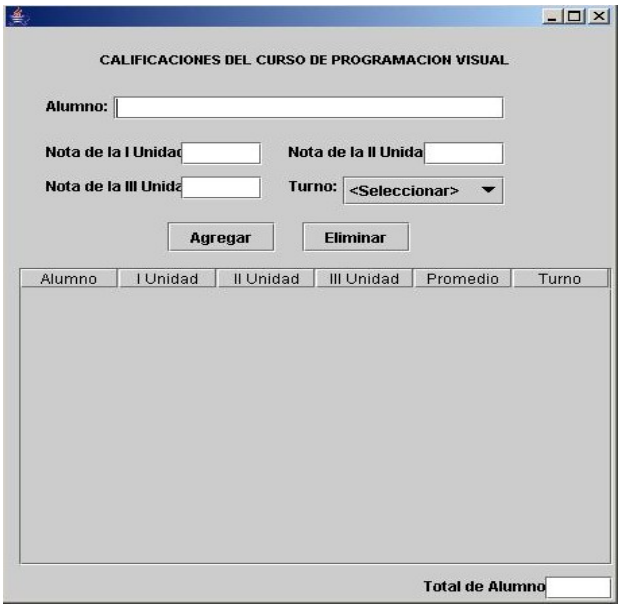
- En el método constructor programamos lo siguiente (sólo escribe lo que se señala la llave de color rojo):

```
public frmNotas() {
    initComponents();
    String titulos[]={"Alumno","I Unidad","II Unidad","III Unidad","Promedio","Turno"};
    dtm.setColumnIdentifiers(titulos);
    Tabla.setModel(dtm);
}
```

Declaramos y creamos una variable de memoria ***titulos*** del tipo cadena y es un arreglo. Esta variable ***titulos*** se inicializa con los valores "Alumno", "I Unidad", "II Unidad", "III Unidad", "Promedio" y "Turno", que serán los títulos de las columnas del objeto JTable. Luego, con el método ***setColumnIdentifiers()*** se define las columnas con sus respectivos títulos en la variable ***dtm*** (modelo del JTable llamado ***Tabla***). Finalmente, se vincula el modelo, representado en la variable ***dtm***, al objeto JTable llamado ***Tabla***.

- Si en estos momentos decidimos ejecutar nuestra aplicación, quedaría nuestro formulario así:

Formulario ⟶




Observamos que el objeto JTable muestra las columnas definas en la programación hecha en el método constructor.

- Procedamos con la programación del botón de comando Agregar (sólo escribe lo que se señala la llave de color rojo).

```
private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
    String datos[]= new String[6];
    int n1,n2,n3,total;
    double promedio;
    boolean verifica;
    n1=Integer.parseInt(txtn1.getText());
    n2=Integer.parseInt(txtn2.getText());
    n3=Integer.parseInt(txtn3.getText());
    verifica=txtAlu.getText().equals("") || txtn1.getText().equals("") || txtn2.getText().equals("") || txtn3.getText().equals("");
    verifica=verifica || cboTurno.getSelectedIndex()==0;
    if (!verifica)
    {
        promedio=(double)(n1+n2+n3)/(double)3;
        datos[0]=txtAlu.getText();
        datos[1]=txtn1.getText();
        datos[2]=txtn2.getText();
        datos[3]=txtn3.getText();
        datos[4]=String.valueOf(promedio);
        datos[5]=(String)cboTurno.getSelectedItem();
        dtm.addRow(datos);
        txtAlu.setText("");
        txtn1.setText("");
        txtn2.setText("");
        txtn3.setText("");
        cboTurno.setSelectedIndex(0);
    }
    else
    {
        JOptionPane.showMessageDialog(null,"Falta el ingreso de algún dato");
    }
    total=dtm.getRowCount();
    txtTotal.setText(String.valueOf(total));
}
```

Declaramos y creamos una variable de memoria ***datos*** de tipo String y de tamaño 6. Luego, declaramos las variables de memoria ***n1***, ***n2***, ***n3*** y ***total*** de tipo entero, la variable ***promedio*** de tipo double y una variable de memoria ***verifica*** de tipo booleano. Las variables de memoria ***n1***, ***n2*** y ***n3*** reciben los valores ingresado en los cuadros de textos ***txtn1***, ***txtn2*** y ***txtn3*** respectivamente. Con la variable de memoria verifica se pretende evaluar si se llegó a escribir en los cuadros de textos y se haya seleccionado un turno. Con la sentencia IF evaluamos la variable verifica y con el operador **!** hacemos negación, es decir, si la variable **verifica** es falso entonces con ! se convierte en verdadero. Si la variable ***verifica*** es falso significa que se ingresó los datos en los cuadros de textos y se seleccionó el turno, entonces procedemos a calcular el promedio teniendo presente que las variables ***n1***,***n2***, ***n3*** siendo enteras deben ser tratadas como reales (double). Posteriormente, hacemos uso del arreglo datos asignando los datos ingresados y el turno seleccionado en cada uno de los elementos. Con el método ***addRow()*** logramos crear una fila con los valores contenidos con el vector o arreglo ***datos***. Luego, limpiamos los cuadros de textos y hacemos que el objeto JComboBox quede en <Seleccionar> al dar el valor cero al método ***setSelectedIndex()***. Si la variable ***verifica*** es verdadero significa que falta ingresar algún dato o seleccionar el turno. Finalmente, se muestra la cantidad de filas agregadas en el cuadro de texto ***txtTotal*** y haciendo uso del método ***setRowCount()*** perteneciente a ***dtm***.

- Procedamos con la programación del botón de comando Eliminar.

```
private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
    int fila,total;
    fila=Tabla.getSelectedRow();
    if (fila>=0)
       dtm.removeRow(fila);
    else
       JOptionPane.showMessageDialog(null,"Selecciona una fila de la Tabla para eliminar");
    total=dtm.getRowCount();
    txtTotal.setText(String.valueOf(total));
}
```

Declaramos las variables ***fila*** y ***total*** de tipo entero. La variable fila se le asigna el valor de la posición de la fila seleccionada en el objeto Jtable llamado ***Tabla***. Con la sentencia IF se evalúa a la variable ***fila*** si es mayor o igual a cero procedemos a remover o borrar la fila previamente seleccionad, caso contrario se muestra un mensaje indicando que se debe seleccionar una fila en la Tabla. Finalmente, se muestra la cantidad de filas agregadas en el cuadro de texto ***txtTotal*** y haciendo uso del método ***setRowCount()*** perteneciente a ***dtm***.

- Procedemos finalmente a ejecutar el formulario.