

Fundación Código Libre Dominicano

Lic. Henry Terrero.
hterrero@codigolibre.org

Ing. Jose Paredes.
jparedes@codigolibre.org



Desarrollo De aplicaciones Java Con Netbeans

EJERCICIO GUIADO. JAVA: INTRODUCCIÓN A LA POO

Introducción a la Programación Orientada a Objetos

La programación orientada a objetos es una nueva forma de entender la creación de programas. Esta filosofía de programación se basa en el concepto de objeto.

Objeto

Un objeto se define como un elemento que tiene unas **propiedades** y **métodos**.

Un objeto posee unas características (ancho, alto, color, etc...) A las características de un objeto se les llama **propiedades**.

Un objeto es un elemento "inteligente". A un objeto se le puede dar órdenes y él obedecerá. A estas órdenes se les llama **métodos**. Con los métodos podemos cambiar las características del objeto, pedirle información, o hacer que el objeto reaccione de alguna forma.

En Java todo son objetos. Veamos algunos ejemplos de uso de objetos en Java:

Ejemplo 1

Supongamos que tenemos una etiqueta llamada *etiTexto*. Esta etiqueta **es un objeto**.

Como objeto que es, la etiqueta *etiTexto* tiene una serie de características, como por ejemplo: el color de fondo, el tamaño, la posición que ocupa en la ventana, el ser opaca o no, el ser invisible o no, etc... Son las *propiedades* de la etiqueta.

A una etiqueta le podemos dar órdenes, a través de *métodos*.

A través de los métodos podemos por ejemplo cambiar las características del objeto. Por ejemplo, se puede cambiar el tamaño y posición de la etiqueta usando el método *setBounds*:

```
etiTexto.setBounds(10, 20, 100, 20);
```

Normalmente, los métodos que permiten cambiar las características del objeto son métodos cuyo nombre empieza por *set*.

Los métodos también permiten pedirle al objeto que me de información. Por ejemplo, podríamos usar el conocido método *getText* para recoger el texto que contenga la etiqueta y almacenarlo en una variable:

```
String texto;  
texto = etiTexto.getText();
```

Los métodos que le piden información al objeto suelen tener un nombre que empieza por *get*.

Los métodos también sirven para ordenarle al objeto que haga cosas. Por ejemplo, podemos ordenar a la etiqueta *etiTexto* que se vuelva a pintar en la ventana usando el método *repaint*:

```
etiTexto.repaint();
```

Ejemplo 2

Supongamos que tenemos un cuadro de texto llamado *txtCuadro*. Como todo en Java, un cuadro de texto es un **objeto**.

Un objeto tiene **propiedades**, es decir, características. Nuestro cuadro de texto *txtCuadro* tiene características propias: un color de fondo, un ancho, un alto, una posición en la ventana, el estar activado o no, el estar visible o no, etc...

A un objeto se le puede dar órdenes, llamadas **métodos**. Estas órdenes nos permiten cambiar las características del objeto, pedirle información, o simplemente pedirle al objeto que haga algo.

Por ejemplo, podemos cambiar el color de fondo del cuadro de texto *txtCuadro* usando el método llamado *setBackground*:

```
txtCuadro.setBackground(Color.RED);
```

Otros métodos que permiten cambiar las propiedades del objeto *txtCuadro* son:

<i>setVisible</i>	- permite poner visible / invisible el cuadro de texto
<i>setEnabled</i>	- permite activar / desactivar el cuadro de texto
<i>setEditable</i>	- permite hacer que se pueda escribir o no en el cuadro de texto
<i>setText</i>	- permite introducir un texto en el cuadro de texto
<i>setBounds</i>	- permite cambiar el tamaño y posición del objeto
<i>setToolTipText</i>	- permite asociar un texto de ayuda al cuadro de texto
<i>etc...</i>	

Un objeto nos da información sobre él. Para pedirle información a un objeto usaremos métodos del tipo *get*. Por ejemplo, para pedirle al cuadro de texto el texto que contiene, usaremos el método *getText*:

```
String cadena = txtCuadro.getText();
```

Otros métodos que le piden información al cuadro de texto son:

<i>getWidth</i>	- te dice la anchura que tiene el cuadro de texto
<i>getHeight</i>	- te dice el alto que tiene el cuadro de texto
<i>getSelectedText</i>	- te devuelve el texto que está seleccionado dentro del cuadro de texto
<i>getToolTipText</i>	- te dice el texto de ayuda que tiene asociado el cuadro de texto
<i>etc...</i>	

También se le puede dar al objeto simplemente órdenes para que haga algo. Por ejemplo, podemos ordenar al cuadro de texto *txtCuadro* que seleccione todo el texto que contiene en su interior a través del método *selectAll*:

```
txtCuadro.selectAll();
```

Otros métodos que ordenan al cuadro de texto son:

<i>repaint</i>	- le ordena al cuadro de texto que se vuelva a pintar
<i>copy</i>	- le ordena al cuadro de texto que copie el texto que tenga seleccionado
<i>cut</i>	- le ordena al cuadro de texto que corte el texto que tenga seleccionado
<i>paste</i>	- le ordena al cuadro que pegue el texto que se hubiera copiado o cortado
<i>etc...</i>	

Clase

Todo objeto es de una “clase” determinada, o dicho de otra forma, tiene un “tipo de datos” determinado.

Por ejemplo, las etiquetas que se usan en las ventanas son objetos que pertenecen a la clase *JLabel*. Los cuadros de texto en cambio son objetos de la clase *TextField*.

Para poder usar un objeto hay que *declararlo* y *construirlo*.

Declarar un Objeto

La declaración de un objeto es algo similar a la declaración de una variable. Es en este momento cuando se le da un nombre al objeto. Para declarar un objeto se sigue la siguiente sintaxis:

```
Clase nombreobjeto;
```

Por ejemplo, para declarar la etiqueta del ejemplo 1, se usaría el siguiente código:

```
JLabel etiTexto;
```

Para declarar, en cambio, el cuadro de texto del ejemplo 2, se usaría el siguiente código:

```
TextField txtCuadro;
```

Construir un Objeto

En el momento de la “construcción” de un objeto, se le asignan a este una serie de propiedades iniciales. Es decir, unas características por defecto. Se puede decir que es el momento en que “nace” el objeto, y este nace ya con una forma predeterminada, que luego el programador podrá cambiar usando los métodos del objeto.

Es necesario construir el objeto para poder usarlo. La construcción del objeto se hace a través del siguiente código general:

```
nombreobjeto = new Clase();
```

Por ejemplo, para construir la etiqueta del ejemplo 1, se haría lo siguiente:

```
etiTexto = new JLabel();
```

Para construir el cuadro de texto del ejemplo 2, se haría lo siguiente:

```
txtCuadro = new TextField();
```

NOTA. En algunos casos, la sintaxis de la declaración y la construcción se une en una sola línea. Por ejemplo, supongamos que queremos declarar la etiqueta *etiTexto* y construirla todo en una línea, entonces se puede hacer lo siguiente:

```
JLabel etiTexto = new JLabel();
```

En general, para declarar y construir un objeto en una sola línea se sigue la siguiente sintaxis:

```
Clase nombreobjeto = new Clase();
```

La Clase como generadora de objetos

Conociendo la Clase, se pueden crear tantos objetos de dicha Clase como se quiera. Es decir, la Clase de un objeto funciona como si fuera una plantilla a partir de la cual fabricamos objetos iguales. Todos los objetos creados a partir de una clase son iguales en un primer momento (cuando se construyen) aunque luego el programador puede dar forma a cada objeto cambiando sus propiedades.

Por ejemplo, la Clase JLabel define etiquetas. Esto quiere decir que podemos crear muchas etiquetas usando esta clase:

```
JLabel etiTexto = new JLabel();
JLabel etiResultado = new JLabel();
JLabel etiDato = new JLabel();
```

En el ejemplo se han declarado y construido tres etiquetas llamadas *etiTexto*, *etiResultado* y *etiDato*. Las tres etiquetas en este momento son iguales, pero a través de los distintos métodos podemos dar forma a cada una:

```
etiTexto.setBackground(Color.RED);
etiTexto.setText("Hola");
etiResultado.setBackground(Color.GREEN);
etiResultado.setText("Error");
etiDato.setBackground(Color.BLUE);
etiDato.setText("Cadena");
```

En el ejemplo se le ha dado, usando el método *setBackground*, un color a cada etiqueta. Y se ha cambiado el texto de cada una. Se le da forma a cada etiqueta.

EJERCICIO

Hasta ahora ha usado objetos aunque no tenga mucha conciencia de ello. Por ejemplo ha usado botones. Como ejercicio se propone lo siguiente:

- ¿Cuál es el nombre de la clase de los botones normales que usa en sus ventanas?
- ¿Cómo declararía un botón llamado *btnAceptar*, y otro llamado *btnCancelar*?
- ¿Cómo construiría dichos botones?
- Indique algunos métodos para cambiar propiedades de dichos botones (métodos set)
- Indique algunos métodos para pedirle información a dichos botones (métodos get)
- Indique algún método para dar órdenes a dichos botones (algún método que no sea ni set ni get)

CONCLUSIÓN

Un Objeto es un elemento que tiene una serie de características llamadas PROPIEDADES.

Por otro lado, al objeto se le pueden dar órdenes que cumplirá de inmediato. A dichas órdenes se les denomina MÉTODOS.

Los métodos se pueden dividir básicamente en tres tipos:

**Para cambiar las propiedades del objeto (Métodos set)
Para pedir información al objeto (Métodos get)
Para dar órdenes al objeto.**

Todo objeto pertenece a una CLASE.

La CLASE nos permite declarar objetos y construirlos:

Declaración:

CLASE nombreobjeto;

Construcción:

nombreobjeto = new CLASE();

Declaración y Construcción en una misma línea

CLASE nombreobjeto = new CLASE(),

En la construcción de un objeto se asignan unas propiedades (características) por defecto al objeto que se construye, aunque luego, estas características pueden ser cambiadas por el programador.

EJERCICIO GUIADO. JAVA: POO. CLASES PROPIAS

Objetos propios del lenguaje

Hasta el momento, todos los objetos que ha usado a la hora de programar en Java, han sido objetos incluidos en el propio lenguaje, que se encuentran disponibles para que el programador los use en sus programas.

Estos objetos son: las etiquetas (JLabel), botones (JButton), los cuadros de texto (JTextField), cuadros de verificación (JCheckBox), botones de opción (JRadioButton), colores (Color), imágenes de icono (ImageIcon), modelos de lista (DefaultListModel), etc, etc.

Todos estos objetos, por supuesto, pertenecen cada uno de ellos a una Clase:

Las etiquetas son objetos de la clase JLabel, los botones son objetos de la clase JButton, etc.

Todos estos objetos tienen propiedades que pueden ser cambiadas en la ventana de diseño:

Ancho, alto, color, alineación, etc.

Aunque también poseen métodos que nos permiten cambiar estas propiedades durante la ejecución del programa:

setText cambia el texto del objeto, setBackground cambia el color de fondo del objeto, setVisible hace visible o invisible al objeto, setBounds cambia el tamaño y la posición del objeto, etc.

En cualquier momento le podemos pedir a un objeto que nos de información sobre sí mismo usando los métodos get:

getText obtenemos el texto que tenga el objeto, getWidth obtenemos la anchura del objeto, getHeight obtenemos la altura del objeto, etc.

Los objetos son como “pequeños robots” a los que se les puede dar órdenes, usando los métodos que tienen disponible.

Por ejemplo, le podemos decir a un objeto que se pinte de nuevo usando el método repaint, podemos ordenarle a un cuadro de texto que coja el cursor, con el método requestFocus, etc.

Todos estos objetos, con sus propiedades y métodos, nos facilitan la creación de nuestros programas. Pero a medida que nos vamos introduciendo en el mundo de la programación y nos especializamos en un tipo de programa en concreto, puede ser necesaria la creación de objetos propios, programados por nosotros mismos, de forma que puedan ser usados en nuestros futuros programas.

Objetos propios

A la hora de diseñar un objeto de creación propia, tendremos que pensar qué propiedades debe tener dicho objeto, y métodos serán necesarios para poder trabajar con él. Dicho de otra forma, debemos pensar en qué características debe tener el objeto y qué órdenes le podré dar.

Para crear objetos propios hay que programar la Clase del objeto. Una vez programada la Clase, ya podremos generar objetos de dicha clase, declarándolos y construyéndolos como si de cualquier otro objeto se tratara.

A continuación se propondrá un caso práctico de creación de objetos propios, con el que trabajaremos en las próximas hojas.

Lo que viene a continuación es un planteamiento teórico de diseño de una Clase de Objetos.

CASO PRÁCTICO: MULTICINES AVENIDA

Planteamiento

Los Multicines Avenida nos encargan un programa para facilitar las distintas gestiones que se realizan en dichos multicines.

El multicine cuenta con varias salas, y cada una de ellas genera una serie de información. Para facilitar el control de la información de cada sala programaremos una Clase de objeto a la que llamaremos **SalaCine**.

La Clase SalaCine

La Clase *SalaCine* definirá características de una sala de cine, y permitirá crear objetos que representen salas de cine. Cuando la Clase *SalaCine* esté programada, se podrán hacer cosas como las que sigue:

Los Multicines Avenida tienen una sala central donde se proyectan normalmente los estrenos. Se podría crear un objeto llamado *central* de la clase *SalaCine* de la siguiente forma:

```
SalaCine central;
```

Por supuesto, este objeto puede ser construido como cualquier otro:

```
central = new SalaCine();
```

El objeto *central* representará a la sala de cine central de los Multicines Avenida.

Otro ejemplo. Los Multicines Avenida tienen una sala donde proyectan versiones originales. Se podría crear un objeto llamado *salaVO* de la clase *SalaCine* de la siguiente forma:

```
SalaCine salaVO; //declaración  
  
salaVO = new SalaCine(); //construcción
```

Propiedades de los objetos de la clase SalaCine

A la hora de decidir las propiedades de un objeto de creación propia, tenemos que preguntarnos, ¿qué información me interesa almacenar del objeto? Trasladando esta idea a nuestro caso práctico, ¿qué información me interesaría tener de cada sala de cine?

Para este ejemplo supondremos que de cada sala de cine nos interesa tener conocimiento de las siguientes características (propiedades):

- **Aforo:** define el número de butacas de la sala (un número entero).
- **Ocupadas:** define el número de butacas ocupadas (un número entero).
- **Película:** define la película que se está proyectando en el momento en la sala (una cadena de texto)
- **Entrada:** define el precio de la entrada (un número double)

Valores por defecto de los objetos *SalaCine*

Cuando se construye un objeto, se asignan unos valores por defecto a sus propiedades. Por ejemplo, cuando se construye una etiqueta (Clase `JLabel`), esta tiene un tamaño inicial definido, un color, etc.

Lo mismo se tiene que hacer con los objetos propios que definimos. Es necesario decidir qué valores tendrá las propiedades del objeto al construirse.

En nuestro caso, las características de un objeto *SalaCine* en el momento de construirse serán las siguientes:

Aforo: 100
Ocupadas: 0
Película: "" (la cadena vacía)
Entrada: 5,00

Observa este código, en él construimos el objeto correspondiente a la sala central del multicine:

```
SalaCine central;  
  
central = new SalaCine();
```

En este momento (en el que el objeto *central* está recién construido) este objeto tiene asignado un aforo de 100, el número de butacas ocupadas es 0, la película que se proyecta en la sala central es "" y la entrada para esta sala es de 5 euros.

Los valores por defecto que se asignan a los objetos de una clase son valores arbitrarios que el programador decidirá según su conveniencia.

Métodos de los objetos *SalaCine*

Para comunicarnos con los objetos de la Clase *SalaCine* que construyamos, tendremos que disponer de un conjunto de métodos que nos permitan asignar valores a las propiedades de los objetos, recoger información de dichos objetos y que le den órdenes al objeto.

Será el programador el que decida qué métodos le interesará que posea los objetos de la Clase que está programando.

Para nuestro caso particular, supondremos que los objetos de la Clase *SalaCine* deberán tener los siguientes métodos:

Métodos de cambio de propiedades (Métodos *set*)

setAforo	- asignará un aforo a la sala de cine
setOcupadas	- asignará una cantidad de butacas ocupadas a la sala de cine
setLibres	- asignará una cantidad de butacas libres a la sala de cine
setPelícula	- asignará un título de película a la sala de cine
setEntrada	- fijará el precio de las entradas a la sala de cine

Gracias a estos métodos podemos dar forma a un objeto *SalaCine* recién creado.

Por ejemplo, supongamos que queremos crear el objeto que representa la sala de versiones originales. Resulta que esta sala tiene de aforo 50 localidades, que se está proyectando la película "Metrópolis" y que la entrada para ver la película es de 3 euros. La sala está vacía de momento.

Para crear el objeto, se usaría el siguiente código:

```
//Se construye el objeto
SalaCine salaVO;
salaVO = new SalaCine();

//Se le asignan características
salaVO.setAforo(50);           //aforo 50
salaVO.setPelicula("Metrópolis"); //la película que se proyecta
salaVO.setEntrada(3);          //entrada a 3 euros
```

Al construir el objeto *salaVO* tiene por defecto los valores siguientes en sus propiedades:

Aforo: 100
Ocupadas: 0
Película: ""
Entrada: 5,00

Gracias a los métodos disponibles hemos asignados estos nuevos valores:

Aforo: 50
Ocupadas: 0
Película: "Metrópolis"
Entrada: 3,00

Métodos para pedirle información al objeto (Métodos get)

Se programarán los siguientes métodos get en la clase *SalaCine*:

getAforo	- devuelve el aforo que tiene el objeto
getOcupadas	- devuelve el número de butacas ocupadas que tiene el objeto
getLibres	- devuelve el número de butacas que tiene libres el objeto
getPorcentaje	- devolverá el porcentaje de ocupación de la sala
getIngresos	- devolverá los ingresos producidos por la sala (entradas vendidas por precio)
getPelicula	- devolverá el título de la película que se está proyectando
getEntrada	- devolverá el precio de la entrada asignado al objeto

Estos métodos nos permitirán obtener información de un objeto del tipo *SalaCine*. Por ejemplo, supongamos que tenemos el objeto llamado *central* (correspondiente a la sala principal del multicine), para obtener la película que se está proyectando en dicha sala solo habría que usar este código:

```
String peli;           //una variable de cadena
peli = central.getPelicula();
```

O, por ejemplo, para saber los ingresos producidos por la sala central...

```
double ingresos;
ingresos = central.getIngresos();
```

Métodos para dar órdenes al objeto

Se programarán los siguientes métodos para dar órdenes a los objetos de la clase *SalaCine*.

vaciar

- Este método pondrá el número de plazas ocupadas a cero y le asignará a la película la cadena vacía.

entraUno

- Este método le dice al objeto que ha entrado una nueva persona en la sala. (Esto debe producir que el número de plazas ocupadas aumente en uno)

RESUMEN SALACINE

He aquí un resumen de la Clase de objetos *SalaCine*, la cual se programará en la próxima hoja:

Clase de objetos: ***SalaCine***

Propiedades de los objetos *SalaCine*:

Aforo	- número entero (int)
Ocupadas	- número entero (int)
Película	- cadena (String)
Entrada	- número decimal (double)

Valores por defecto de los objetos del tipo *SalaCine*:

Aforo: **100**
Ocupadas: **0**
Película: **(cadena vacía)**
Entrada: **5**

Métodos de los objetos del tipo *SalaCine*:

Métodos de asignación de propiedades (set)

setAforo	- modifica la propiedad Aforo
setOcupadas	- modifica la propiedad Ocupadas
setLibres	- modifica la propiedad Ocupadas también
setPelícula	- modifica la propiedad Película
setEntrada	- modifica la propiedad Entrada

Métodos de petición de información (get)

getAforo	- devuelve el valor de la propiedad Aforo
getOcupadas	- devuelve el valor de la propiedad Ocupadas
getLibres	- devuelve el número de butacas libres
getPorcentaje	- devuelve el porcentaje de ocupación de la sala
getIngresos	- devuelve los ingresos obtenidos por la venta de entradas
getPelícula	- devuelve el valor de la propiedad Película
getEntrada	- devuelve el valor de la propiedad Entrada

Métodos de orden

Vaciar	- vacía la ocupación de la sala y borra la película
entraUno	- le indica al objeto que ha entrado una persona más en la sala

EJERCICIO PRÁCTICO

Supongamos que programamos una Clase de objetos llamada *Rectangulo*, la cual permitirá construir objetos que representen a rectángulos.

- ¿Cómo declararía y construiría un objeto llamado *suelo* del tipo *Rectangulo*?
- Supongamos que las propiedades de los objetos de la clase *Rectangulo* sean las siguientes:
 - o Base (double)
 - o Altura (double)

¿Qué métodos de tipo *set* programaría para cambiar las propiedades de los objetos del tipo *Rectangulo*?

- Como ejemplo de los métodos anteriores, suponga que quiere asignar al objeto *suelo* una base de 30 y una altura de 50. ¿Qué código usaría?
- Teniendo en cuenta que nos puede interesar conocer el área de un objeto *Rectangulo* y su perímetro, y teniendo en cuenta que los objetos *Rectangulo* tienen dos propiedades (Base y Altura), ¿qué cuatro métodos *get* serían interesantes de programar para los objetos del tipo *Rectangulo*?
- Teniendo en cuenta los métodos del punto 4. Supongamos que quiero almacenar en una variable *double* llamada *area* el área del objeto *suelo*. ¿Qué código usaría? Y si quiero almacenar el perímetro del objeto *suelo* en una variable llamada *peri*?

CONCLUSIÓN

Para crear un objeto propio, necesita tener claro lo siguiente:

Nombre de la Clase del objeto.

Propiedades que tendrán los objetos de dicha clase.

Valores iniciales que tendrán las propiedades cuando se construya cada objeto.

Métodos que serán interesantes de programar:

**Métodos de ajuste de propiedades (set)
Métodos de petición de información (get)
Métodos de orden**

Una vez que se tenga claro lo anterior, se podrá empezar la programación de la Clase de objetos.

EJERCICIO GUIADO. JAVA: POO. PROGRAMACIÓN DE UNA CLASE

Programación de una Clase

En este ejercicio guiado, crearemos la Clase *SalaCine*, que hemos descrito en la hoja anterior. Luego, a partir de esta clase, fabricaremos objetos representando salas de cine, y los usaremos en un proyecto Java.

Recuerda las características que hemos decidido para la Clase *SalaCine* en la hoja anterior:

CLASE SALACINE

Nombre de la Clase: *SalaCine*

Propiedades de los objetos *SalaCine*:

Aforo	- número entero (int)
Ocupadas	- número entero (int)
Película	- cadena (String)
Entrada	- número decimal (double)

Valores por defecto de los objetos del tipo *SalaCine*:

Aforo: **100**
Ocupadas: **0**
Película: **(cadena vacía)**
Entrada: **5**

Métodos de los objetos del tipo *SalaCine*:

Métodos de asignación de propiedades (set)

setAforo	- modifica la propiedad Aforo
setOcupadas	- modifica la propiedad Ocupadas
setLibres	- modifica la propiedad Ocupadas también
setPelícula	- modifica la propiedad Película
setEntrada	- modifica la propiedad Entrada

Métodos de petición de información (get)

getAforo	- devuelve el valor de la propiedad Aforo
getOcupadas	- devuelve el valor de la propiedad Ocupadas
getLibres	- devuelve el número de butacas libres
getPorcentaje	- devuelve el porcentaje de ocupación de la sala
getIngresos	- devuelve los ingresos obtenidos por la venta de entradas
getPelícula	- devuelve el valor de la propiedad Película
getEntrada	- devuelve el valor de la propiedad Entrada

Métodos de orden

Vaciar	- vacía la ocupación de la sala y borra la película
entraUno	- le indica al objeto que ha entrado una persona más en la sala

PROGRAMACIÓN DE UNA CLASE

Fichero de la Clase

La programación de una clase de objetos se realiza en un fichero aparte, cuyo nombre es exactamente el mismo que el de la propia clase, y cuya extensión es .java.

Por ejemplo, si queremos programar la clase SalaCine, esto se debe hacer en un fichero llamado:

SalaCine.java

Cuando programemos esta clase dentro de NetBeans, veremos las facilidades que nos proporciona este para la creación de la clase. De momento, solo veremos de forma teórica como hay que programar la clase. (No tiene que introducir lo que viene a continuación en ningún sitio)

Estructura básica de la Clase

Dentro del fichero de la clase, comenzará la programación de esta de la siguiente forma:

```
public class SalaCine {  
  
  
}
```

La programación de una clase comienza siempre con una línea de código como la que sigue:

```
public class NombreDeLaClase {  
  
  
}
```

Toda la programación de la clase se introducirá dentro de las dos llaves.

Propiedades de la Clase

Lo primero que se debe introducir en la clase que se está programando son las propiedades. Las propiedades de una clase son básicamente variables globales de ésta. Si introducimos las propiedades de la clase *SalaCine*, esta nos quedaría así:

```
public class SalaCine {  
  
    int Aforo;  
    int Ocupadas;  
    String Película;  
    double Entrada;  
  
}
```

Constructor de la Clase

Cuando se planteó la clase *SalaCine*, se tuvo que decidir qué valores iniciales deberían tener las propiedades de la clase. Para asignar estos valores iniciales, es necesario programar lo que se denomina el *Constructor*.

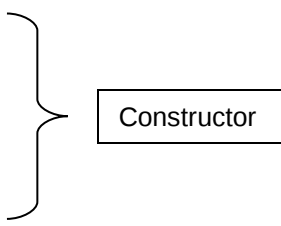
El *Constructor* de una clase es un método (un procedimiento para entendernos) un poco especial, ya que debe tener el mismo nombre de la clase y no devuelve nada, pero no lleva la palabra *void*. Dentro del constructor se inicializan las propiedades de la clase.

En general, la programación del constructor sigue la siguiente sintaxis:

```
public NombreDeLaClase() {  
    propiedad1 = valor;  
    propiedad2 = valor;  
    etc...  
}
```

La clase *SalaCine*, añadiendo el *Constructor*, tendrá el siguiente aspecto:

```
public class SalaCine {  
  
    int Aforo;  
    int Ocupadas;  
    String Película;  
    double Entrada;  
  
    //Constructor  
    public SalaCine() {  
        Aforo = 100;  
        Ocupadas = 0;  
        Película = "";  
        Entrada = 5.0;  
    }  
}
```



Observa como usamos el constructor de la clase *SalaCine* para asignar a cada propiedad los valores por defecto decididos en el diseño de la clase que se hizo en la hoja anterior.

Métodos del tipo set

Todas las clases suelen contener métodos del tipo *set*. Recuerda que estos métodos permiten asignar valores a las propiedades de la clase.

Debes tener en cuenta también que cuando se habla de método de una clase, en realidad se está hablando de un procedimiento o función, que puede recibir como parámetro determinadas variables y que puede devolver valores.

Los métodos del tipo *set* son básicamente procedimientos que reciben valores como parámetros que introducimos en las propiedades. Estos métodos no devuelven nada, así que son *void*.

Se recomienda, que el parámetro del procedimiento se llame de forma distinta a la propiedad que se asigna.

Veamos la programación del método *setAforo*, de la clase *SalaCine*:

```
public void setAforo(int afo) {  
  
    Aforo = afo;  
  
}
```

Observa este método:

1. Es void, es decir, no devuelve nada (*el significado de la palabra public se verá más adelante*)
2. El método recibe como parámetro una variable del mismo tipo que la propiedad que queremos modificar (en este caso *int*) y un nombre que se recomienda que no sea igual al de la propiedad (en nuestro caso, *afo*, de aforo)
3. Puedes observar que lo que se hace simplemente en el método es asignar la variable pasada como parámetro a la propiedad.

La mayoría de los procedimientos *set* usados para introducir valores en las propiedades tienen la misma forma. Aquí tienes la programación de los demás procedimientos *set* de la clase *SalaCine*.

```
//Método setOcupadas
public void setOcupadas(int ocu) {
    Ocupadas = ocu;
}

//Método setPelícula
public void setPelícula(String peli) {
    Película = peli;
}

//Método setEntrada
public void setEntrada(double entra) {
    Entrada = entra;
}
```

Hay un método *set* de la clase *SalaCine* llamado *setLibres* cuya misión es asignar el número de localidades libres del cine. Sin embargo la clase *SalaCine* no tiene una propiedad “Libres”. En realidad, este método debe modificar el número de localidades ocupadas. Observa su programación:

```
//Método setLibres
public void setLibres(int lib) {
    int ocu;

    ocu = Aforo - lib;
    Ocupadas = ocu;
}
```

Al asignar un número de localidades ocupadas, estamos asignando indirectamente el número de localidades libres. Como puedes observar en el método, lo que se hace es calcular el número de localidades ocupadas a partir de las libres, y asignar este valor a la propiedad *Ocupadas*.

No se pensó en crear una propiedad de la clase llamada *Libres* ya que en todo momento se puede saber cuantas localidades libres hay restando el Aforo menos las localidades *Ocupadas*.

La clase *SalaCine*, añadiendo los métodos *set*, quedaría de la siguiente forma:

```

public class SalaCine {

    int Aforo;
    int Ocupadas;
    String Película;
    double Entrada;

    //Constructor
    public SalaCine() {
        Aforo = 100;
        Ocupadas = 0;
        Pelicula = "";
        Entrada = 5.0;
    }

    //Métodos set

    //Método setAforo
    public void setAforo(int afo) {
        Aforo = afo;
    }

    //Método setOcupadas
    public void setOcupadas(int ocu) {
        Ocupadas = ocu;
    }

    //Método setPelícula
    public void setPelícula(String peli) {
        Pelicula = peli;
    }

    //Método setEntrada
    public void setEntrada(double entra) {
        Entrada = entra;
    }

    //Método setLibres
    public void setLibres(int lib) {
        int ocu;

        ocu = Aforo - lib;
        Ocupadas = ocu;
    }

}

```

Métodos Set

Métodos del tipo get

Al igual que los métodos set, los métodos *get* son muy fáciles de programar ya que suelen tener siempre la misma forma.

Estos métodos no suelen llevar parámetros y devuelven el valor de la propiedad correspondiente usando la típica instrucción *return* usada tanto en las funciones. Por tanto, un método get nunca es void. Siempre será del mismo tipo de datos que la propiedad que devuelve.

Veamos la programación del método *getAforo*:

```
//Método getAforo
public int getAforo() {
    return Aforo;
}
```

Como puedes ver el método simplemente devuelve el valor de la propiedad Aforo. Como esta propiedad es int, el método es int.

Los métodos que devuelven el resto de las propiedades son igual de sencillos de programar:

```
//Método getOcupadas
public int getOcupadas() {
    return Ocupadas;
}

//Método getPelicula
public String getPelicula() {
    return Película;
}

//Método getEntrada
public double getEntrada() {
    return Entrada;
}
```

Todos estos métodos son iguales. Solo tienes que fijarte en el tipo de datos de la propiedad que devuelven.

Existen otros métodos get que devuelven cálculos realizados con las propiedades. Estos métodos realizan algún cálculo y luego devuelven el resultado. Observa el siguiente método *get*:

```
//Método getLibres
public int getLibres() {
    int lib;
    lib = Aforo - Ocupadas;
    return lib;
}
```

No existe una propiedad *Libres*, por lo que este valor debe ser calculado a partir del Aforo y el número de localidades Ocupadas. Para ello restamos y almacenamos el valor en una variable a la que hemos llamado *lib*. Luego devolvemos dicha variable.

Los dos métodos *get* que quedan por programar de la clase *SalaCine* son parecidos:

```
//Método getPorcentaje
public double getPorcentaje() {
    double por;
    por = (double) Ocupadas / (double) Aforo * 100.0;
    return por;
}
```

Este método calcula el porcentaje de ocupación de la sala (es un valor double)

```
//Método getIngresos
public double getIngresos() {
    double ingre;
```

```

        ingre = Ocupadas * Entrada;
        return ingre;
    }

```

Los ingresos se calculan multiplicando el número de entradas por lo que vale una entrada.

La clase *SalaCine* una vez introducidos los métodos get quedaría de la siguiente forma:

```

public class SalaCine {

    int Aforo;
    int Ocupadas;
    String Película;
    double Entrada;

    //Constructor
    public SalaCine() {
        Aforo = 100;
        Ocupadas = 0;
        Pelicula = "";
        Entrada = 5.0;
    }

    //Métodos set

    //Método setAforo
    public void setAforo(int afo) {
        Aforo = afo;
    }

    //Método setOcupadas
    public void setOcupadas(int ocu) {
        Ocupadas = ocu;
    }

    //Método setPelícula
    public void setPelícula(String peli) {
        Pelicula = peli;
    }

    //Método setEntrada
    public void setEntrada(double entra) {
        Entrada = entra;
    }

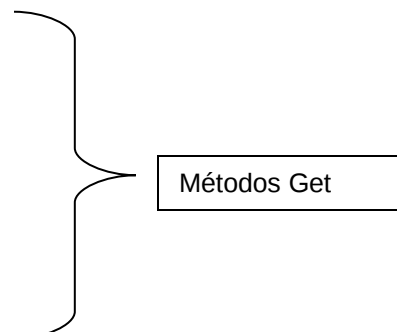
    //Método setLibres
    public void setLibres(int lib) {
        int ocu;

        ocu = Aforo - lib;
        Ocupadas = ocu;
    }

    //Métodos get

    //Método getAforo
    public int getAforo() {
        return Aforo;
    }
}

```



```

    }

    //Método getOcupadas
    public int getOcupadas() {
        return Ocupadas;
    }

    //Método getPelicula
    public String getPelicula() {
        return Película;
    }

    //Método getEntrada
    public double getEntrada() {
        return Entrada;
    }

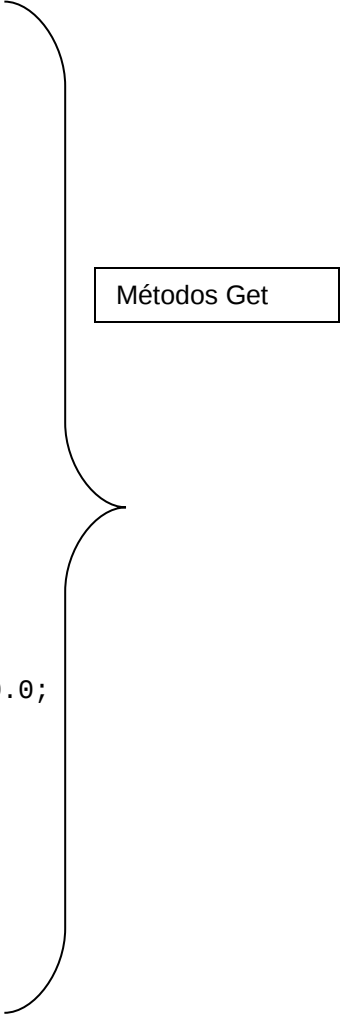
    //Método getLibres
    public int getLibres() {
        int lib;
        lib = Aforo - Ocupadas;
        return lib;
    }

    //Método getPorcentaje
    public double getPorcentaje() {
        double por;
        por = (double) Ocupadas / (double) Aforo * 100.0;
        return por;
    }

    //Método getIngresos
    public double getIngresos() {
        double ingre;
        ingre = Ocupadas * Entrada;
        return ingre;
    }

}

```



Métodos Get

Métodos de orden

Para finalizar la programación de la clase *SalaCine*, se programarán los dos métodos de orden que hemos indicado en el planteamiento de la clase. Estos métodos suelen realizar alguna tarea que involucra a las propiedades de la clase, modificándola internamente. No suelen devolver ningún valor, aunque pueden recibir parámetros.

Veamos la programación del método *Vaciar*, cuyo objetivo es vaciar la sala y quitar la película en proyección:

```
//Método Vaciar
public void Vaciar() {
    Ocupadas = 0;
    Película = "";
}
```

Como se puede observar, es un método muy sencillo, ya que simplemente cambia algunas propiedades de la clase.

El método *entraUno* es también muy sencillo de programar. Este método le indica al objeto que ha entrado un nuevo espectador. Sabiendo esto, el objeto debe aumentar en uno el número de localidades ocupadas:

```
//Método entraUno
public void entraUno() {
    Ocupadas++;
}
```

Añadiendo estos dos últimos métodos, la programación de la clase *SalaCine* quedaría finalmente como sigue:

```
public class SalaCine {

    int Aforo;
    int Ocupadas;
    String Película;
    double Entrada;

    //Constructor
    public SalaCine() {
        Aforo = 100;
        Ocupadas = 0;
        Película = "";
        Entrada = 5.0;
    }

    //Métodos set

    //Método setAforo
    public void setAforo(int afo) {
        Aforo = afo;
    }

    //Método setOcupadas
    public void setOcupadas(int ocu) {
        Ocupadas = ocu;
    }

    //Método setPelícula
    public void setPelícula(String peli) {
        Película = peli;
    }

    //Método setEntrada
    public void setEntrada(double entra) {
        Entrada = entra;
    }
}
```

Propiedades (variables globales)

Constructor

Métodos Set

```
//Método setLibres
public void setLibres(int lib) {
    int ocu;

    ocu = Aforo - lib;
    Ocupadas = ocu;
}
```



```

//Métodos get

//Método getAforo
public int getAforo() {
    return Aforo;
}

//Método getOcupadas
public int getOcupadas() {
    return Ocupadas;
}

//Método getPelicula
public String getPelicula() {
    return Película;
}

//Método getEntrada
public double getEntrada() {
    return Entrada;
}

//Método getLibres
public int getLibres() {
    int lib;
    lib = Aforo - Ocupadas;
    return lib;
}

//Método getPorcentaje
public double getPorcentaje() {
    double por;
    por = (double) Ocupadas / (double) Aforo * 100.0;
    return por;
}

//Método getIngresos
public double getIngresos() {
    double ingre;
    ingre = Ocupadas * Entrada;
    return ingre;
}

//Métodos de orden

//Método Vaciar
public void Vaciar() {
    Ocupadas = 0;
    Película = "";
}

//Método entraUno
public void entraUno() {
    Ocupadas++;
}

```

Métodos Get

Métodos de orden y otros métodos.

}

EJERCICIOS RECOMENDADOS

Supongamos que tenemos una clase llamada *Rectangulo* que nos permitirá generar objetos de tipo rectángulo.

Sea el planteamiento de la clase *Rectangulo* el que sigue:

CLASE RECTANGULO

Nombre de la clase: Rectangulo

Propiedades de los objetos de la clase Rectangulo:

Base (double)
Altura (double)

Valores iniciales de las propiedades de los objetos de la clase Rectangulo:

Base – 100
Altura – 50

Métodos:

Métodos set:

setBase – permite asignar un valor a la propiedad Base.
setAltura – permite asignar un valor a la propiedad Altura.

Métodos get:

getBase – devuelve el valor de la propiedad Base
getAltura – devuelve el valor de la propiedad Altura
getArea – devuelve el área del rectángulo
getPerímetro – devuelve el perímetro del rectángulo

Otros métodos:

Cuadrar – este método debe hacer que la Altura tenga el valor de la Base.

SE PIDE:

Realiza (en papel) la programación de la clase Rectangulo a partir del planteamiento anterior.

CONCLUSIÓN

La programación de una clase se realiza en un fichero que tiene el mismo nombre que la clase y extensión .java

La estructura general de una clase es la siguiente:

```
public class NombreClase {  
  
    Propiedades (variables globales)  
  
    Constructor  
  
    Métodos set  
  
    Métodos get  
  
    Métodos de orden y otros métodos  
  
}
```

El Constructor es un procedimiento que no devuelve nada pero que no es void. El constructor debe llamarse igual que la clase y se usa para asignar los valores iniciales a las propiedades.

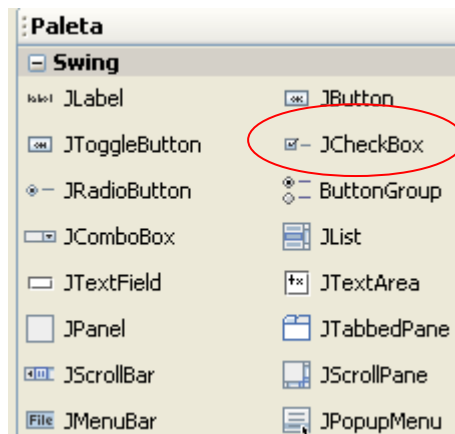
Los métodos set son void y reciben como parámetro un valor que se asigna a la propiedad correspondiente.

Los métodos get no tienen parámetros y devuelven el valor de una propiedad de la clase, aunque también pueden realizar cálculos y devolver sus resultados.

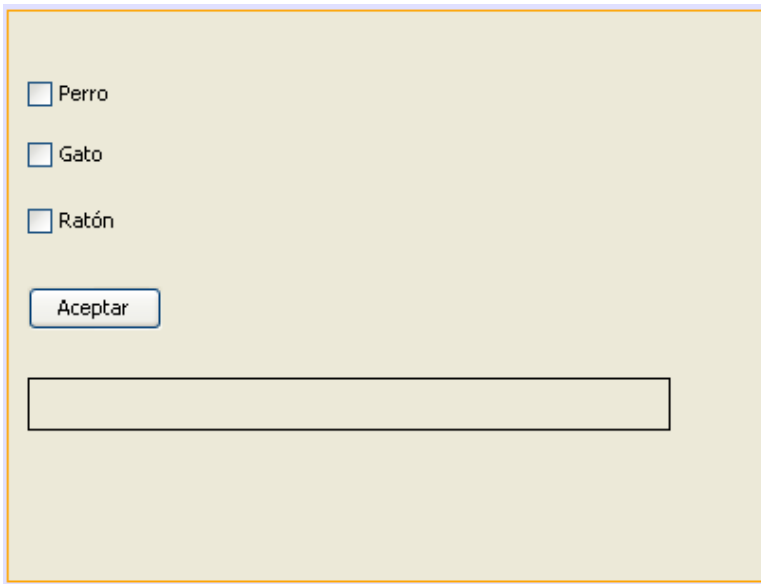
Los métodos de orden realizan alguna tarea específica y a veces modifican las propiedades de la clase de alguna forma.

EJERCICIO GUIADO. JAVA: CUADROS DE VERIFICACIÓN

4. Realiza un nuevo proyecto.
5. En la ventana principal debes añadir lo siguiente:
 - f. Un botón “Aceptar” llamado btnAceptar.
 - g. Una etiqueta con borde llamada etiResultado.
8. Añade también tres cuadros de verificación. Estos cuadros son objetos del tipo JCheckBox.



9. Añade tres JCheckBox y cambia el texto de ellos, de forma que aparezca “Perro”, “Gato” y “Ratón”.
10. Debe cambiar el nombre de cada uno de ellos. Se llamarán: chkPerro, chkGato, chkRaton.
11. La ventana tendrá el siguiente aspecto cuando termine:



12. El programa debe funcionar de la siguiente forma:

Cuando el usuario pulse aceptar, en la etiqueta aparecerá un mensaje indicando qué animales han sido “seleccionados”. Para ello hay que programar el evento *actionPerformed* del botón Aceptar. En ese evento añade el siguiente código:

```
String mensaje="Animales elegidos: ";
if (chkPerro.isSelected()) {
    mensaje=mensaje+"Perro ";
}

if (chkGato.isSelected()) {
    mensaje=mensaje+"Gato ";
}

if (chkRaton.isSelected()) {
    mensaje=mensaje+"Raton ";
}

etiResultado.setText(mensaje);
```

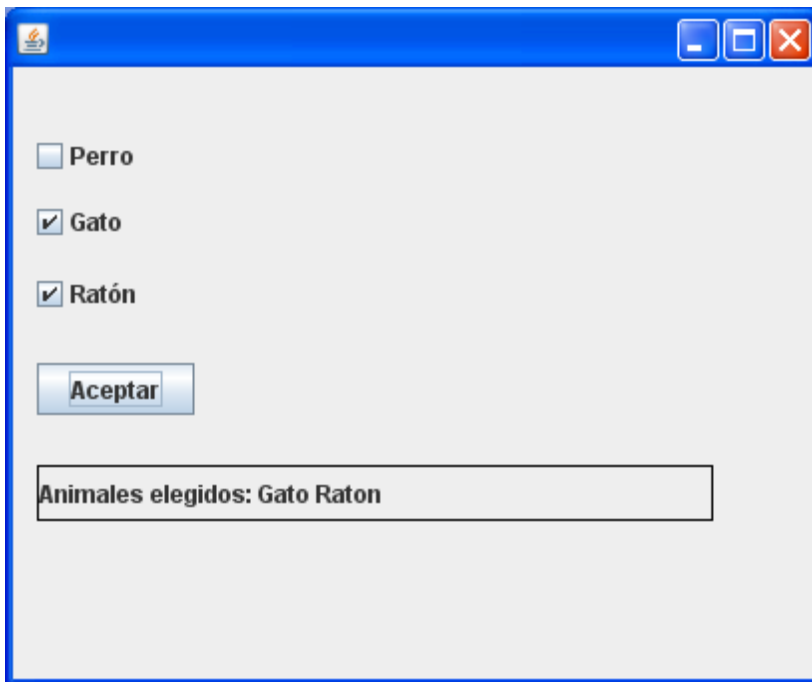
13. Observa el código. En él se hace lo siguiente:

- n. Se crea una variable de cadena llamada *mensaje*.
- o. En esa variable se introduce el texto “Animales elegidos: “
- p. Luego, compruebo si está seleccionada la casilla de verificación *chkPerro*. Si es así concateno a la cadena *mensaje* la palabra “Perro”.
- q. Luego compruebo si está seleccionada la casilla de verificación *chkGato* y hago lo mismo.

- r. Lo mismo con la casilla chkRaton.
- s. Finalmente presento la cadena mensaje en la etiqueta etiResultado.

20. Observa el método isSelected() propio de las casillas de verificación, permiten saber si una casilla está activada o no.

21. Ejecute el programa. Seleccione por ejemplo las casillas Gato y Ratón. Al pulsar Aceptar el resultado debe ser el siguiente:



CONCLUSIÓN

Los cuadros de verificación (JCheckBox) se usan cuando quieres seleccionar varias opciones.

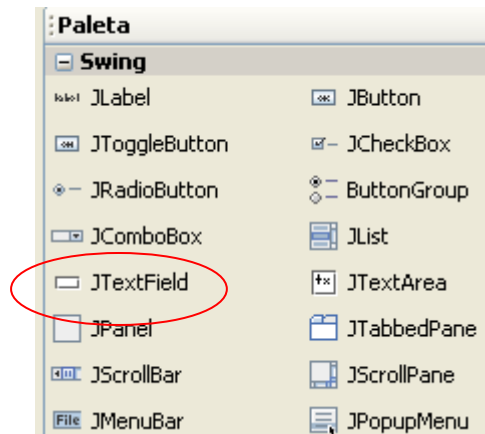
EJERCICIO GUIADO. JAVA: BOTONES DE OPCIÓN

22. Realiza un nuevo proyecto.

23. En la ventana principal debes añadir lo siguiente:

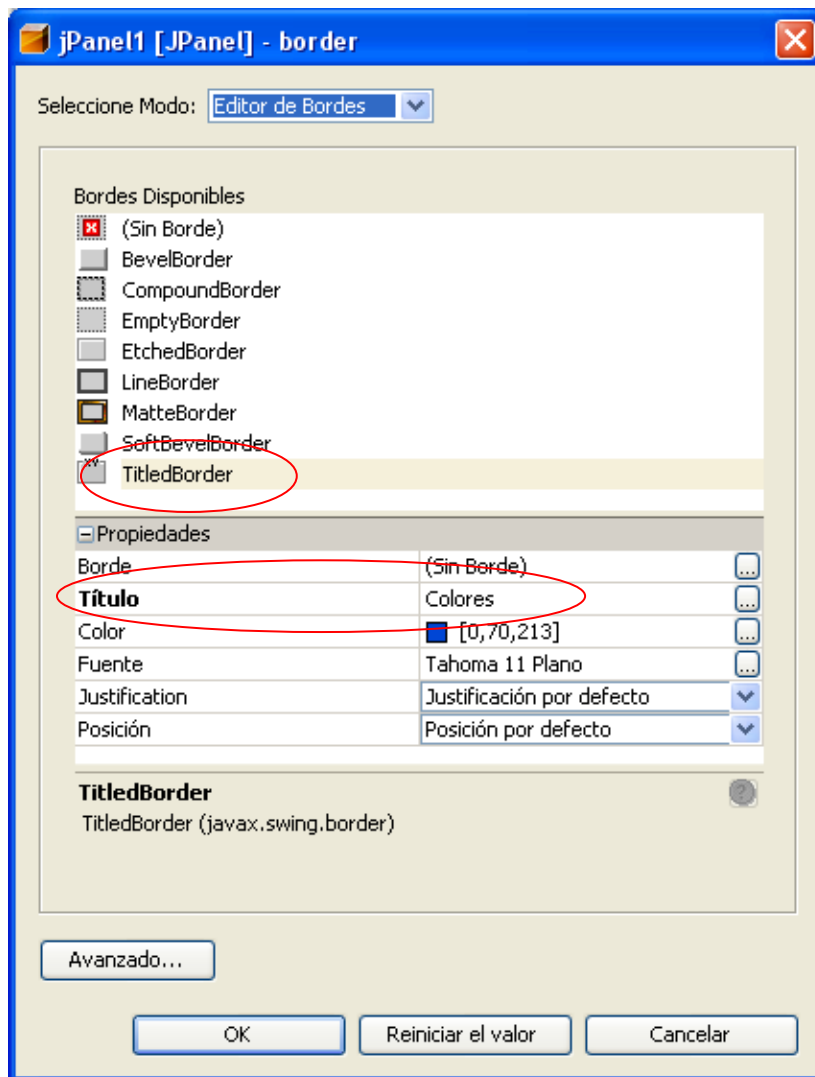
- x. Un botón “Aceptar” llamado btnAceptar.
- y. Una etiqueta con borde llamada etiResultado.

26. Añade un panel. Un panel es una zona rectangular que puede contener elementos (botones, etiquetas, etc) La forma de poner un panel es a través del objeto JPanel.

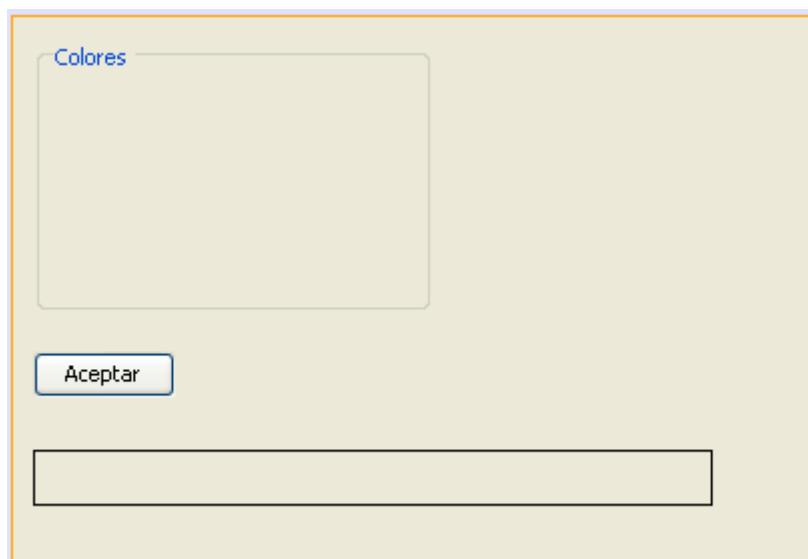


27. Una vez añadido el panel en el JFrame, le pondremos un borde para poder localizarlo fácilmente. Debes hacer lo siguiente:

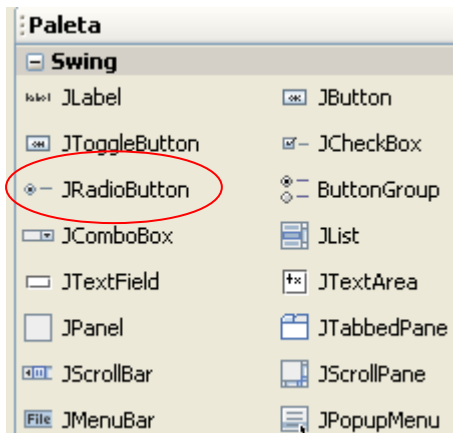
- bb. Selecciona el panel que has añadido.
- cc. Activa la propiedad Border (botón con tres puntos)
- dd. Busca el tipo de borde llamado TitledBorder (borde con título) y pon el título colores.



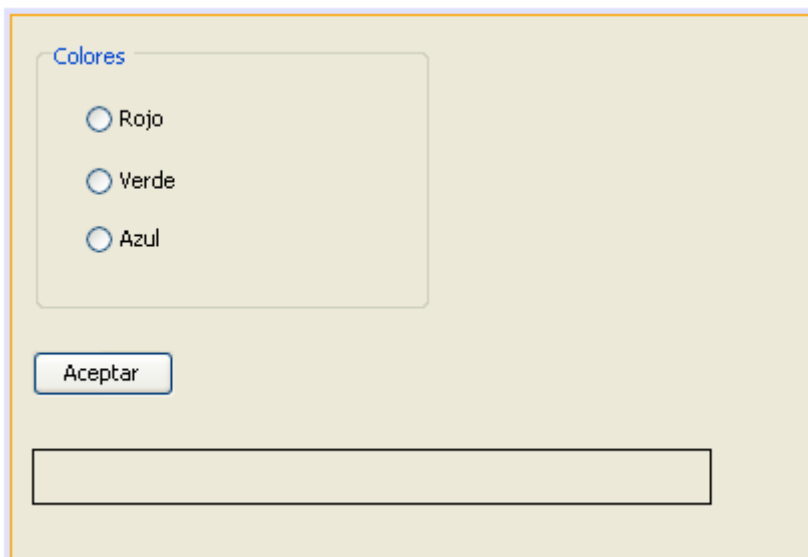
31. Tu ventana debe quedar más o menos así:



32. Ahora debes añadir tres botones de opción (botones de radio) dentro del panel. Estos botones son objetos del tipo `JRadioButton`.



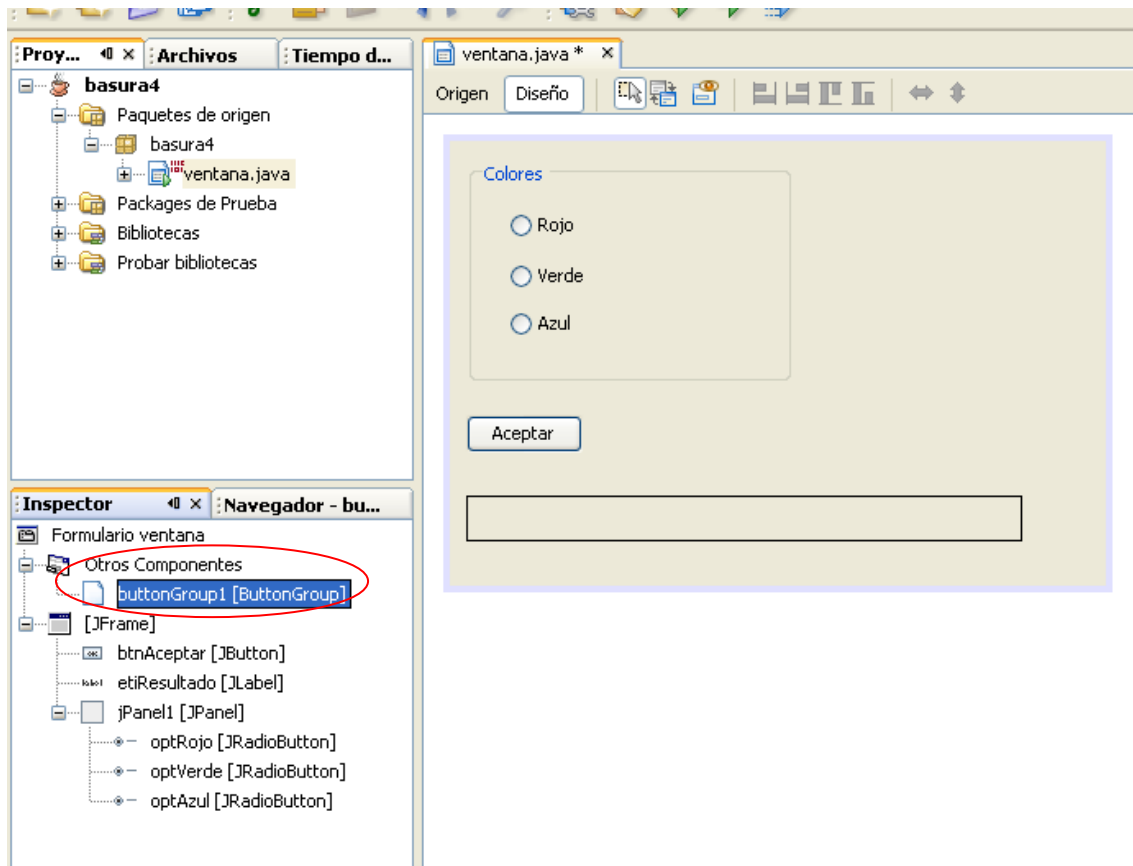
33. Añade tres `JRadioButton` y cambia el texto de ellos, de forma que aparezca “Rojo”, “Verde” y “Azul”.
34. Debe cambiar el nombre de cada uno de ellos. Se llamarán: `optRojo`, `optVerde`, `optAzul`.
35. La ventana tendrá el siguiente aspecto cuando termine:



36. Si ejecuta el programa, observará que pueden seleccionarse varios colores a la vez. Esto no es interesante, ya que los botones de opción se usan para activar solo una opción entre varias.

37. Hay que hacer que solo un botón de opción pueda estar seleccionado a la vez. Para ello, debe añadir un nuevo objeto. Realice los siguientes pasos:

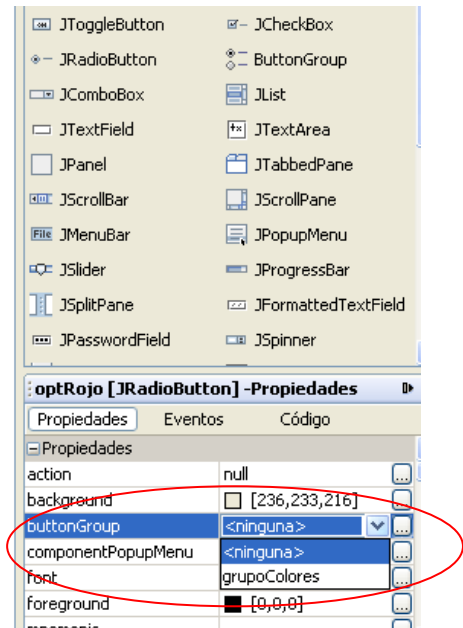
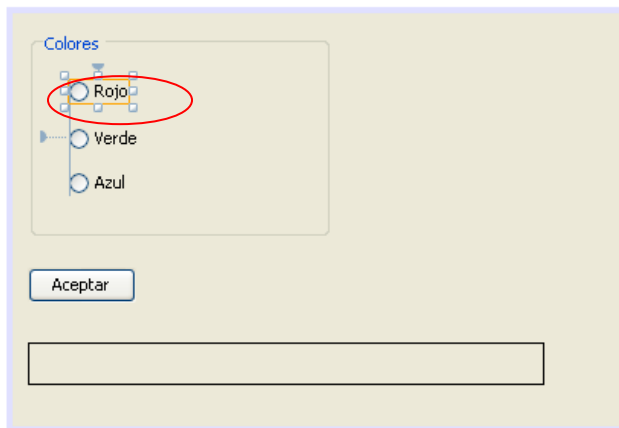
- II. Añada un objeto del tipo ButtonGroup al formulario. ¡Atención! Este objeto es invisible, y no se verá en el formulario, sin embargo, lo podréis ver en el Inspector, en la parte de “Otros Componentes”:



mm. Tienes que darle un nombre al ButtonGroup. El nombre será “grupoColores”.

nn. Ahora, hay que conseguir que los tres botones pertenezcan al mismo grupo. Es decir, que pertenezcan al grupo grupoColores.

oo. Selecciona el botón de opción optRojo y cambia su propiedad buttonGroup, indicando que pertenece al grupo colores (observa la imagen):



pp. Haz lo mismo con los botones optVerde y optAzul.

43. Acabas de asociar los tres botones de opción a un mismo grupo. Esto produce que solo una de las tres opciones pueda estar activada. Pruébalo ejecutando el programa.

44. Ahora interesa que la opción "Rojo" salga activada desde el principio. Una forma de hacer esto es programando en el "Constructor" lo siguiente:

```
optRojo.setSelected(true);
```

El método `setSelected` hace que se pueda activar o desactivar un botón de opción.

Prueba el programa. Observa como la opción Rojo está activada inicialmente.

45. El programa no está terminado aún. Interesa que cuando el usuario pulse el botón Aceptar, en la etiqueta aparezca el color elegido. Para ello, en el `actionPerformed` del botón Aceptar programe lo siguiente:

```
String mensaje="Color elegido: ";

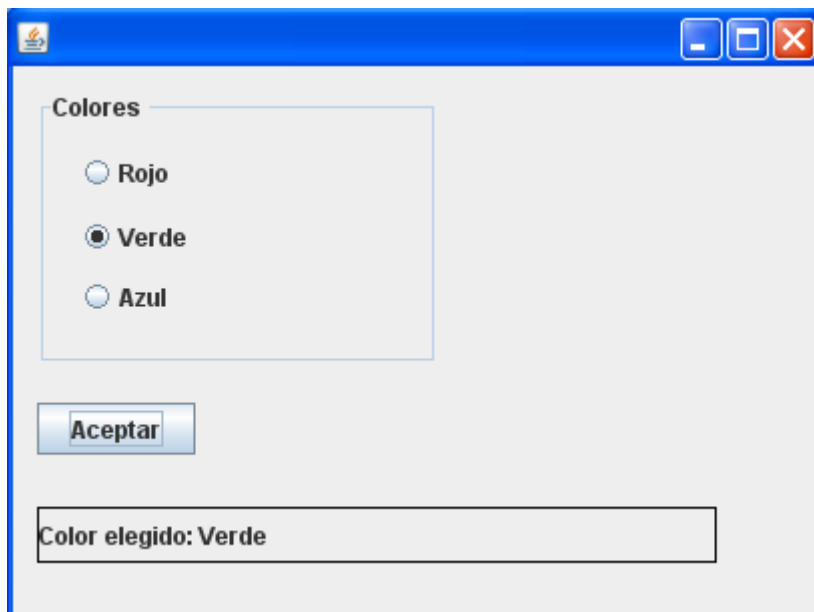
if (optRojo.isSelected()) {
    mensaje=mensaje+"Rojo";
} else if (optVerde.isSelected()) {
    mensaje=mensaje+"Verde";
} else if (optAzul.isSelected()) {
    mensaje=mensaje+"Azul";
}
```

```
etiResultado.setText(mensaje);
```

46. Observa el código. En él se hace lo siguiente:

- uu. Se crea una variable de cadena llamada *mensaje*.
- vv. En esa variable se introduce el texto “Color elegido: “
- ww. Luego se comprueba que opción está seleccionada, usando el método `isSelected` de los botones de opción. Este método te dice si un botón está seleccionado o no.
- xx. Según la opción que esté seleccionada, se añade un texto u otro a la cadena *mensaje*.
- yy. Finalmente se muestra la cadena *mensaje* en la etiqueta `etiResultado`.

52. Ejecute el programa. Seleccione por ejemplo la Verde. Al pulsar Aceptar el resultado debe ser el siguiente:



CONCLUSIÓN

Los botones de opción, también llamados botones de radio (`JRadioButton`) se usan cuando quieres que el usuario pueda elegir una opción de entre varias.

Es interesante que los botones de radio aparezcan dentro de un panel `JPanel`. Se recomienda colocar un borde al panel.

Es totalmente necesario añadir un objeto del tipo `ButtonGroup`, y hacer que los botones de radio pertenezcan a dicho grupo. En caso contrario, será posible activar varios botones de opción a la vez.

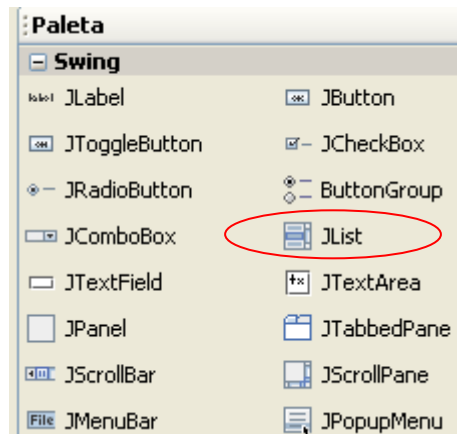
EJERCICIO GUIADO. JAVA: CUADROS DE LISTA

53. Realiza un nuevo proyecto.

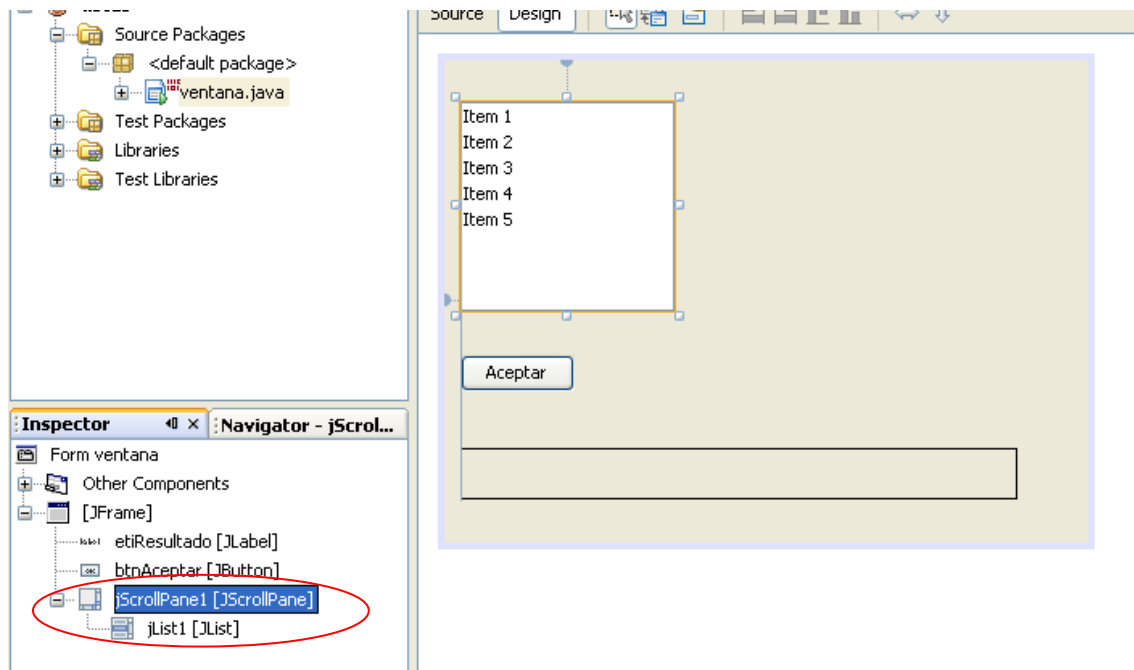
54. En la ventana principal debes añadir lo siguiente:

- ccc. Un botón “Aceptar” llamado btnAceptar.
- ddd. Una etiqueta con borde llamada etiResultado.

57. Añade un cuadro de lista. Los cuadros de listas son objetos JList.



58. Cámbiale el nombre al JList. Ten cuidado, ya que en los JList aparecen siempre dentro de otro objeto llamado JScrollPane. Si miras en el Inspector, verás que al pulsar en el botón + del JScrollPane aparecerá tu JList:

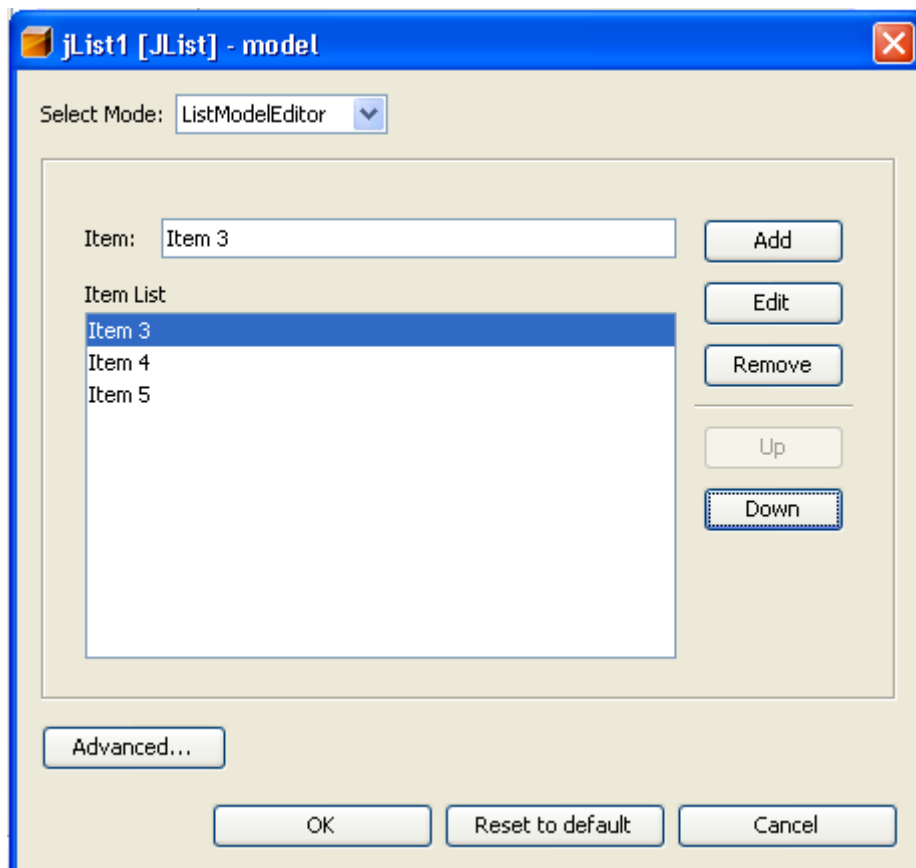


59. Aprovecha para cambiarle el nombre al JList. El nuevo nombre será IstColores.

60. Si te fijas en el JList, consiste en un cuadro que contiene una serie de Items. Estos elementos pueden ser cambiados a través de la propiedad Model del JList.

61. Busca la propiedad Model y haz clic en el botón de los tres puntos. Aparecerá un cuadro de diálogo parecido al siguiente. Solo tienes que seleccionar los elementos que quieras y pulsar el botón “Borrar” (Remove) para eliminarlos de la lista.

62. Puedes añadir elementos escribiéndolos en el cuadro Artículo y luego pulsando el botón “Añadir” (Add).



63. Debes hacer que la lista sea la siguiente:

Rojo
Verde
Azul

64. Ahora programaremos el *actionPerformed* del botón Aceptar. Debes introducir el siguiente código:

```
String mensaje;  
  
mensaje="El color seleccionado es: "+lstColores.getSelectedValue().toString();  
etiResultado.setText(mensaje);
```

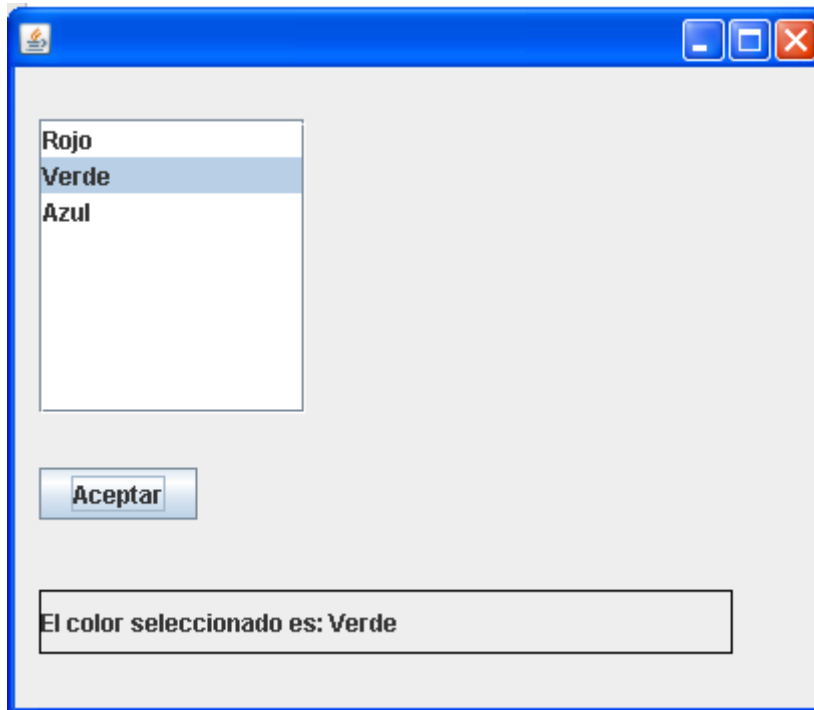
65. Observa el código:

- nnn. Se crea una variable de cadena llamada *mensaje*.
- ooo. Y dentro de esta variable se introduce una concatenación de cadenas.
- ppp. Observa la parte: `lstColores.getSelectedValue()`, esta parte devuelve el valor seleccionado de la lista.
- qqq. Hay que tener en cuenta que este valor no es una cadena, por eso hay que convertirla a cadena añadiendo `.toString()`.

rrr. De esta manera puedes extraer el elemento seleccionado de un cuadro de lista.

sss. Luego simplemente ponemos la cadena mensaje dentro de la etiqueta.

72. Ejecuta el programa y observa su funcionamiento. Por ejemplo, si seleccionas el color verde y pulsas aceptar el resultado será el siguiente:



73. Vamos a mejorar el programa. Puede suceder que el usuario no seleccione ningún valor del cuadro de lista, y sería interesante en este caso que el programa avisara de ello. Cambie el código del botón Aceptar por este otro código:

```
String mensaje;  
  
if (lstColores.getSelectedIndex()==-1) {  
    mensaje="No hay un color seleccionado.";  
} else {  
    mensaje="El color seleccionado es: "+lstColores.getSelectedValue().toString();  
}  
etiResultado.setText(mensaje);
```

74. Observa el código:

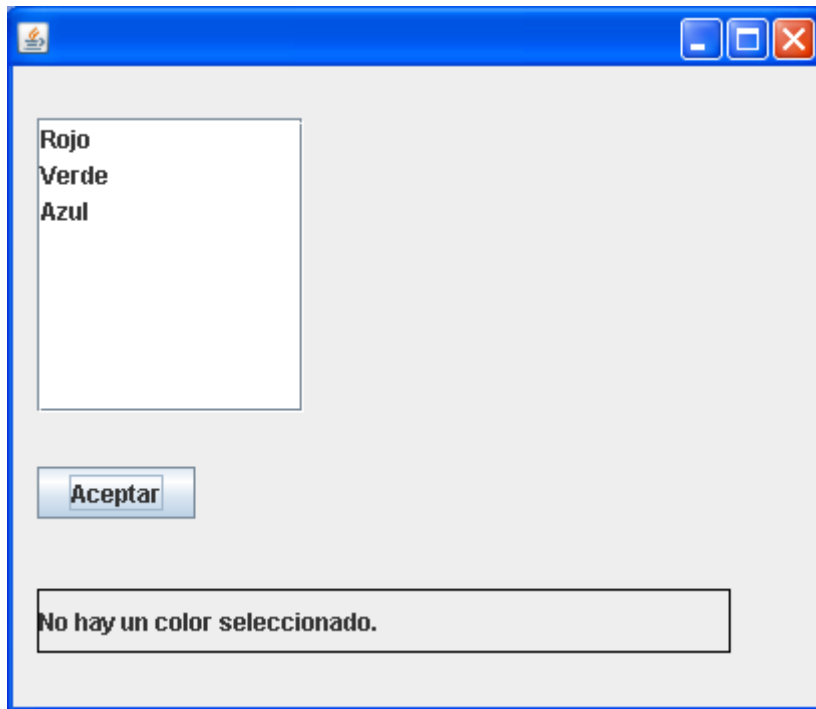
www. El método `getSelectedIndex` me dice el índice del elemento que está seleccionado.

xxx. Por ejemplo, si está seleccionado el primero el índice es 0, si está seleccionado el segundo el índice es 1, etc.

yyy. Si este método devuelve -1, entonces es señal de que no hay ningún elemento seleccionado.

zzz. Aprovecho esto para mostrar un mensaje indicando lo sucedido.

79. Si ejecuta el programa y pulsa el botón Aceptar sin seleccionar nada el resultado debería ser el siguiente:



80. Se podría haber prescindido del botón aceptar si el código anterior se hubiera puesto en el evento `MouseClicked` del cuadro de lista en vez de en el `ActionPerformed` del botón Aceptar. En este caso, cada vez que se seleccionara un elemento de la lista, automáticamente aparecería el mensaje en la etiqueta.

Se anima a que realice esta modificación.

CONCLUSIÓN

El objeto `JList` permite crear cuadros de lista. Estos objetos contienen una serie de elementos que pueden ser seleccionados.

A través del método `getSelectedValue` se puede obtener el elemento que está seleccionado. (Recuerda convertirlo a cadena con `toString`)

A través del método `getSelectedIndex` se puede saber la posición del elemento seleccionado. Si este índice es `-1`, entonces sabremos que no hay ningún elemento seleccionado.

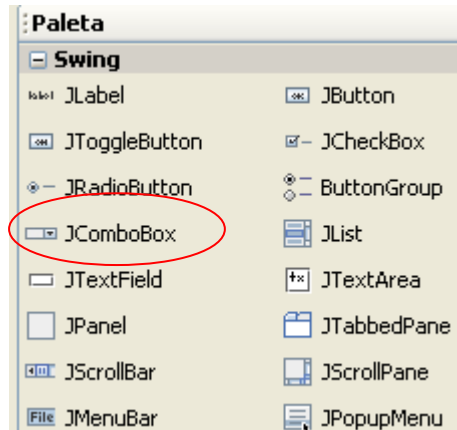
EJERCICIO GUIADO. JAVA: CUADROS COMBINADOS

81. Realiza un nuevo proyecto.

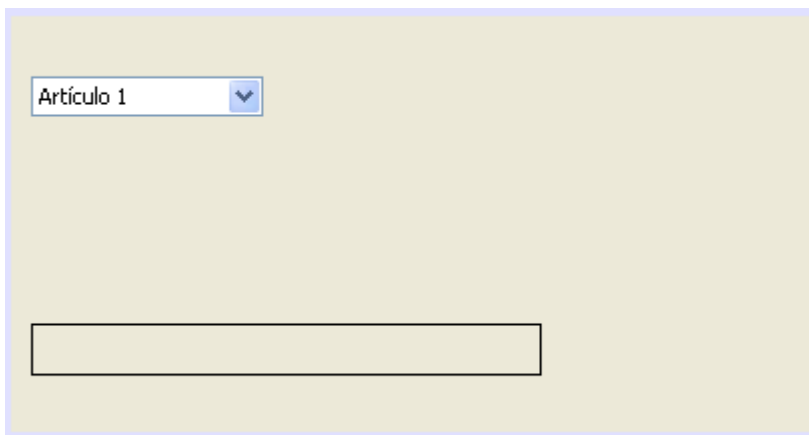
82. En la ventana principal debes añadir lo siguiente:

eeee. Una etiqueta con borde llamada etiResultado.

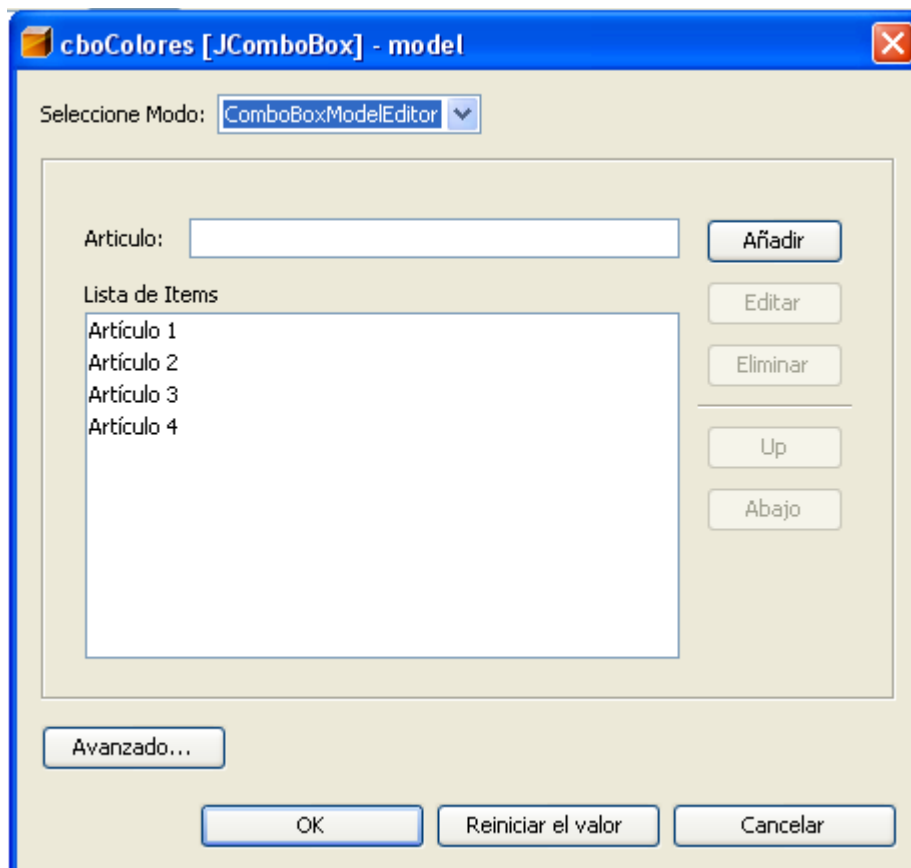
84. Añade un cuadro combinado (combo). Los cuadros combinados son objetos del tipo JComboBox. Básicamente, un combo es una lista desplegable.



85. Cámbiale el nombre al JComboBox. El nombre será cboColores. Tu programa debe tener más o menos este aspecto.



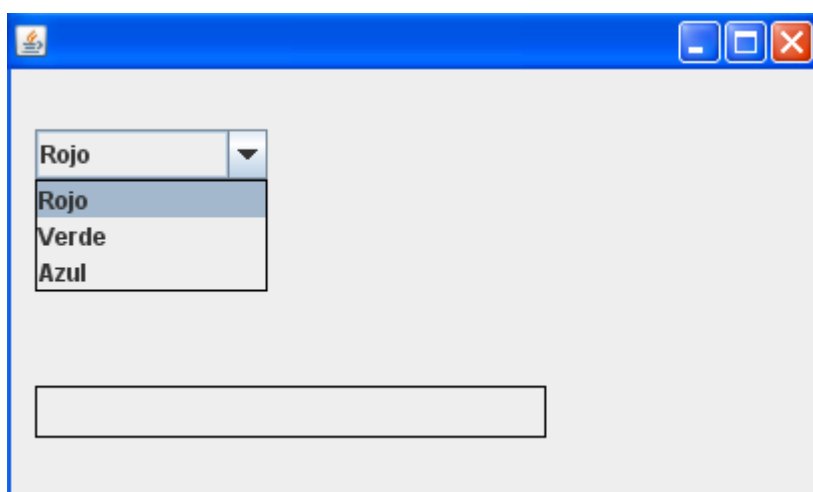
86. Los elementos del cboColores pueden ser cambiados a través de la propiedad Model. Selecciona el combo y activa la propiedad Model (el botoncito con los tres puntos) Aparecerá lo siguiente:



87. Al igual que pasaba con los cuadros de lista, se pueden eliminar los elementos que contiene el combo y añadir elementos propios. Use los botones Añadir y Eliminar para añadir la siguiente lista de elementos:

Rojo
Verde
Azul

88. Ejecuta el programa y observa el funcionamiento del desplegable...



89. Vamos a hacer que cuando se elija un elemento del desplegable, en la etiqueta aparezca un mensaje indicando el color elegido.

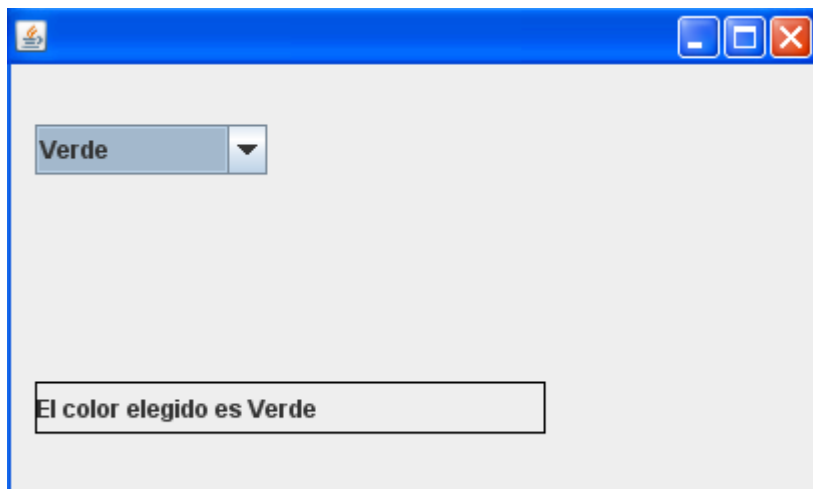
Para ello, debes programar el evento *actionPerformed* del combo y añadir el siguiente código:

```
String mensaje="El color elegido es ";  
  
mensaje=mensaje+cboColores.getSelectedItem().toString();  
etiResultado.setText(mensaje);
```

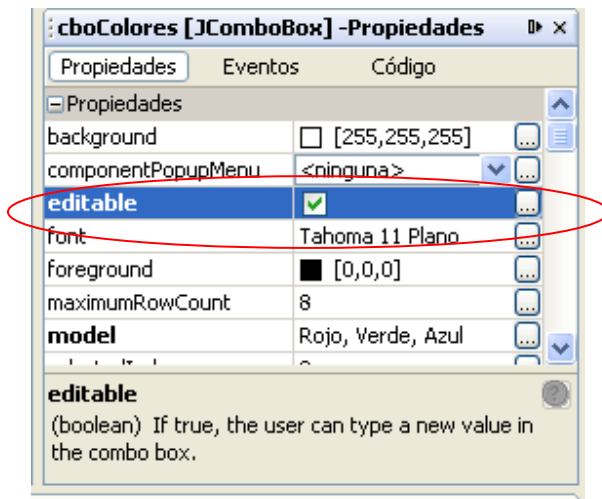
90. Este código hace lo siguiente:

mmmm. Crea una variable de cadena.
nnnn. Concatena dentro de ella el mensaje "El color elegido es" con el color seleccionado.
oooo. Observa el método *getSelectedItem*, se usa para saber el elemento seleccionado del combo. Es necesario convertirlo a texto con *toString*.
pppp. Finalmente se coloca el mensaje en la etiqueta.

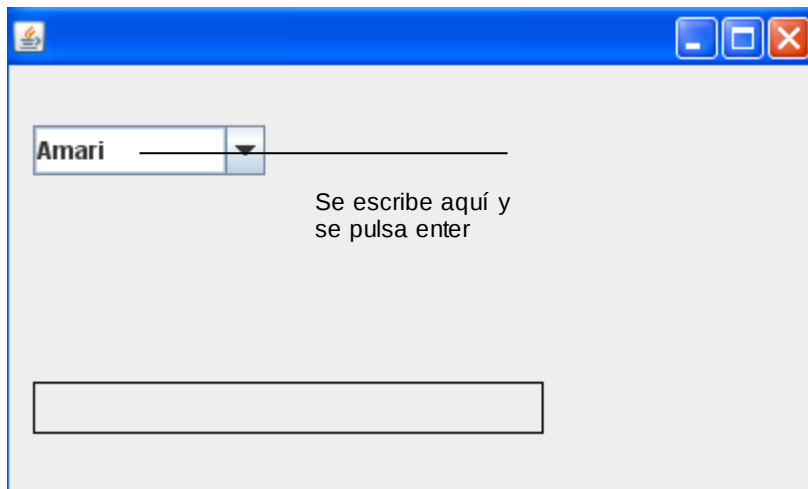
95. Ejecuta el programa y comprueba su funcionamiento. Por ejemplo, si elegimos el color verde, el aspecto del programa será el siguiente:



96. Los cuadros combinados pueden funcionar también como cuadros de texto. Es decir, pueden permitir que se escriba texto dentro de ellos. Para hacer esto, basta con cambiar su propiedad "editable" y activarla.



97. Ejecuta el programa y observa como se puede escribir dentro del combo. Al pulsar Enter, el programa funciona igualmente con el texto escrito.



CONCLUSIÓN

Los combos son listas desplegables donde se puede elegir una de las opciones propuestas.

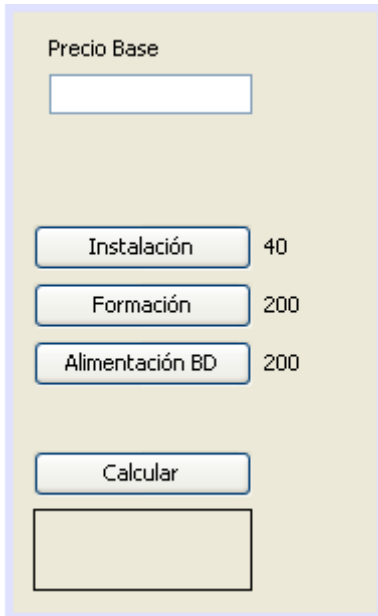
Los combos pueden funcionar también como cuadros de textos, si se activa la opción editable.

A través del método `getSelectedItem` se puede extraer la opción seleccionada o el texto escrito en el combo.

EJERCICIO GUIADO. JAVA: TOGGLEBUTTONS

98. Realiza un nuevo proyecto.

99. Crearás una ventana como la que sigue teniendo en cuenta lo siguiente:



vvvv. Se añadirá una etiqueta con el texto "Precio Base". No hace falta cambiarle su nombre.

www. Se añadirá un cuadro de texto llamado txtPrecioBase.

xxxx. Se creará un botón "Calcular", llamado btnCalcular.

yyyy. Se creará una etiqueta vacía y con borde llamada etiTotal. Use la propiedad *font* de esta etiqueta para hacer que el texto tenga un mayor tamaño.

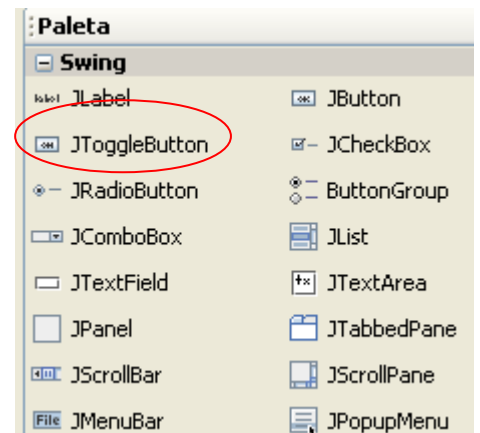
zzzz. Debes añadir también tres botones, con el texto "Instalación", "Formación" y "Alimentación BD" respectivamente.

Estos botones no son botones normales, son botones del tipo `JToggleButton`. Usa este tipo de objeto para crearlos.

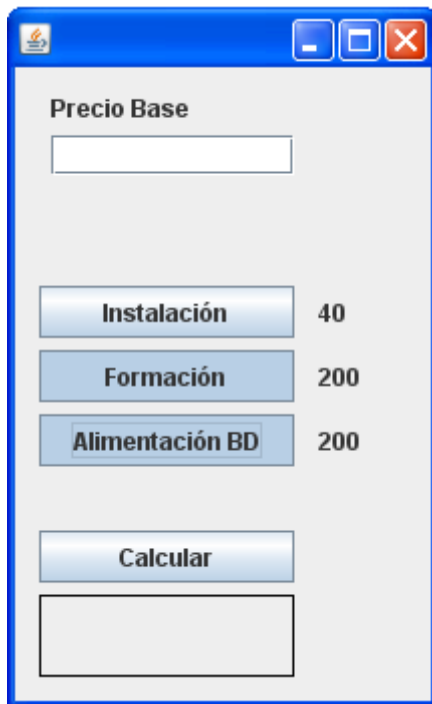
Estos botones, se diferencian de los botones normales en que se quedan pulsados cuando se hace un clic sobre ellos, y no vuelven a su estado normal hasta que no se vuelve a hacer clic sobre ellos.

Los tres botones se llamarán respectivamente: `tbtnInstalacion`, `tbtnFormacion`, `tbtnAlimentacionBD`.

aaaaa. Añade finalmente tres etiquetas conteniendo los números 40, 200, 200. La primera se llamará `etiPrecioInstalacion`, la segunda `etiPrecioFormacion` y la tercera `etiPrecioAlimentacionBD`.



106. Prueba el programa y comprueba el funcionamiento de los botones `JToggleButton`:



Observa como al pulsar los JToggleButton estos se quedan pulsados.

Si se vuelven a activar se “despulsan”.

107. Se pretende que el programa funcione de la siguiente forma:

ddddd. El usuario introducirá un precio base para el servicio que se vende.

eeee. A continuación, si el cliente quiere la instalación, activará el botón Instalación.

ffff. Si el cliente quiere la formación, activará el botón Formación.

ggggg. Si el cliente quiere la Alimentación de Base de Datos, activará el botón Alimentación BD.

hhhhh. Ten en cuenta que el cliente puede querer una o varias de las opciones indicadas.

iiii. Finalmente se pulsará el botón calcular y se calculará el precio total. Este precio se calcula de la siguiente forma:

Precio Total = Precio Base + Precio Extras.

El precio de los Extras dependerá de las opciones elegidas por el usuario. Por ejemplo, si el usuario quiere Instalación y Formación, los extras costarán 240 euros.

114. Así pues, se programará el *actionPerformed* del botón Calcular de la siguiente forma:


```

double precio_base;
double precio_instal; //precio instalación
double precio_for; //precio formacion
double precio_ali; //precio alimentacion

//Recojo datos desde la ventana:

precio_base = Double.parseDouble(txtPrecioBase.getText());
precio_instal = Double.parseDouble(etiPrecioInstalacion.getText());
precio_for = Double.parseDouble(etiPrecioFormacion.getText());
precio_ali = Double.parseDouble(etiPrecioAlimentacionBD.getText());

//Ahora que tengo los datos, puedo hacer cálculos.

//Al precio base se le van añadiendo precio de extras
//según estén o no activados los JToggleButton

double precio_total;

precio_total = precio_base;

if (tbtnInstalacion.isSelected()) { //Si se seleccionó instalación
    precio_total = precio_total+precio_instal;
}

if (tbtnFormacion.isSelected()) { //Si se seleccionó formación
    precio_total = precio_total+precio_for;
}

if (tbtnAlimentacionBD.isSelected()) { //Si se seleccionó Alimentación BD
    precio_total = precio_total+precio_ali;
}

//Finalmente pongo el resultado en la etiqueta
etiTotal.setText(precio_total+" €");

```

115. Veamos una explicación del código:

IIII. Primero se crean variables doubles (ya que se admitirán decimales) para poder hacer los cálculos.

mmmmm. Se extraerán los datos de la ventana y se almacenarán en dichas variables. Para ello, hay que convertir desde cadena a double:

```

precio_base = Double.parseDouble(txtPrecioBase.getText());
precio_instal = Double.parseDouble(etiPrecioInstalacion.getText());
precio_for = Double.parseDouble(etiPrecioFormacion.getText());
precio_ali = Double.parseDouble(etiPrecioAlimentacionBD.getText());

```

nnnnn. Una vez obtenidos los datos en forma numérica, ya se pueden hacer cálculos con ellos. El programa declara una nueva variable *precio_total* donde se introducirá el resultado. En primer lugar se introduce en esta variable el precio base.

```

double precio_total;

precio_total = precio_base;

```

ooooo. A continuación se le suma al precio_total los precios de los extras si el botón correspondiente está seleccionado. Esto se hace a través de if. Por ejemplo, para sumar el extra por instalación:

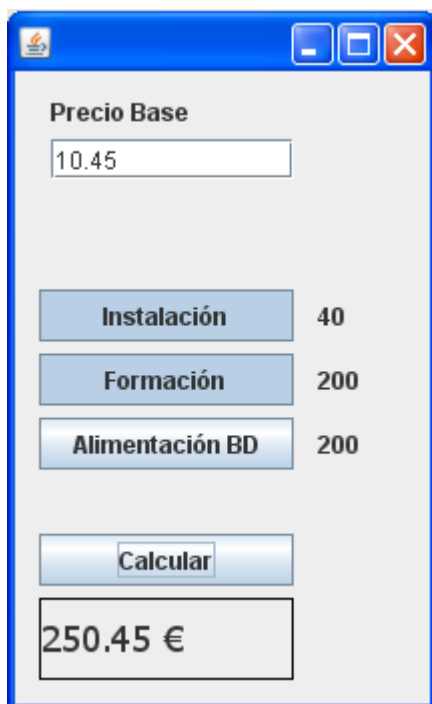
```
if (tbtnInstalacion.isSelected()) { //Si se seleccionó instalación
    precio_total = precio_total+precio_instal;
}
```

Esto significa: “Si el botón instalación está seleccionado, añade al precio total el precio por instalación”

Observa el uso del método *isSelected* para saber si el botón tbtnInstalacion ha sido seleccionado.

ppppp. Finalmente el resultado se muestra en la etiqueta de total.

121. Comprueba el funcionamiento del programa...



The screenshot shows a Java Swing window with a blue title bar. Inside, there's a section titled 'Precio Base' with a text input field containing '10.45'. Below this are three buttons: 'Instalación' (with '40' to its right), 'Formación' (with '200' to its right), and 'Alimentación BD' (with '200' to its right). All buttons are light blue. Below these is a 'Calcular' button. At the bottom, there's a larger text field displaying '250.45 €'.

Introduce una cantidad (usa el punto para los decimales)

Selecciona los extras que desees.

Pulsa Calcular y obtendrás el resultado.

122. Supongamos que normalmente (en el 90 por ciento de los casos) la instalación es solicitada por el usuario. Podría ser interesante que el botón Instalación ya saliera activado al ejecutarse el programa. Para ello, añade en el *Constructor* la siguiente línea.

```
tbtnInstalacion.setSelected(true);
```

Esta línea usa el método *setSelected* para activar al botón tbtnInstalación.

Comprueba esto ejecutando el programa.

CONCLUSIÓN

Los JToggleButton son botones que pueden quedarse pulsados.

A través del método isSelected podemos saber si un JToggleButton está seleccionado.

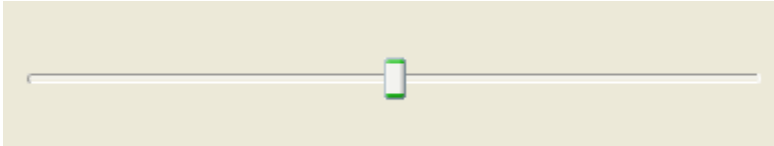
También puedes usar el método setSelected para seleccionar o no un botón de este tipo.

Realmente, estos botones no suelen ser muy usados, ya que pueden ser sustituidos por Cuadros de Verificación (JCheckBox) que son más conocidos.

EJERCICIO GUIADO. JAVA: SLIDERS

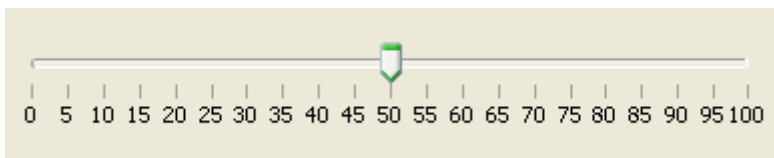
Introducción a los JSliders

La clase JSlider permite crear objetos como el siguiente:

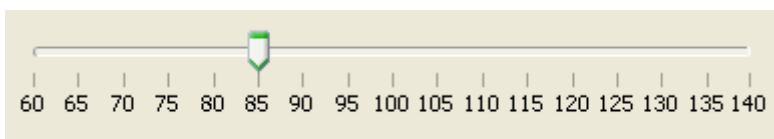


Estos elementos tienen un pequeño recuadro que se puede arrastrar a derecha o izquierda. Según la posición del recuadro, el JSlider tendrá un valor concreto.

El JSlider se puede configurar para que muestre los distintos valores que puede tomar:



También se puede configurar de forma que los valores mínimo y máximo sean distintos:

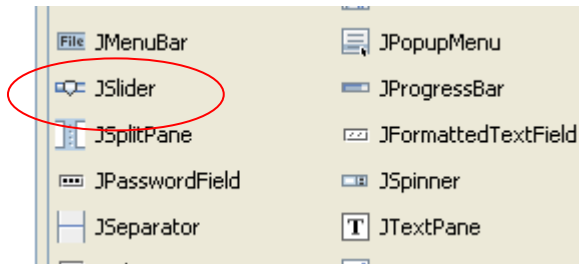


El valor que tiene un JSlider es el valor al que apunta el recuadro del JSlider. En la imagen anterior, el JSlider tiene un valor de 85.

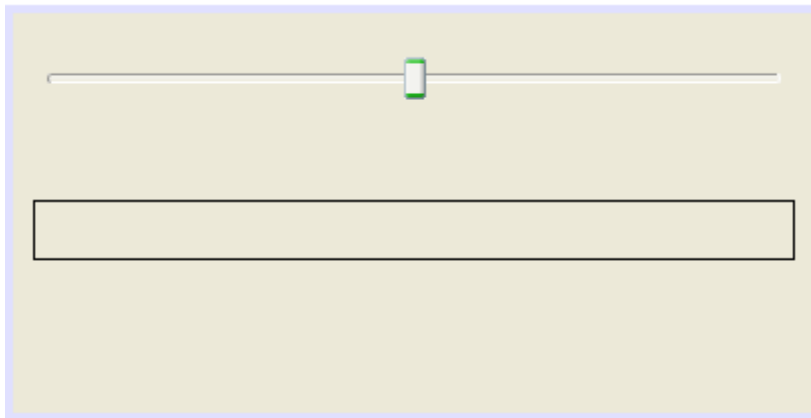
Se verá a continuación las características más interesantes de los JSlider y como programarlos.

Ejercicio guiado

- Crea un nuevo proyecto.
- Añade en él un JSlider. Su nombre será *slDeslizador*.



- Añade una etiqueta con borde. Su nombre será *etiValor*.
- La ventana tendrá el siguiente aspecto:




- Un JSlider tiene un valor mínimo y un valor máximo. El valor mínimo es el valor que tiene cuando el recuadro está pegado a la parte izquierda, mientras que el valor máximo es el valor que tiene cuando el recuadro está pegado a la parte derecha.

El valor mínimo y máximo del JSlider se puede cambiar. Busca las propiedades *maximum* y *minimum* del JSlider y asigna los siguientes valores:

Máximo: 500

Mínimo: 100

font	Tahoma 11 Pla
foreground	 [236,233,2
majorTickSpacing	0
maximum	500
minimum	100
minorTickSpacing	0
minorTickSpacing	
(int) Sets the number of values between mir	

- Se puede asignar un valor inicial al JSlider a través de su propiedad *value*. Busque esta propiedad y asigne un valor de 400. Observe donde se sitúa el recuadro del JSlider.

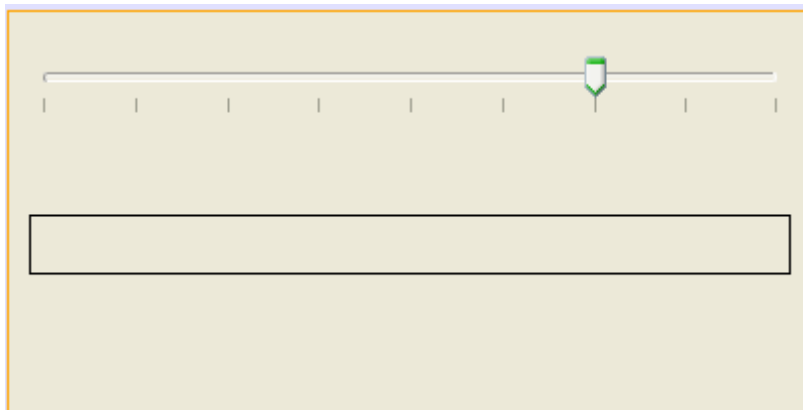
snapToTicks	<input type="checkbox"/>
toolTipText	null
value	400
Other Properties	
UIClassID	Other propertie

- Se puede mejorar el JSlider definiendo unas divisiones (medidas) Por ejemplo, haremos que cada 50 unidades aparezca una división. Para ello use la propiedad *majorTickSpacing* y asigne un 50.

foreground	<input type="checkbox"/> [236,233,
majorTickSpacing	50
maximum	500

- Esto, en realidad, no produce ningún cambio en el JSlider. Para que las divisiones se vean, es necesario que active también la propiedad *paintTicks*. Esta propiedad pintará divisiones en el JSlider:

paintLabels	<input type="checkbox"/>
paintTicks	<input checked="" type="checkbox"/>
paintTrack	<input checked="" type="checkbox"/>
snapToTicks	<input type="checkbox"/>

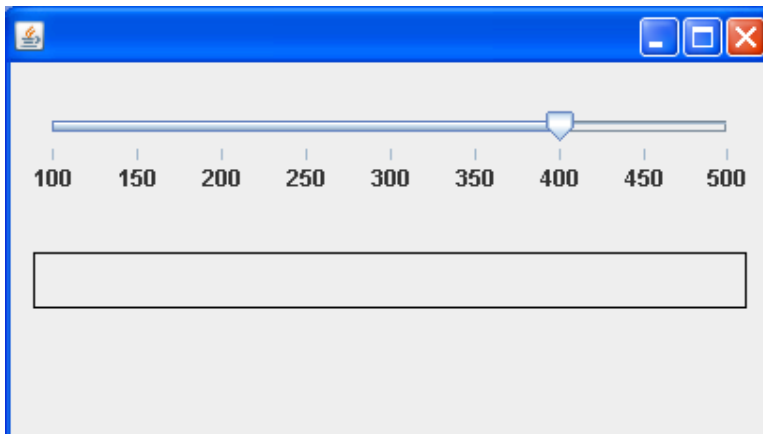


Medidas cada 50 unidades

- Aún se puede mejorar la presentación del JSlider, si hacemos que aparezca el valor de cada división. Para ello debes activar la propiedad *paintLabel*.

orientation	HORIZONTAL
paintLabels	<input checked="" type="checkbox"/>
paintTicks	<input checked="" type="checkbox"/>
paintTrack	<input checked="" type="checkbox"/>

- Ejecuta el programa para ver el funcionamiento del Deslizador y su aspecto. Debe ser parecido al siguiente:



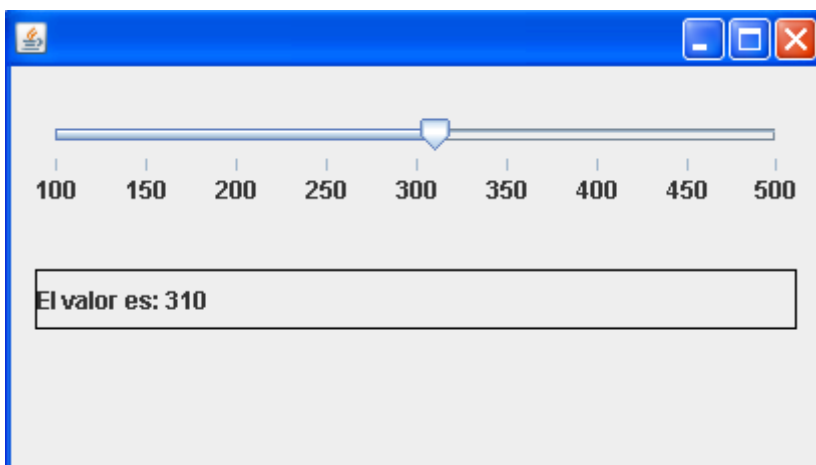
- Bien. Ahora se pretende que cuando el usuario arrastre el deslizador, en la etiqueta aparezca el valor correspondiente. Para ello tendrá que programar el evento *stateChanged* del JSlider.

El evento *stateChanged* sucede cuando el usuario arrastra el recuadro del deslizador.

En este evento programe lo siguiente:

```
etiValor.setText("El valor es: "+slDeslizador.getValue());
```

- Ejecute el programa y observe lo que sucede cuando arrastra el deslizador.
- La explicación del código es la siguiente:
 - o El método *getValue* del deslizador nos devuelve el valor que tiene actualmente el deslizador.
 - o Este valor es concatenado a la cadena "El valor es:" y es mostrado en la etiqueta a través del conocido *setText*.



Movemos aquí.
Y aparece el valor correspondiente aquí.

- A continuación se mencionan otras propiedades interesantes de los JSlider que puedes probar por tu cuenta:

orientation

Permite cambiar la orientación del JSlider. Podrías por ejemplo hacer que el JSlider estuviera en vertical.

minorTickSpacing

Permite asignar subdivisiones a las divisiones ya asignadas. Prueba por ejemplo a asignar un 10 a esta propiedad y ejecuta el programa. Observa las divisiones del JSlider.

snapToTicks

Cuando esta propiedad está activada, no podrás colocar el deslizador entre dos divisiones. Es decir, el deslizador siempre estará situado sobre una de las divisiones. Prueba a activarla.

paintTrack

Esta propiedad permite pintar o no la línea sobre la que se desliza el JSlider. Prueba a desactivarla.

CONCLUSIÓN

Los JSliders son objetos “deslizadores”. Permiten elegir un valor arrastrando un pequeño recuadro de derecha a izquierda o viceversa.

El valor de un JSliders puede ser obtenido a través de su método *getValue*.

Si quieres programar el cambio (el arrastre) en el deslizador, tienes que programar el evento llamado *stateChanged*.

EJERCICIO GUIADO. JAVA: SPINNER

Introducción a los JSpinner

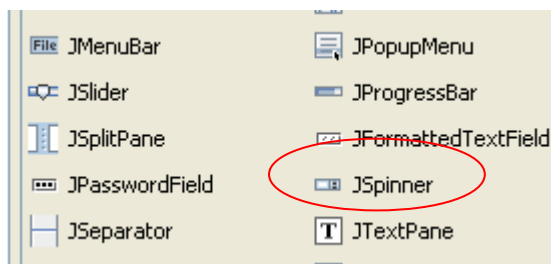
La clase JSpinner permite crear cuadros como el siguiente:



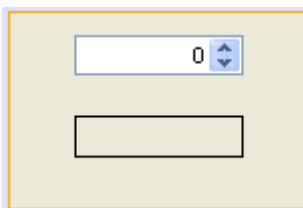
Son elementos muy comunes en los programas. A través de los dos botones triangulares se puede hacer que el valor del cuadro aumente o disminuya. También se puede escribir directamente un valor dentro del cuadro.

Ejercicio guiado

- Crea un nuevo proyecto.
- Añade en él un JSpinner. Su nombre será *spiValor*.



- Añade una etiqueta con borde. Su nombre será *etiValor*.
- La ventana tendrá el siguiente aspecto:



- Interesa que cuando cambie el JSpinner (ya sea porque se pulsaron los botones triangulares o porque se escribió dentro) aparezca el valor correspondiente dentro de la etiqueta. Para ello, tendrá que programar el evento *stateChanged* del JSpinner.

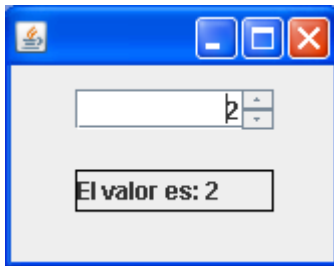
En el evento *stateChanged* introduzca el siguiente código:

```
etiValor.setText("El valor es: "+spiValor.getValue().toString());
```

- Como puedes observar, lo que hace el programa es recoger el valor que tiene el JSpinner a través del método *getValue* y luego se lo asigna a la etiqueta con el clásico *setText*. (Es muy parecido a los deslizadores)

Debes tener en cuenta que el valor devuelto no es un número ni una cadena, así que en el ejemplo se ha usado el método *toString()* para convertirlo a una cadena.

- Prueba el programa y observa su funcionamiento:



El usuario modifica el valor del JSpinner...

Y aquí aparece el valor seleccionado.

- Observa como los valores del JSpinner aumentan o disminuyen en 1. Por otro lado, no parece haber un límite para los valores del JSpinner.

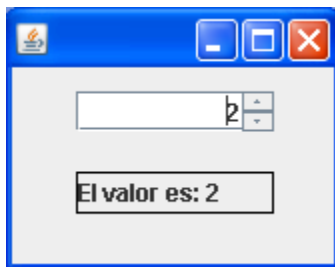
La pregunta ahora es: ¿Se puede modificar el contenido del JSpinner de forma que tenga unos valores concretos? La respuesta es sí. Veamos como hacerlo.

- Entra dentro del código del programa y, dentro del constructor, añade este código debajo de *initComponents*:

```
SpinnerNumberModel nm = new SpinnerNumberModel();  
nm.setMaximum(10);  
nm.setMinimum(0);  
spiValor.setModel(nm);
```

- Este código hace lo siguiente:
 - o El JSpinner, al igual que los JList y los JComboBox, es un objeto que contiene otro objeto “modelo”, y es el objeto “modelo” el que contiene los números visualizados en el JSpinner.
 - o En el código anterior se crea un “modelo” para el JSpinner, se definen los valores que contendrá, y luego se asigna al JSpinner. Estudiemos las líneas del código.

- o La primera línea crea un “modelo” llamado *nm*. Los modelos de los JSpinner son del tipo *SpinnerNumberModel*. Necesitarás incluir el import correspondiente (atento a la bombilla)
 - o En la segunda línea se define como valor máximo del modelo el 10, a través de un método llamado *setMaximum*.
 - o En la tercera línea se define como valor mínimo del modelo el 0, a través de un método llamado *setMinimum*.
 - o Finalmente se asigna el modelo creado al JSpinner.
 - o Este código, en definitiva, hará que el JSpinner muestre los valores comprendidos entre 0 y 10.
- Prueba el programa y observa los valores que puede tomar el JSpinner.



Ahora los valores están comprendidos entre 0 y 10

- Vamos a añadir otra mejora. Cambie el código del constructor por este otro. (Observa que solo se ha añadido una línea):

```
SpinnerNumberModel nm = new SpinnerNumberModel();
nm.setMaximum(10);
nm.setMinimum(0);
nm.setStepSize(2);
spiValor.setModel(nm);
```

- La línea añadida es:

```
nm.setStepSize(2);
```

Esta línea usa un método del modelo del JSpinner que permite definir el valor de cambio del JSpinner. Dicho de otra forma, esta línea hace que los valores del JSpinner salten de 2 en 2.

- Ejecuta el programa de nuevo y observa como cambian los valores del JSpinner.
- El modelo del JSpinner tiene también un método llamado *setValue* que permite asignar un valor inicial al modelo. Pruebe a usar este método para hacer que el JSpinner muestre desde el principio el valor 4.

CONCLUSIÓN

Los JSpinners son objetos que permiten seleccionar un número, ya sea escribiéndolo en el recuadro, o bien a través de dos botones triangulares que permiten aumentar o disminuir el valor actual.

Los JSpinners son objetos con “modelo”. Es decir, este objeto contiene a su vez otro objeto “modelo” que es el que realmente contiene los datos.

Datos → Modelo → JSpinner

Para definir el contenido del JSpinner es necesario crear un modelo del tipo SpinnerNumberModel. Se le asigna al modelo los números deseados, y finalmente se une el modelo con el JSpinner.

El objeto modelo del JSpinner permite definir el valor mínimo y el valor máximo, así como el intervalo de aumento de los valores.

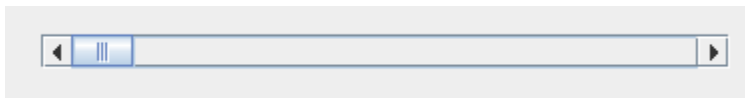
EJERCICIO GUIADO. JAVA: SCROLLBARS

Introducción a las JscrollBars (Barras de desplazamiento)

La clase JScrollBar permite crear barras de desplazamiento independientes, como la que se muestra a continuación:

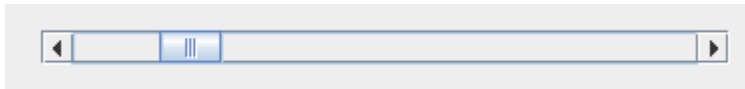


La barra tiene un valor mínimo, que se consigue haciendo que el recuadro de la barra de desplazamiento esté pegado a la parte izquierda.



Valor mínimo

Cuando se pulsa algunos de los botones de la barra de desplazamiento, el valor de la barra se incrementa / decrementa poco a poco. A este incremento / decremento lo llamaremos *incremento unitario*.



Decrementa el valor poco a poco (*incremento unitario*)

Incrementa el valor poco a poco (*incremento unitario*)

Cuando se pulsa directamente sobre la barra, el valor de la barra se incrementa / decrementa en mayor cantidad. A este incremento / decremento lo llamaremos *incremento en bloque*.

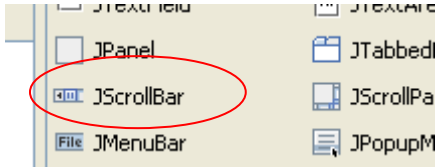


Al pulsar directamente sobre la barra se decrementa en mayor cantidad (*incremento en bloque*)

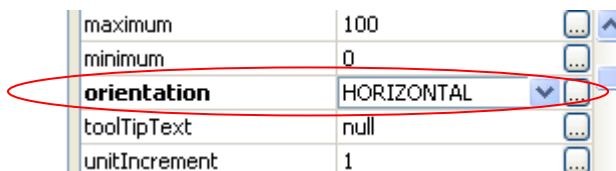
Al pulsar directamente sobre la barra se incrementa en mayor cantidad (*incremento en bloque*)

Ejercicio guiado

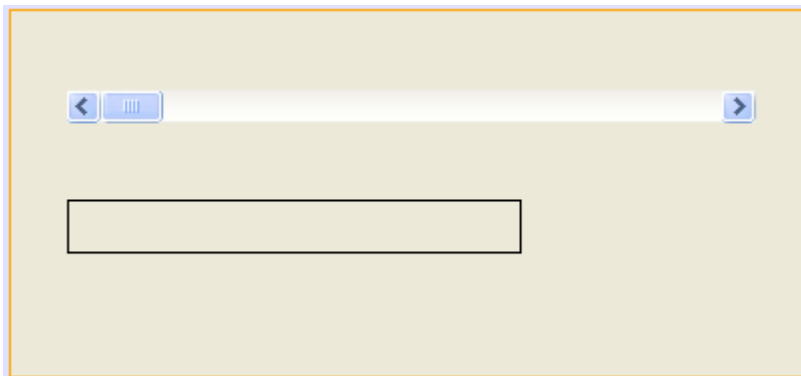
1. Para comprender mejor el funcionamiento de las barras de desplazamiento se creará un proyecto nuevo.
2. Añade en el proyecto una barra de desplazamiento (JScrollBar) y llámala *desValor*.



3. La barra de desplazamiento aparecerá en vertical. Use la propiedad de la barra llamada *Orientation* para hacer que la barra aparezca en posición horizontal.



4. Añade también una etiqueta con borde y llámala *etiValor*.
5. La ventana debe quedar más o menos así:



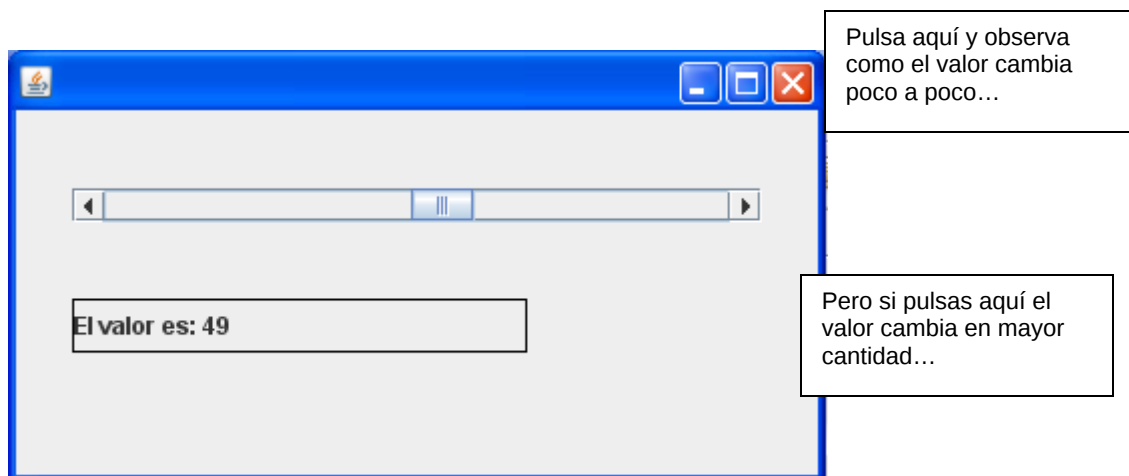
6. Interesa que cuando el usuario cambie de alguna manera la barra de desplazamiento, en la etiqueta aparezca el valor de la barra.

Para ello, se debe programar el evento *AdjustmentValueChanged* de la barra de desplazamiento.

En este evento programa lo siguiente:

```
etiValor.setText("El valor es: "+desValor.getValue());
```

7. Como ves, se coloca en la etiqueta el valor de la barra. El valor de la barra se obtiene con el método *getValue*. Ejecuta el programa para ver su funcionamiento.



8. Sigamos estudiando el programa. Se pide que cambies las siguientes propiedades de tu barra:

Minimum – Permite asignar el valor mínimo de la barra. Escribe un 50

Maximum – Permite asignar el valor máximo de la barra. Escribe un 150

foreground	[236,233]
maximum	150
minimum	50
orientation	HORIZONTAL

UnitIncrement – Permite cambiar el *incremento unitario*. Escribe un 2.

toolTipText	null
unitIncrement	2
value	50

BlockIncrement – Permite cambiar el *incremento en bloque*. Escribe un 20.

background	[212,208]
blockIncrement	20
componentPopupMenu	<ninguna>
font	Tahoma 11 P

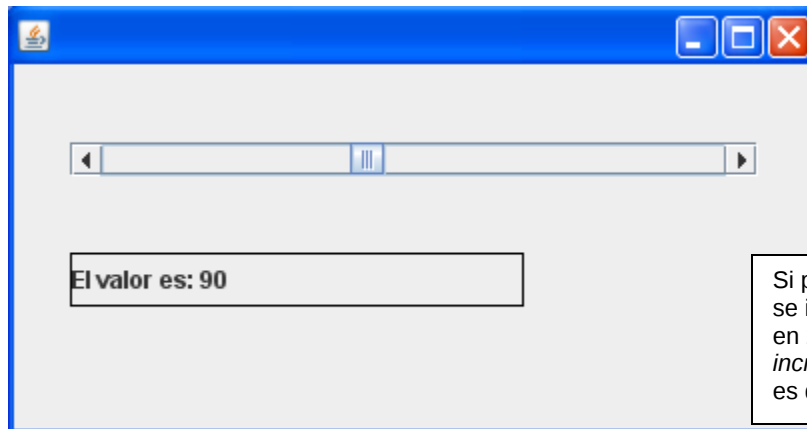
VisibleAmount – Permite cambiar el ancho del recuadro de la barra. Escribe un 5.

value	50
visibleAmount	5
- Otras Propiedades	



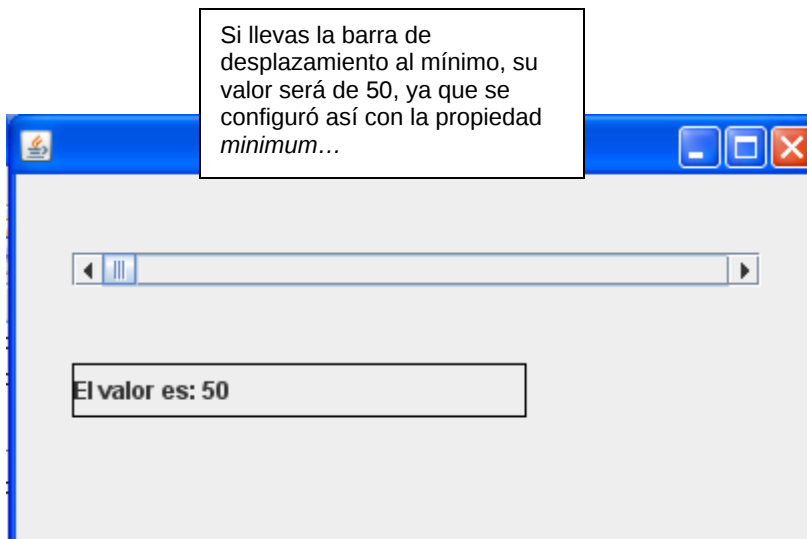
5

9. Ejecuta ahora el programa y comprueba su funcionamiento:

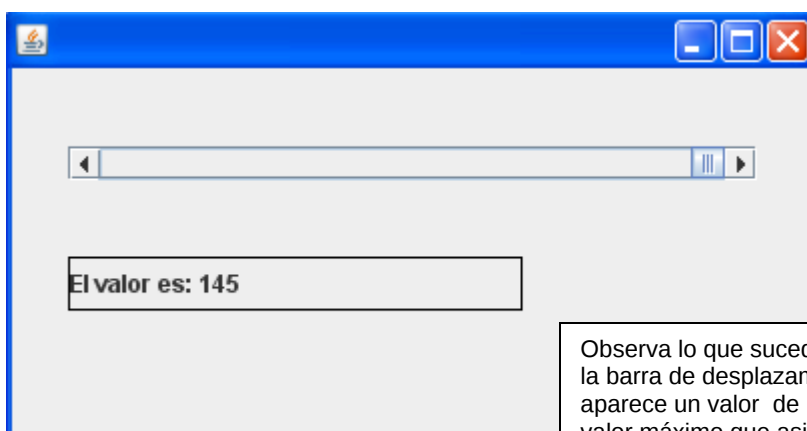


Si pulsas aquí, el valor se incrementa de 2 en 2, ya que el *incremento unitario* se configuró en 2.

Si pulsas aquí, el valor se incrementa de 20 en 20, ya que el *incremento en bloque* es de 20.



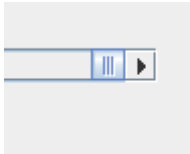
Si llevas la barra de desplazamiento al mínimo, su valor será de 50, ya que se configuró así con la propiedad *minimum...*



Observa lo que sucede cuando llevas la barra de desplazamiento al máximo: aparece un valor de 145, cuando el valor máximo que asignamos fue de 150 ¿por qué?

10. Tal como se ha indicado anteriormente, pasa algo raro con la barra de desplazamiento cuando esta está al máximo. Se esperaba que alcanzara el valor 150, y sin embargo, el valor máximo alcanzado fue de 145. La explicación es la siguiente:

Valor máximo (150) *



Valor de la barra (145) **

* Nuestra barra tiene un valor máximo de 150.

** Sin embargo, el valor de la barra viene indicado por el lado izquierdo del recuadro interno.

*** Como el recuadro interno tiene un ancho definido a través de la propiedad *VisibleAmount*, el valor máximo que la barra puede alcanzar es de:

$$\text{Valor} = \text{ValorMáximo} - \text{Ancho del recuadro.}$$

Es decir,

$$\text{Valor alcanzable} = 150 - 5 = 145$$

11. A través del método *setValue* de la barra de desplazamiento se puede asignar un valor inicial a la barra. Programme en el constructor de su programa lo necesario para que la barra de desplazamiento tenga un valor de 70 al empezar el programa.

CONCLUSIÓN

Las **JScrollBar**s son barras de desplazamiento independientes. Al igual que los **JSliders**, las **JScrollBar**s tienen un valor concreto, que puede ser obtenido a través del método *getValue*.

Entre las características programables de una barra de desplazamiento, tenemos las siguientes:

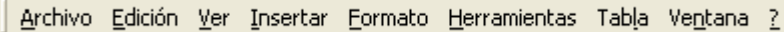
- Valor mínimo (propiedad *Minimum*)
- Valor máximo (propiedad *Maximum*)

- Incremento unitario (propiedad `UnitIncrement`)
- Incremento en bloque (propiedad `BlockIncrement`)
- Tamaño del recuadro de la barra (propiedad `VisibleAmount`)

EJERCICIO GUIADO. JAVA: BARRA DE MENUS

Barras de Menús

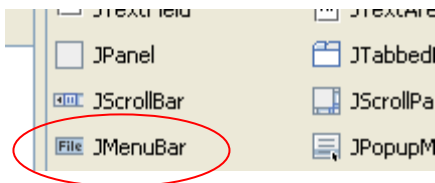
La barra de menús nos permitirá acceder a las opciones más importantes del programa. Todo programa de gran envergadura suele tener una barra de menús.



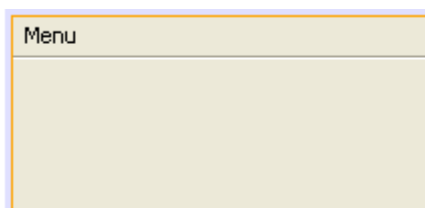
Ejercicio guiado

12. Veamos como añadir una barra de menús a nuestras aplicaciones. En primer lugar, crea un proyecto con el NetBeans.

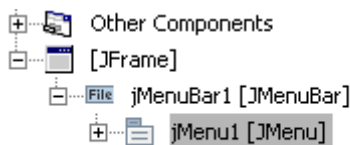
13. Añade a tu ventana un objeto JMenuBar



14. En la parte superior de tu ventana aparecerá esto:

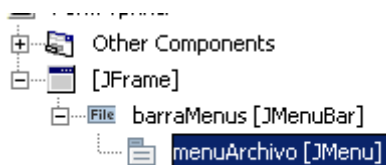


15. En el inspector (parte inferior izquierda) observarás como aparece un objeto JMenuBar, y, dentro de él, un objeto del tipo JMenu. Los objetos JMenu representan las opciones principales contenidas dentro de la barra de menús.



16. Aprovecha el *Inspector* para cambiar el nombre al objeto JMenuBar. Llámalo *barraMenus*.

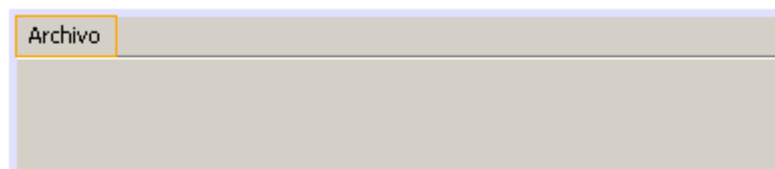
17. Cambia también el nombre al objeto JMenu. Asígnale el nombre *menuArchivo*. El *Inspector* tendrá el siguiente aspecto:



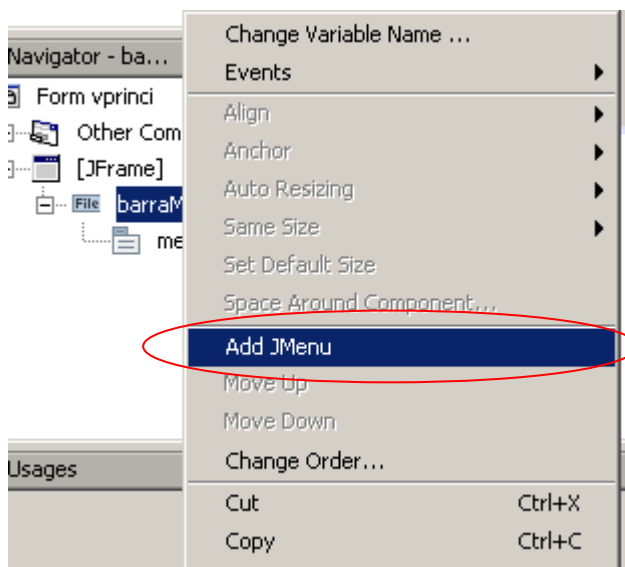
18. Ahora, la única opción de la barra de menús muestra el texto “Menu”. Esto se puede cambiar seleccionándola y cambiando su propiedad *text*. Asígnale el texto “Archivo” a la opción del menú:



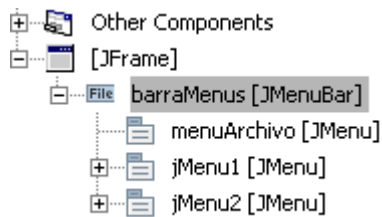
19. Ahora el aspecto de la barra de menús será el siguiente:



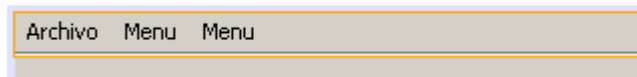
20. Puedes añadir más opciones principales a la barra de menús haciendo clic con el derecho sobre el objeto de la barra de menús y activando la opción “Añadir JMenu”.



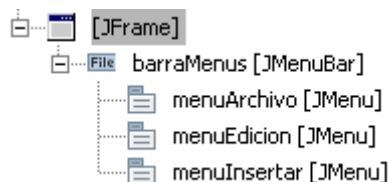
21. Añade dos opciones más a la barra de menús. El inspector debe tener ahora el siguiente aspecto:



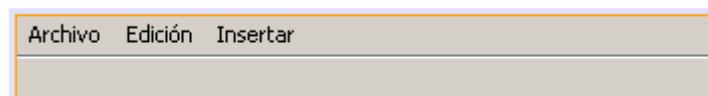
22. Y la barra de menús presentará este otro aspecto:



23. Cambia los nombres de las dos nuevas opciones. Sus nombres serán: menuEdicion y menuInsertar.

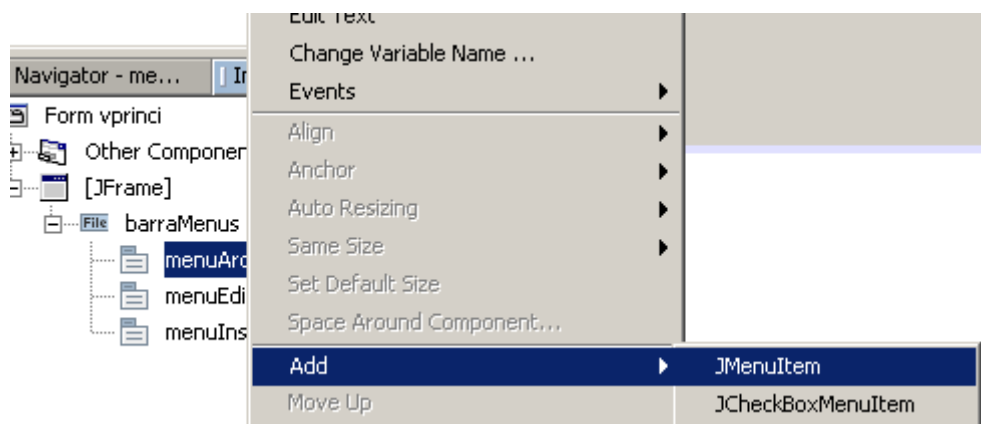


24. Cambia los textos de ambas opciones. Sus textos serán: “Edición” e “Insertar”.



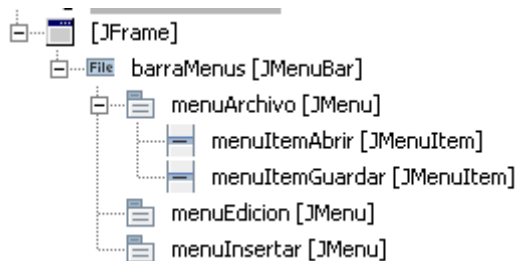
25. Ya tenemos creada la barra de menús (JMenuBar) con sus opciones principales (JMenu). Ahora se tendrán que definir las opciones contenidas en cada opción principal. Por ejemplo, crearemos las opciones contenidas en el menú Archivo.

26. Haz clic con el botón derecho sobre el objeto menuArchivo y activa la opción “Añadir – JMenuItem”.



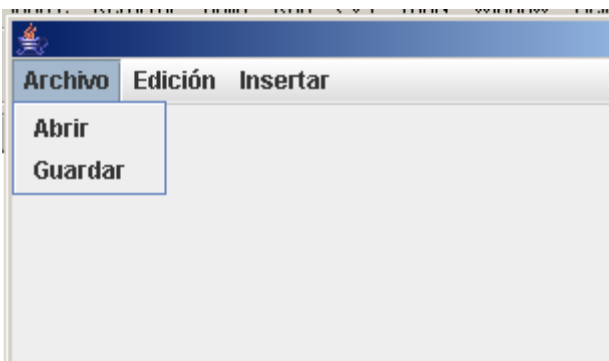
Los JMenuItem son objetos que representan las opciones contenidas en los menús desplegables de la barra de menús.

27. Añade un JMenuItem más al menuArchivo y luego cambia el nombre a ambos. Sus nombres serán *menuItemAbrir* y *menuItemGuardar*. El aspecto del *Inspector* será el siguiente:



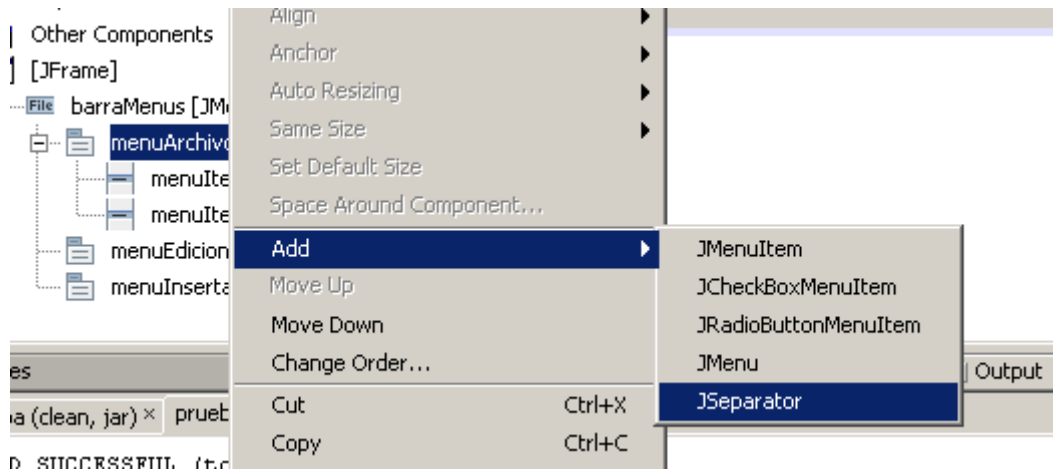
28. Usa ahora la propiedad *Text* de ambos JMenuItem para asignarles un texto. El primero tendrá el texto "Abrir" y el segundo el texto "Guardar".

29. Ya podemos ejecutar el programa para ver que es lo que se ha conseguido. Use el menú:



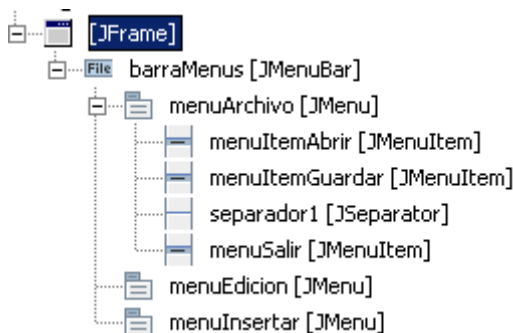
Observa como la opción Archivo se despliega mostrando dos submenús: Abrir y Guardar.

30. Seguiremos añadiendo elementos al menú. Ahora haga clic con el derecho sobre el elemento *menuArchivo* y añada un JSeparator.

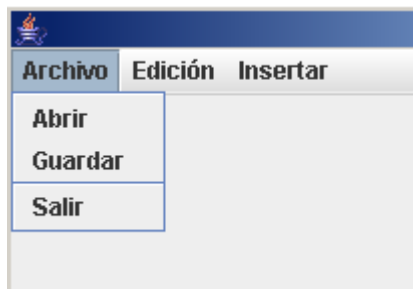


Los JSeparator son objetos que definen una separación entre las opciones de un menú. Cámbiele el nombre y llámelo “separador1”:

31. Añada un nuevo JMenuItem al menú Archivo y ponle el nombre menuSalir. El texto de esta opción será “Salir” (use su propiedad *text*) El aspecto del *Inspector* será el siguiente:



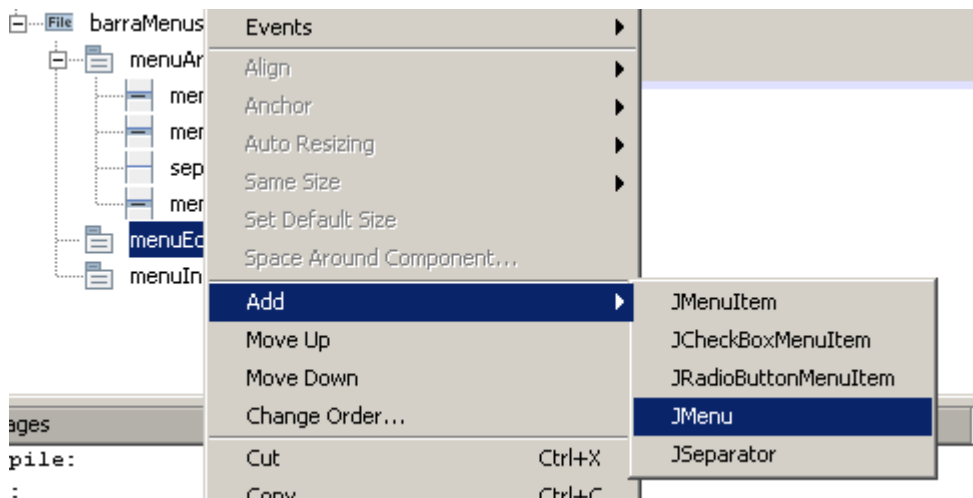
32. Ejecuta el programa y observa el contenido de la opción Archivo del menú:



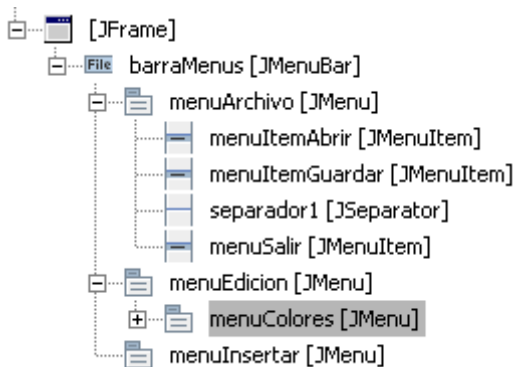
Observa el efecto que produce el separador.

33. Un JMenu representa las opciones principales de la barra de menús. A su vez, un JMenuItem contiene JMenuItem, que son las opciones contenidas en cada opción principal, y que se ven cuando se despliega el menú.

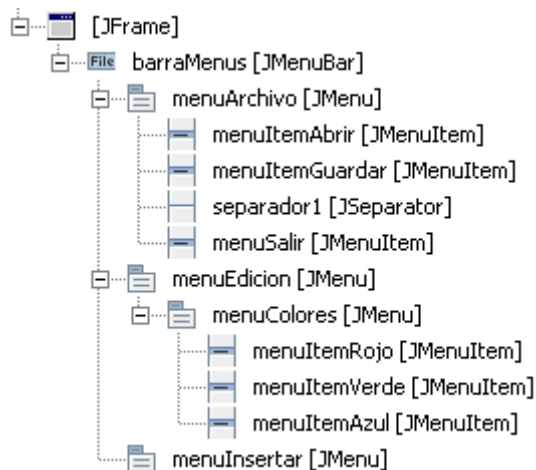
Sin embargo, un JMenu puede contener a otros JMenu, que a su vez contendrán varios JMenuItem. Usando el botón derecho del ratón y la opción “Añadir”, añade un JMenu dentro de menuEdicion:



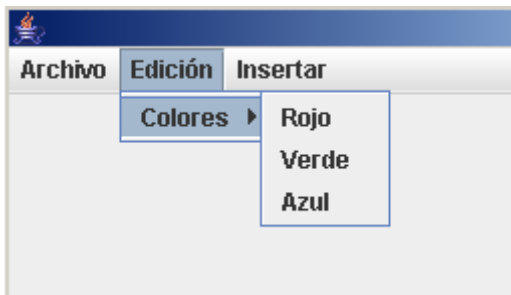
34. Llama al nuevo JMenu *menuColores* y asigne el texto “Colores”.



35. Ahora añada dentro del *menuColores* tres JMenuItem llamados respectivamente: *menuItemRojo*, *menuItemVerde*, *menuItemAzul*. Sus textos serán “Rojo”, “Verde” y “Azul”.



36. Ejecuta el programa y observa como ha quedado el menú Edición:



La opción Edición (JMenu) contiene una opción Colores (JMenu) que a su vez contiene las opciones Rojo, Verde y Azul (JMenuItems)

37. De nada sirve crear un menú si luego este no reacciona a las pulsaciones del ratón. Cada objeto del menú tiene un evento `ActionPerformed` que permite programar lo que debe suceder cuando se active dicha opción del menú.

38. Marque en el inspector el objeto `menuItemRojo` y acceda a su evento `ActionPerformed`. Dentro de él programe este sencillo código:

```
this.getContentPane().setBackground(Color.RED);
```

Este código cambia el color de fondo de la ventana a rojo.

39. Compruebe el funcionamiento de la opción “Rojo” del menú ejecutando el programa.

40. Programa tu mismo las opciones “Verde” y “Azul”.

CONCLUSIÓN

Las barras de menús son un conjunto de objetos de distinto tipo que se contienen unos a los otros:

La barra en sí está representada por un objeto del tipo JMenuBar.

La barra contiene opciones principales, representadas por objetos JMenu.

Las opciones principales contienen opciones que aparecen al desplegarse el menú. Estas opciones son objetos del tipo JMenuItem.

Un JMenu también puede contener otros JMenu, que a su vez contendrán JMenuItem.

También puede añadir separadores (JSeparator) que permiten visualizar mejor las opciones dentro de un menú.

EJERCICIO GUIADO. JAVA: BARRA DE HERRAMIENTAS

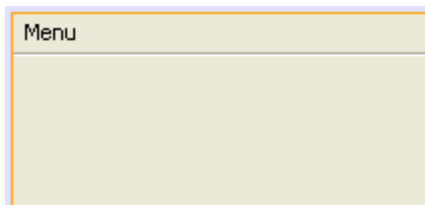
Barras de herramientas

Una barra de herramientas es básicamente un contenedor de botones y otros elementos propios de la ventana.

A través de estos botones se pueden activar de forma rápida las opciones del programa, las cuales suelen estar también incluidas dentro de la barra de menús.

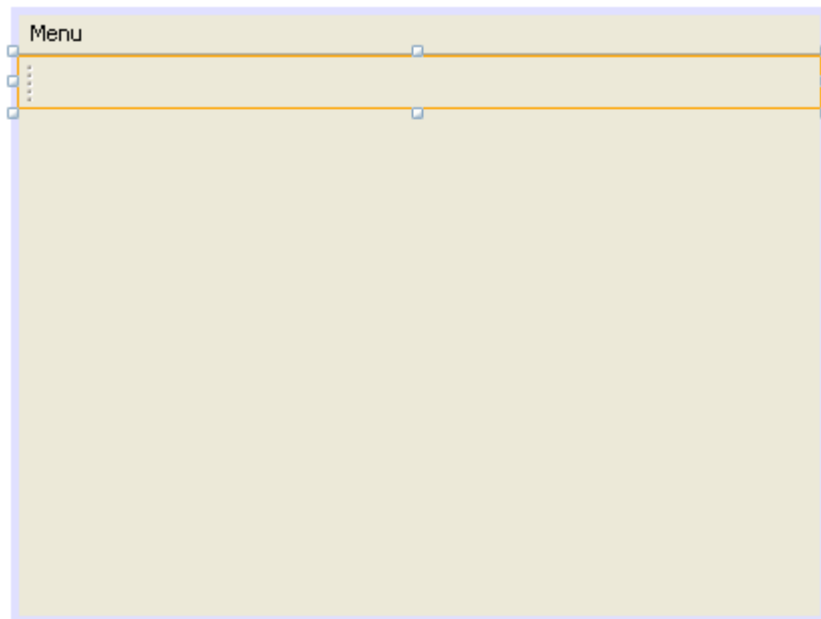
Ejercicio guiado

41. Veamos como añadir una barra de herramientas a nuestras aplicaciones. En primer lugar, crea un proyecto con el NetBeans.
42. Añade a tu ventana un objeto JMenuBar (una barra de menús)
43. En la parte superior de tu ventana aparecerá esto:



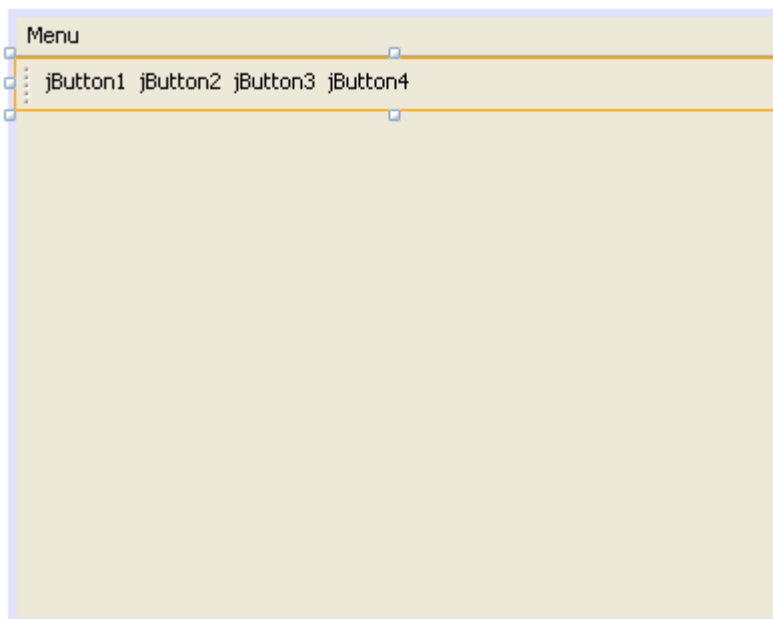
44. Debajo de la barra de menús colocaremos una barra de herramientas, así que añade un objeto del tipo JToolBar. Haz que la barra se coloque debajo de la barra de menús y que alcance desde la parte izquierda de la ventana a la parte derecha.

La ventana quedará así:

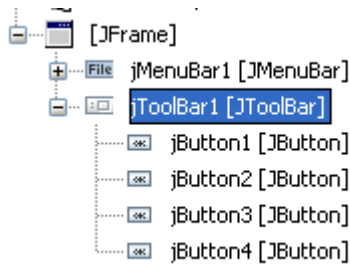


45. Las barras de herramientas son simples contenedoras de objetos. Dentro de ellas se pueden colocar botones, combos, etiquetas, etc.

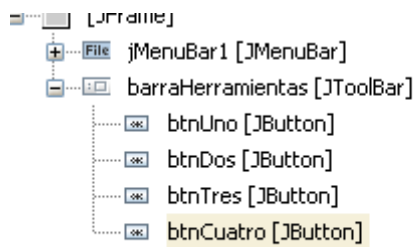
Normalmente, las barras de herramientas contienen botones. Así que añade cuatro botones (JButton) dentro de la barra. Solo tienes que colocarlos dentro de ella.



46. Puedes ver si los botones están bien colocados observando el *Inspector*: Observa como los botones colocados se encuentran dentro de la barra.

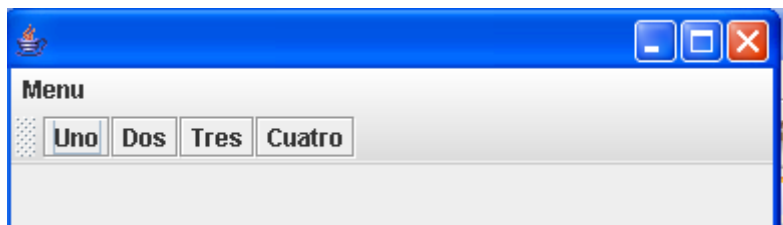


47. Aprovecharemos el inspector para cambiar el nombre a la barra y a cada botón. A la barra la llamaremos *barraHerramientas*, y a los botones los llamaremos *btnUno*, *btnDos*, *btnTres* y *btnCuatro*:



48. Cambia el texto de los botones. Estos contendrán el texto: “Uno”, “Dos”, “Tres” y “Cuatro”.

49. Ejecuta el programa y observa el resultado.



50. La forma de programar cada botón no varía, aunque estos se encuentren dentro de la barra herramientas. Solo hay que seleccionar el botón y acceder a su evento *actionPerformed*.

51. Solo como demostración de esto último, entra en el *actionPerformed* del primer botón y programa esto:

```
JOptionPane.showMessageDialog(null, "Activaste el botón uno");
```

Luego ejecuta el programa y comprueba el funcionamiento del botón.

52. Los botones de la barra de herramientas normalmente no contienen texto, sino que contienen un icono que representa la función que realiza. La forma de colocar un icono dentro de un botón es a través de su propiedad *icon*.

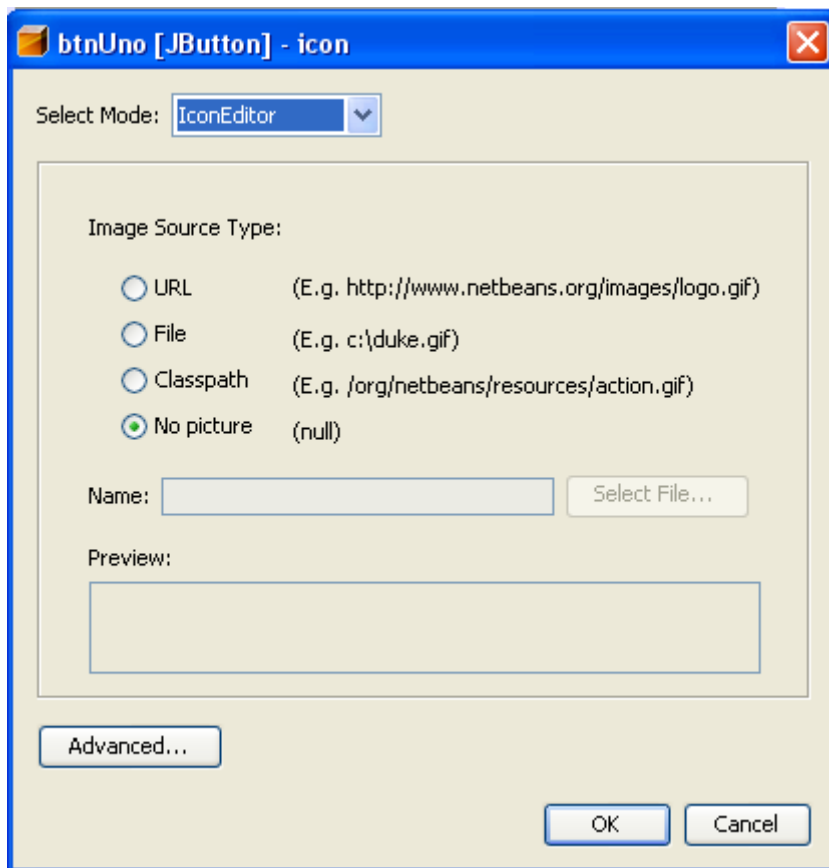
53. A través de la propiedad *icon* de un botón podrá seleccionar un fichero de imagen que contenga la imagen a mostrar en el botón.

54. Activa la propiedad *icon* del primer botón. Luego elige la opción *Fichero* y pulsa el botón *Seleccionar Fichero* para buscar un fichero con imagen.

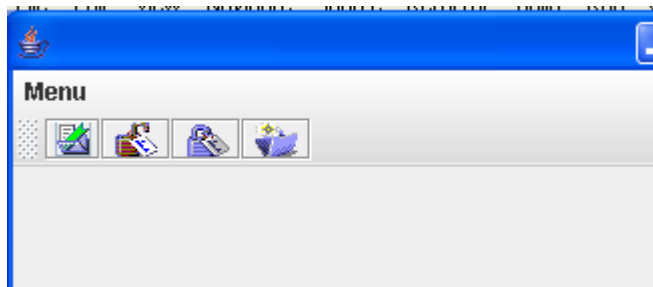
Nota: Busca un fichero de imagen que sea del tipo .gif o .jpg.

Nota: Procura que la imagen sea pequeña.

Nota: Se recomienda buscar imágenes .gif en Internet para practicar.



55. Una vez colocadas las imágenes a los botones, se puede quitar el texto de estos. Un ejemplo de cómo podría quedar la barra de herramientas es este:



CONCLUSIÓN

Las barras de herramientas son simplemente contenedores de objetos. Normalmente botones.

Los elementos de la barra de herramientas se manejan de la misma forma que si no estuvieran dentro de la barra.

Lo normal es hacer que los botones de la barra no tengan texto y tengan iconos asociados.

EJERCICIO GUIADO. JAVA: MENUS EMERGENTES

El evento mouseClicked

El evento `mouseClicked` es capaz de capturar un clic del ratón sobre un determinado elemento de la ventana.

Este evento recibe como parámetro un objeto del tipo `MouseEvent`, y gracias a él se puede conseguir información como la siguiente:

1. Qué botón del ratón fue pulsado.
2. Cuantas veces (clic, doble clic, etc)
3. En qué coordenadas fue pulsado el botón.
4. Etc.

Se puede usar esta información para saber por ejemplo si se pulsó el botón derecho del ratón, y sacar en este caso un menú contextual en pantalla.

En este ejercicio guiado se estudiarán las posibilidades del evento `mouseClicked` y se aplicarán a la creación y visualización de menús contextuales (o emergentes)

Ejercicio guiado

56. Crea un nuevo proyecto.

57. No hace falta que añada nada a la ventana.

58. Programaremos la pulsación del ratón sobre el formulario, así que haga clic sobre el formulario y active el evento *`mouseClicked`*.

59. Observe el código del evento:

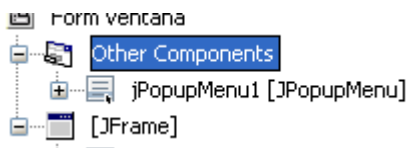
```
private void formMouseClicked(java.awt.event.MouseEvent evt) {  
    // TODO add your handling code here:  
  
}
```

Este evento recibe como parámetro un objeto llamado *`evt`* del tipo `MouseEvent` (en rojo en el código) que nos permite saber en qué condiciones se hizo clic.

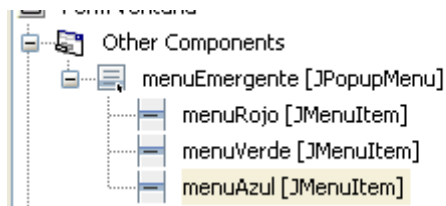
60. Dentro del evento programe lo siguiente:

```
if (evt.getButton()==1) {  
    JOptionPane.showMessageDialog(null,"Pulso el izquierdo");  
} else if (evt.getButton()==2) {  
    JOptionPane.showMessageDialog(null,"Pulso el central");  
} else if (evt.getButton()==3) {  
    JOptionPane.showMessageDialog(null,"Pulso el derecho");  
}
```

61. Ejecuta el programa y haz clic sobre el formulario con el botón derecho, con el izquierdo y con el central. Observa el resultado.
62. Ahora quizás puedas comprender el código anterior. En él, se usa el método *getButton* del objeto *evt* para saber qué botón se pulsó. El método *getButton* devuelve un entero que puede ser 1, 2 o 3 según el botón pulsado.
63. Se puede aprovechar el método *getButton* para controlar la pulsación del botón derecho del ratón y así sacar un menú contextual. Pero antes, es necesario crear el menú.
64. Agrega a tu formulario un objeto del tipo *JPopupMenu*. Estos objetos definen menús emergentes.
65. Los objetos *JPopupMenu* no se muestran en el formulario, pero puedes verlo en el *Inspector* dentro de la rama de *Otros Componentes*:



66. Aprovecharemos el inspector para cambiar el nombre al menú. Llámalo *menuEmergente*.
67. Los menús emergentes se crean igual que las opciones de menús normales, añadiendo con el botón derecho del ratón objetos *JMenuItem*.
68. Añade al menú emergente tres *JMenuItem*, y asígneles los siguientes nombres a cada uno: *menuRojo*, *menuVerde*, *menuAzul*. El *inspector debería tener el siguiente aspecto*:



69. Tienes que cambiar la propiedad *text* de cada opción del menú. Recuerda que esta propiedad define lo que aparece en el menú. Asignarás los siguientes textos: "Rojo", "Verde" y "Azul".
70. El menú emergente ya está construido. Ahora tenemos que hacer que aparezca cuando el usuario pulse el botón derecho del ratón sobre el formulario. Para ello,

entraremos de nuevo en el evento *mouseClicked* del formulario y cambiaremos su código por el siguiente:

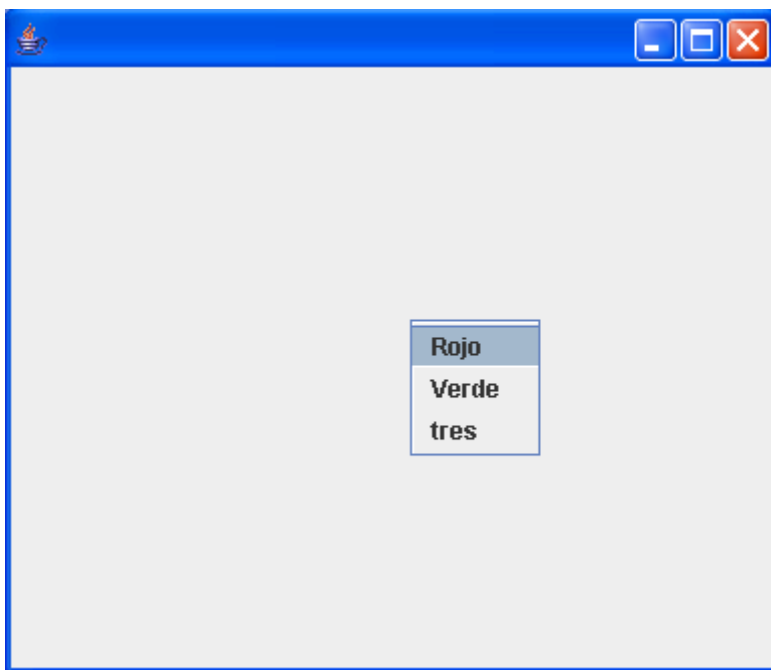
```
menuEmergente.show(this, evt.getX(), evt.getY());
```

71. Este código significa lo siguiente:

5. El método `show` le da la orden al `menuEmergente` para que se muestre.
6. El método `show` recibe tres elementos: por un lado la ventana donde actúa (`this`)
7. Por otro lado la posición `x` donde debe mostrarse el menú. Esta posición es aquella donde se pulsó el ratón, y se puede conseguir gracias al método `getX` del objeto `evt`.
8. Por último se necesita la posición `y`. Esta posición se puede conseguir gracias al método `getY` del objeto `evt`.

Es decir, decidimos mostrar el menú emergente justo en las coordenadas donde se hizo clic.

72. Ejecuta el programa y observa el resultado.



Al hacer clic con el derecho se mostrará el menú contextual.

73. Para hacer que al pulsarse una opción suceda algo, solo hay que activar el método *actionPerformed* del *JMenuItem* correspondiente. Por ejemplo, active el *actionPerformed* del `menuRojo` y dentro programe lo siguiente:

```
this.getContentPane().setBackground(Color.RED);
```

74. Ejecuta el programa y comprueba lo que sucede al pulsar la opción Rojo del menú contextual.

CONCLUSIÓN

Los menús contextuales son objetos del tipo `JPopupMenu`. Estos objetos contienen `JMenuItem` al igual que las opciones de menú normales.

Cuando se asigna un `JPopupMenu` a un formulario, no aparece sobre la ventana, pero sí en el *inspector*.

Para hacer que aparezca el menú emergente, es necesario programar el evento *mouseClicked* del objeto sobre el que quiera que aparezca el menú.

Tendrá que usar el método *show* del menú emergente para mostrar dicho menú.

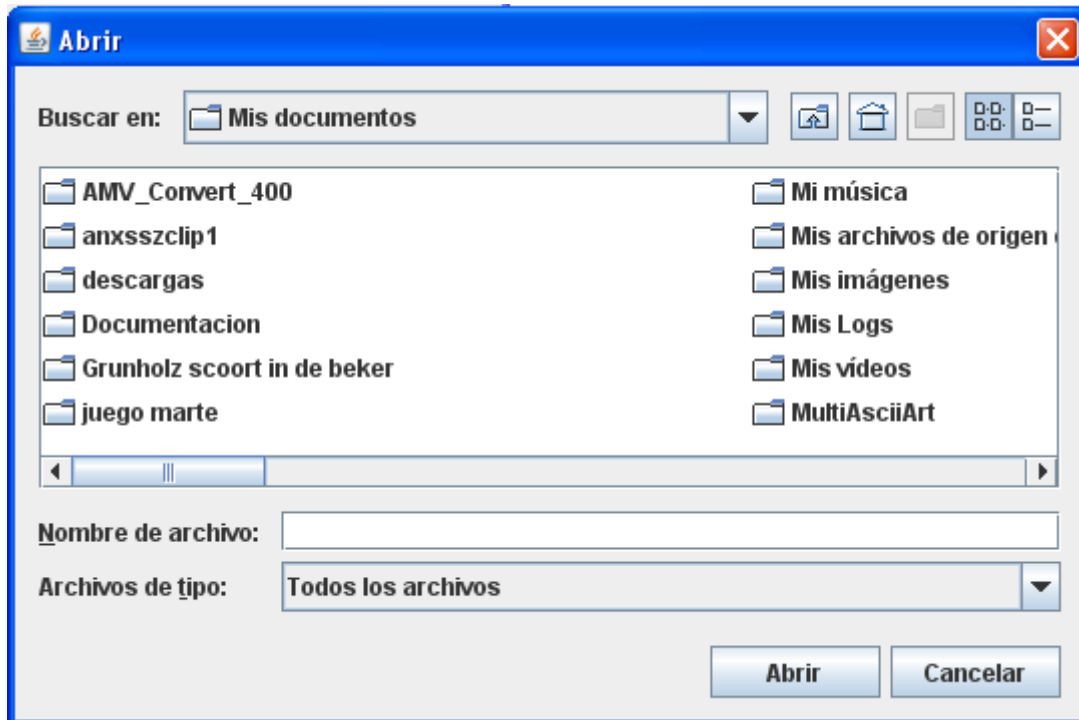
EJERCICIO GUIADO. JAVA: FILECHOOSER

Cuadros de diálogo Abrir y Guardar

Las opciones Abrir y Guardar son opciones muy comunes en las aplicaciones. Estas opciones permiten buscar en el árbol de carpetas del sistema un fichero en concreto y abrirlo, o bien guardar una información dentro de un fichero en alguna carpeta.

Java proporciona una clase llamada JFileChooser (*elegir fichero*) que permite mostrar la ventana típica de Abrir o Guardar:

Ventana Abrir fichero:

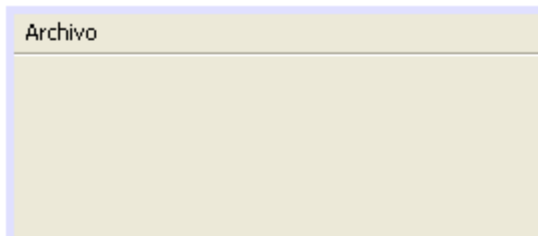


(La ventana de guardar es la misma, solo que muestra en su barra de título la palabra *Guardar*)

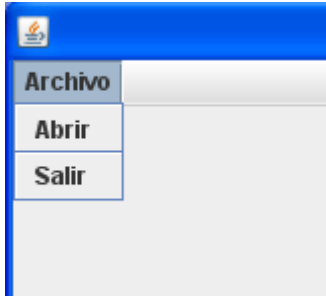
El objeto JFileChooser nos facilita la labor de elegir el fichero, pero no realiza la apertura o la acción de guardar la información en él. Esto tendrá que ser programado.

Ejercicio guiado

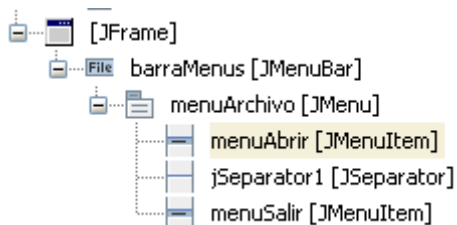
- Vamos a practicar con el JFileChooser. Para ello, crea un nuevo proyecto.
- Añade en el proyecto los siguientes elementos:
 - o Una barra de menús. Llámala *barraMenus*.
 - o Dentro de ella una opción “Archivo” llamada *menuArchivo*.
 - o Dentro de la opción “Archivo”, introduce los siguientes elementos:
 - Una opción “Abrir”, llamada *menuAbrir*.
 - Un separador (llámalo como quieras)
 - Una opción “Salir”, llamada *menuSalir*.
- Una vez hecho esto tu formulario tendrá la siguiente forma:



- Si ejecutas el programa el menú se verá así:



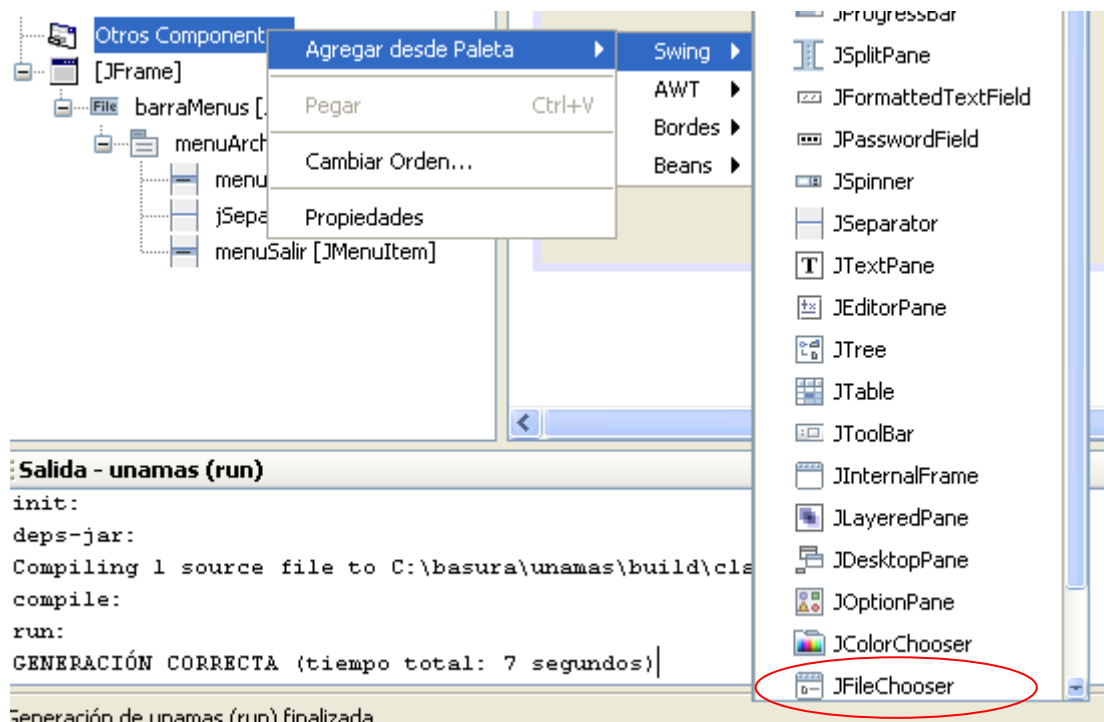
- Si observas el *Inspector*, tendrá un aspecto parecido al siguiente:



- Para que al pulsar la opción “Abrir” de nuestro programa aparezca el diálogo de apertura de ficheros, es necesario añadir a nuestro programa un objeto del tipo JFileChooser.

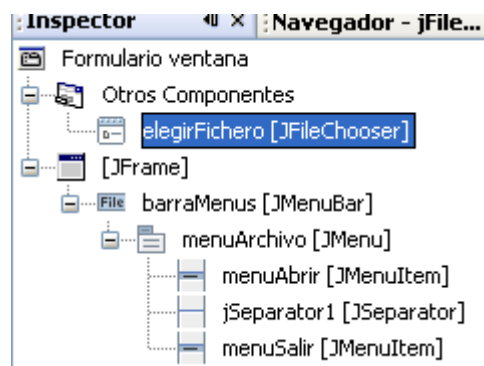
Los objetos JFileChooser se añadirán el la zona de “*Otros Componentes*” del inspector.

- Haz clic con el derecho sobre la zona de “*otros componentes*” y activa la opción *Agregar desde Paleta – Swing – JFileChooser*:



- Aparecerá entonces un objeto JFileChooser dentro de *Otros Componentes*. Aprovecha para cambiarle el nombre a este objeto. Su nombre será *elegirFichero*.

El *inspector* quedará así:



- Una vez hecho esto, ya podemos programar la opción Abrir del menú. Activa el evento *actionPerformed* de la opción “Abrir” y programa dentro de él lo siguiente:

```
int resp;

resp=elegirFichero.showOpenDialog(this);

if (resp==JFileChooser.APPROVE_OPTION) {
    JOptionPane.showMessageDialog(null,elegirFichero.getSelectedFile(
        ).toString());
} else if (resp==JFileChooser.CANCEL_OPTION) {
    JOptionPane.showMessageDialog(null,"Se pulsó la opción Cancelar");
}
```

- Ejecuta el código y prueba la opción “Abrir” del menú. Prueba a elegir algún fichero y abrirlo. Prueba a cancelar la ventana de apertura. Etc
- Analicemos el código anterior:

```
int resp;

resp=elegirFichero.showOpenDialog(this);
```

75. Estas dos líneas crean una variable entera *resp* (respuesta) y a continuación hacen que se muestre la ventana “Abrir Fichero”. Observa que para conseguirlo hay que usar el método *showOpenDialog* del objeto *elegirFichero*. Este método lleva como parámetro la ventana actual (*this*)

76. El método *showOpenDialog* no solo muestra la ventana “Abrir Fichero” sino que también devuelve un valor entero según el botón pulsado por el usuario en esta ventana. Esto es: botón “Abrir” o botón “Calcelar”.

77. Se pueden usar dos *if* para controlar lo que sucede si el usuario pulsó el botón “Abrir” o el botón “Calcelar” de la ventana “Abrir Fichero”:

```
if (resp==JFileChooser.APPROVE_OPTION) {
    JOptionPane.showMessageDialog(null,elegirFichero.getSelectedFile(
        ).toString());
} else if (resp==JFileChooser.CANCEL_OPTION) {
    JOptionPane.showMessageDialog(null,"Se pulsó la opción Cancelar");
}
```

78. En el primer *if* se compara la variable *resp* con la constante *JFileChooser.APPROVE_OPTION*, para saber si el usuario pulsó “Abrir”.

79. En el segundo *if* se compara la variable *resp* con la constante *JFileChooser.CANCEL_OPTION*, para saber si el usuario pulsó “Calcelar”.

80. En el caso de que el usuario pulsara “Abrir”, el programa usa el método *getSelectedFile* del objeto *elegirFichero* para recoger el camino del fichero elegido. Este camino debe ser convertido a cadena con el método *toString*.
81. El programa aprovecha esto para mostrar dicho camino en pantalla gracias al típico *JOptionPane*.
82. En el caso del que el usuario pulsara el botón “Cancelar” el programa muestra un mensaje indicándolo.
- Hay que volver a dejar claro que el cuadro de diálogo “Abrir” realmente no abre ningún fichero, sino que devuelve el camino del fichero elegido usando el código:

```
elegirFichero.getSelectedFile().toString()
```

Luego queda en manos del programador el trabajar con el fichero correspondiente de la forma que desee.

CONCLUSIÓN

Los objetos *JFileChooser* permiten mostrar el cuadro de diálogo “Abrir Fichero” o “Guardar Fichero”.

Estos objetos no abren ni guardan ficheros, solo permiten al usuario elegir el fichero a abrir o guardar de forma sencilla.

El *JFileChooser* devuelve el camino del fichero elegido, y luego el programador trabajará con dicho fichero como mejor le interese.

EJERCICIO GUIADO. JAVA: PANELES DE DESPLAZAMIENTO

Paneles de Desplazamiento

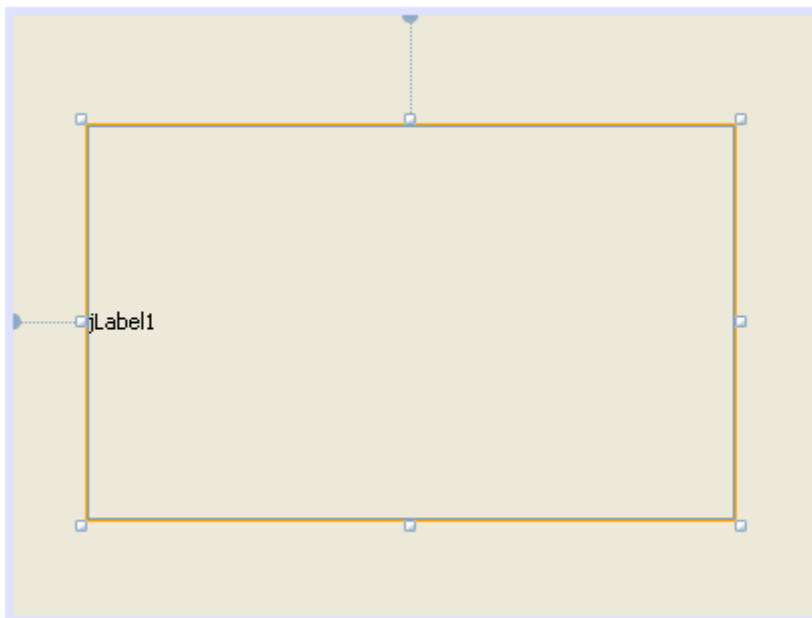
Llamaremos paneles de desplazamiento a paneles que contienen elementos tan grandes que no pueden ser mostrados en su totalidad. Estos paneles contienen entonces dos barras de desplazamiento que permiten visualizar el interior del panel de desplazamiento correctamente.

Por ejemplo, un panel de desplazamiento podría contener una imagen tan grande que no se viera entera:

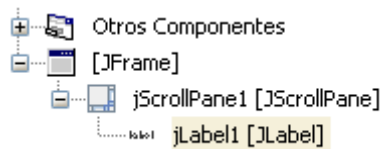
Los paneles de desplazamiento son objetos del tipo JScrollPane.

Ejercicio guiado 1

- Vamos a practicar con los JScrollPane. Para ello, crea un nuevo proyecto.
- Añade en el proyecto un JScrollPane.
- Un JScrollPane, por sí mismo, no contiene nada. Es necesario añadir dentro de él el objeto que contendrá. Para nuestro ejemplo añadiremos dentro de él una etiqueta (JLabel)
- El formulario debe tener ahora este aspecto:

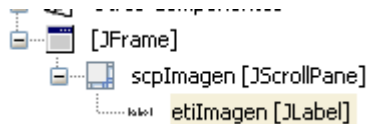


- Si observas el *Inspector* verás claramente la distribución de los objetos:



Observa como tienes un JScrollPane que contiene una etiqueta.

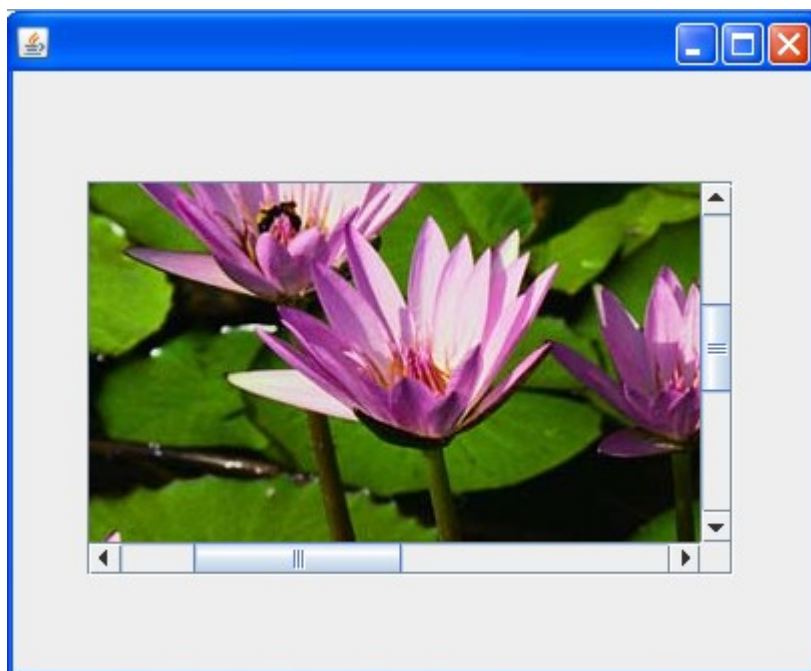
- Aprovechemos el *Inspector* para cambiar el nombre a cada objeto. Al JScrollPane le llamaremos *scpImagen* y a la etiqueta *etiImagen*.



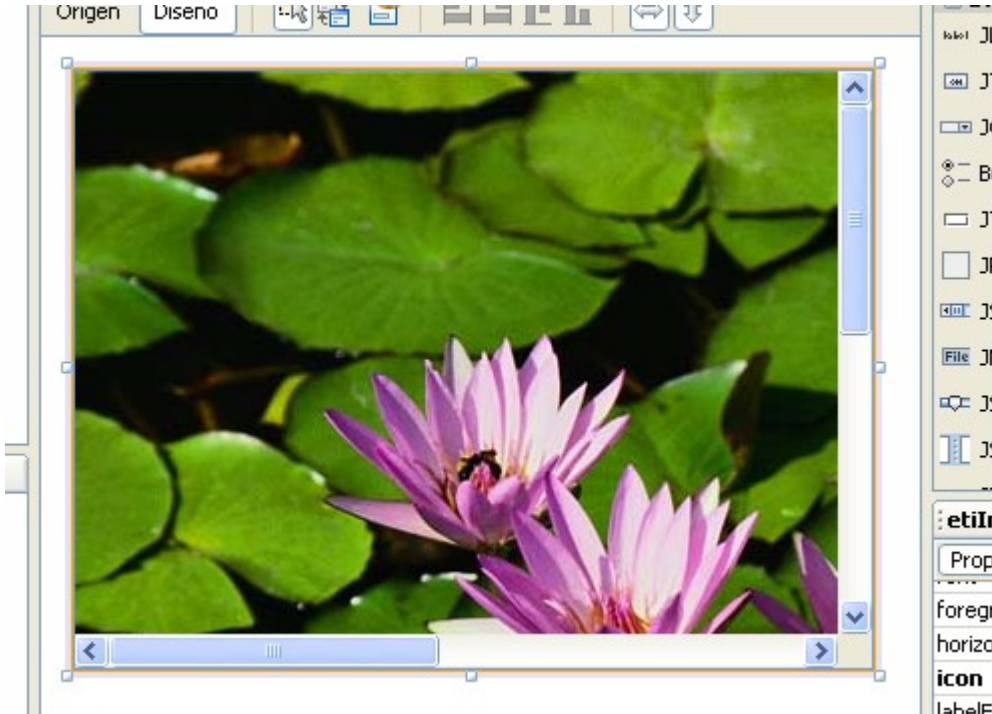
- Elimina el texto contenido en la etiqueta *etiImagen*. Solo tienes que borrar el contenido de la propiedad *text*.
- Luego introduciremos una imagen dentro de la etiqueta, a través de la propiedad *icon*. La imagen la introduciremos desde fichero, y elegiremos la siguiente imagen de tu disco duro:

Mis Documentos / Mis Imágenes / Imágenes de Muestra / Nenúfares.jpg

- Esta imagen es tan grande que no se podrá ver entera dentro del panel de desplazamiento. Ejecuta el programa y observarás el uso de las barras de desplazamiento dentro del panel.



- Puedes mejorar el programa si agrandas el panel de desplazamiento de forma que ocupe todo el formulario:



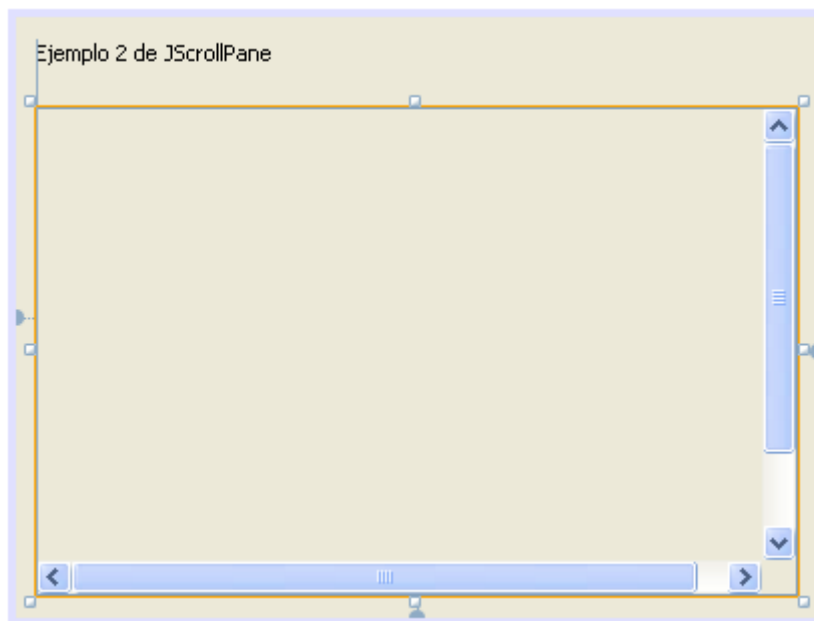
De esta forma, cuando ejecutes el programa, al agrandar la ventana, se agrandará el panel de desplazamiento, viéndose mejor la imagen contenida.

- Ejecuta el programa y compruébalo.

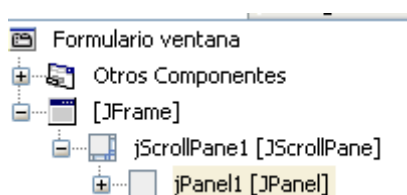
Ejercicio guiado 2

Los JScrollPane no solo están diseñados para contener imágenes. Pueden contener cualquier otro elemento. Vamos a ver, con otro proyecto de ejemplo, otro uso de los JScrollPane.

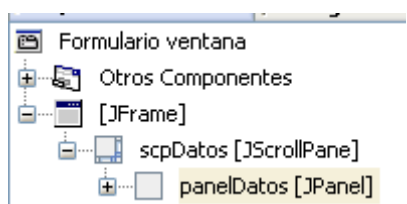
1. Crea un nuevo proyecto.
2. Añade a la ventana una etiqueta con el texto "Ejemplo 2 de JScrollPane" y un JScrollPane de forma que esté asociado a los límites de la ventana. Observa la imagen:



3. Ahora añada dentro del JScrollPane un panel normal (JPanel). En la ventana no notarás ninguna diferencia, pero en el *Inspector* debería aparecer esto:



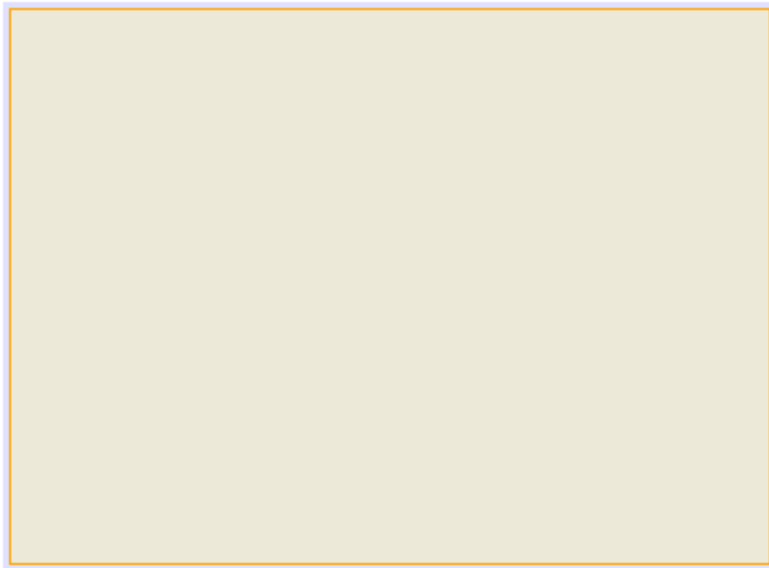
4. Como ves, el JScrollPane contiene a un objeto JPanel.
5. Aprovechemos para cambiar el nombre a ambos objetos. Al JScrollPane lo llamaremos *scpDatos* y al JPanel lo llamaremos *panelDatos*.



6. Los JPanel son objetos contenedores. Es decir, pueden contener otros objetos como por ejemplo botones, etiquetas, cuadros de texto, etc.

Además, los JPanel pueden ser diseñados independientemente de la ventana. Haz doble clic sobre el *panelDatos* en el *Inspector* y observa lo que ocurre:

7. En la pantalla aparecerá únicamente el JPanel, para que puede ser diseñado aparte de la ventana completa:



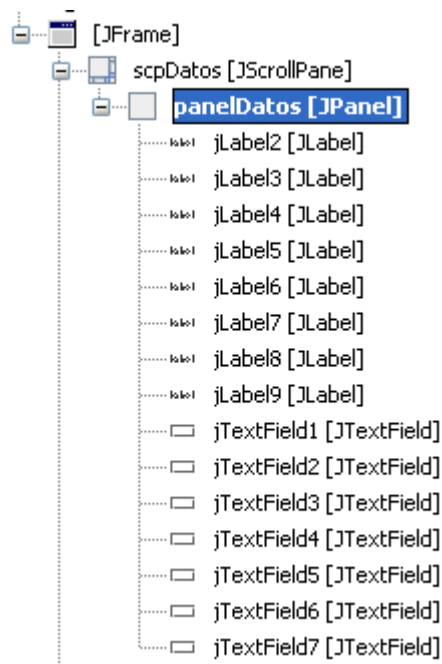
8. Para distinguirlo de lo que es en sí la ventana, haremos las siguientes cosas con el panel:
 123. Cambia el color de fondo y asígnale un color verde.
 124. Añade en él una etiqueta con el texto "Panel de Datos".
 125. Añade varias etiquetas y cuadros de textos correspondientes a los días de la semana.
 126. Agranda el panel.

El panel debería quedar así. Toma como referencia esta imagen:

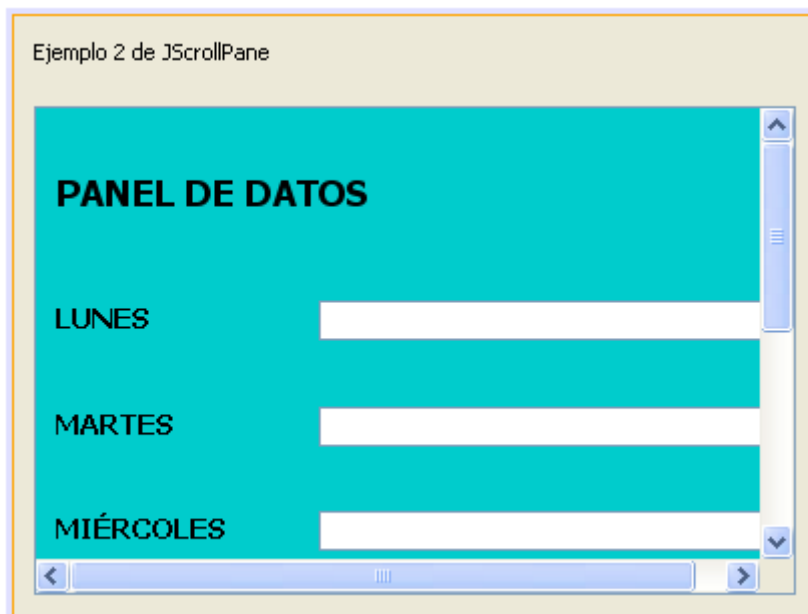
PANEL DE DATOS

LUNES	<input type="text"/>
MARTES	<input type="text"/>
MIÉRCOLES	<input type="text"/>
JUEVES	<input type="text"/>
VIERNES	<input type="text"/>
SÁBADO	<input type="text"/>
DOMINGO	<input type="text"/>

Es muy interesante que observes el *Inspector*. En él podrás observar la distribución de los objetos en la ventana. Podrás ver como el JFrame contiene un JScrollPane (scpDatos) que a su vez contiene un JPanel (panelDatos) que a su vez contiene una serie de etiquetas y cuadros de textos a los que aún no les hemos asignado un nombre:



9. Haz doble clic sobre el JFrame (en el *Inspector*) para poder ver globalmente la ventana. En la pantalla debería aparecer esto:



Como ves, el JPanel contenido en el JScrollPane es más grande que él, por lo que no se podrá visualizar completamente. Será necesario usar las barras de desplazamiento del JScrollPane.

10. Ejecuta el programa para entender esto último.

CONCLUSIÓN

Los objetos JScrollPane son paneles de desplazamiento. Estos paneles pueden contener objetos mayores que el propio panel de desplazamiento. Cuando esto sucede, el panel muestra barras de desplazamiento para poder visualizar todo el contenido del panel.

Los JScrollPane son ideales para mostrar imágenes, paneles y otros elementos cuyo tamaño pueda ser mayor que la propia ventana.

EJERCICIO GUIADO. JAVA: VARIABLES GLOBALES

Variables Globales / Propiedades de la Clase

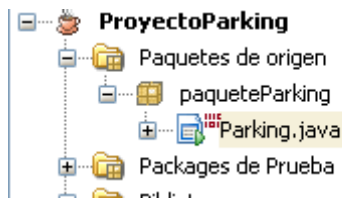
Las propiedades de la clase en java es el equivalente a las variables globales en lenguajes estructurados como el C.

Una propiedad es una variable que puede ser accedida desde cualquier evento programado. Esta variable se inicializa a un valor cuando se ejecuta el programa y los distintos eventos pueden ir cambiando su valor según se necesite.

Veamos un ejemplo para entender el funcionamiento de las propiedades de la clase / variables globales.

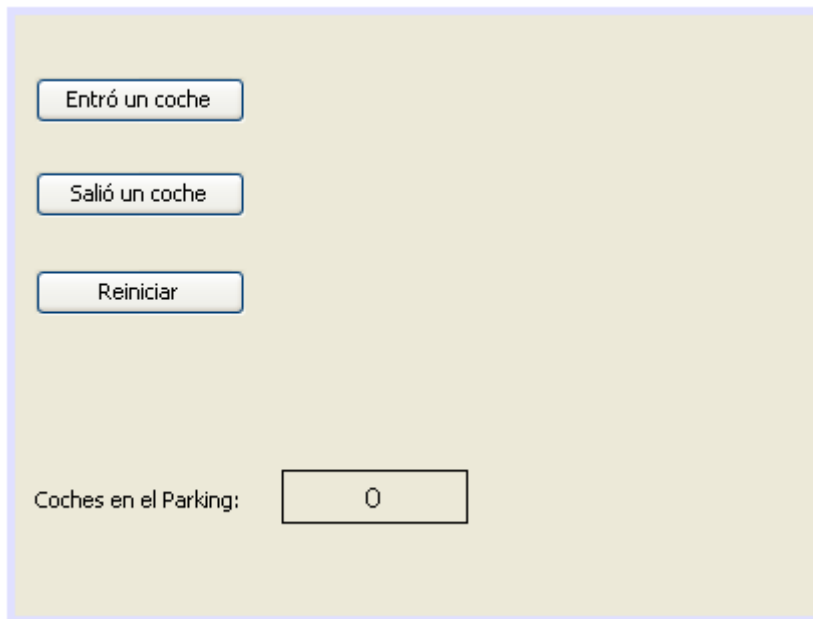
Ejercicio guiado 1

- Crea un nuevo proyecto llamado *ProyectoParking*. Dentro de él añade un paquete llamado *paqueteParking*. Y finalmente añade un JFrame llamado *Parking*. El aspecto de tu proyecto será el siguiente:



Clase Principal

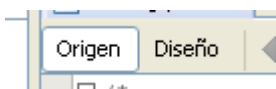
- Tu clase principal es la clase *Parking*.
- Se pretende hacer un pequeño programa que controle los coches que van entrando y van saliendo de un parking. En todo momento el programa debe decir cuantos coches hay dentro del parking. Para ello debes crear una ventana como la que sigue:



- Esta ventana contiene lo siguiente:
 - Un botón “Entró un coche” llamado btnEntro.
 - Un botón “Salió un coche” llamado btnSalio.
 - Un botón “Reiniciar” llamado btnReiniciar.
 - Una etiqueta con el texto “Coches en el Parking”.
 - Una etiqueta con borde llamada etiCoches que contenga un “0”.
- Se pretende que el programa funcione de la siguiente forma:
 - o Cuando el usuario pulse el botón “Entró un coche”, el número de coches del parking aumenta en 1, mostrándose en la etiqueta etiCoches.
 - o Si el usuario pulsa el botón “Salió un coche”, el número de coches del parking disminuye en 1, mostrándose en la etiqueta etiCoches.
 - o El botón Reiniciar coloca el número de coches del parking a 0.
- Para poder controlar el número de coches en el Parking, es necesario que nuestra clase principal *Parking* tenga una propiedad (variable global) a la que llamaremos *coches*. Esta variable será entera (int).

Esta variable contendrá en todo momento los coches que hay actualmente en el *Parking*. Esta variable podrá ser usada desde cualquier evento.

- Para crear una variable global haz clic en el botón “Origen” para acceder al código:



- Luego busca, al comienzo del código una línea que comenzará por

```
public class
```

Seguida del nombre de tu clase principal “*Parking*”.

Debajo de dicha línea es donde se programarán las propiedades de la clase (las variables globales)

```
/**
 *
 * @author didact
 */
public class Parking extends javax.swing.JFrame {

    /**
     * Creates new form Parking
     */
    public Parking() {
        initComponents();
    }
}
```

Aquí se declaran las variables globales, también llamadas propiedades de la clase.

- En dicho lugar declararás la variable *coches* de tipo int:

```
public class Parking extends javax.swing.JFrame {

    int coches;

    /**
     * Creates new form Parking
     */
    public Parking() {
```

Declaración de una variable global int llamada coches.

- Cuando el programa arranque, será necesario que la variable global *coches* tenga un valor inicial. O dicho de otra forma, será necesario inicializar la variable. Para inicializar la variable iremos al constructor. Añade lo siguiente al código:

```
public class Parking extends javax.swing.JFrame {

    int coches;

    /**
     * Creates new form Parking
     */
    public Parking() {
        initComponents();
        coches=0;
    }

    /** This method is called from within the construct
```

Inicialización de la propiedad coches.

Inicializamos a cero ya que se supone que cuando arranca el programa no hay ningún coche dentro del parking.

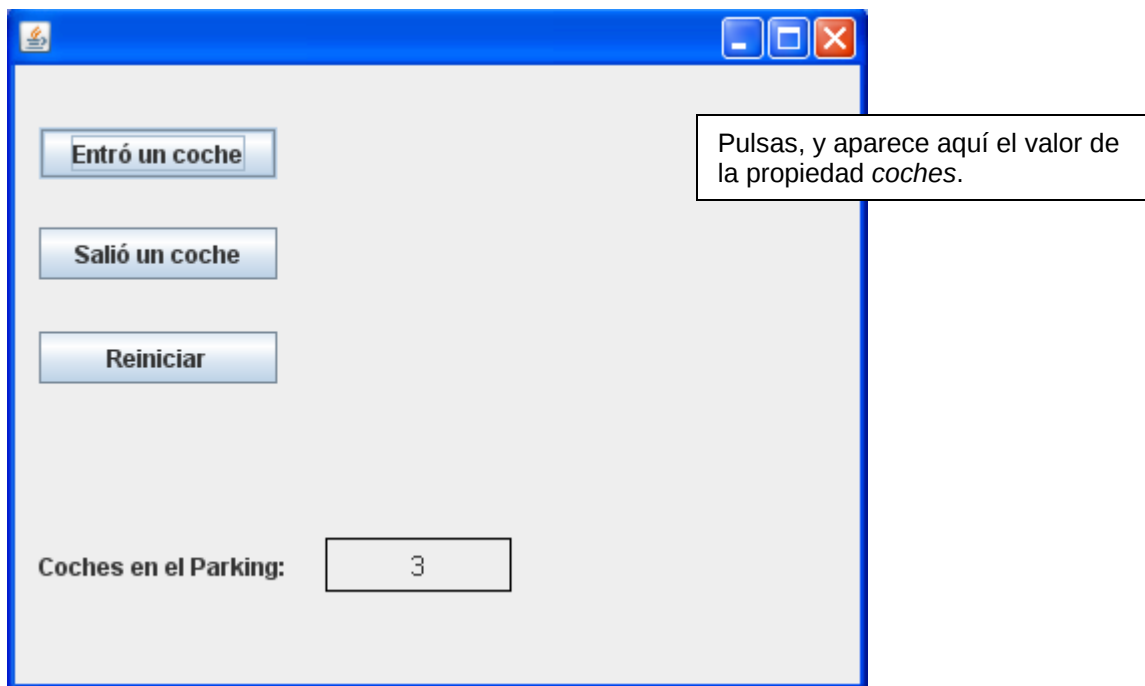
- Ahora que ya tenemos declarada e inicializada la variable global *coches*, esta puede ser usada sin problemas desde cualquier evento que programemos.

Por ejemplo, empezaremos programando la pulsación del botón “Entró un coche”. Acceda a su evento *actionPerformed* y programe esto:

```
coches=coches+1;  
etiCoches.setText(""+coches);
```

Como ves, se le añade a la variable *coches* uno más y luego se coloca su valor actual en la etiqueta.

- Ejecuta el programa y prueba este botón varias veces.



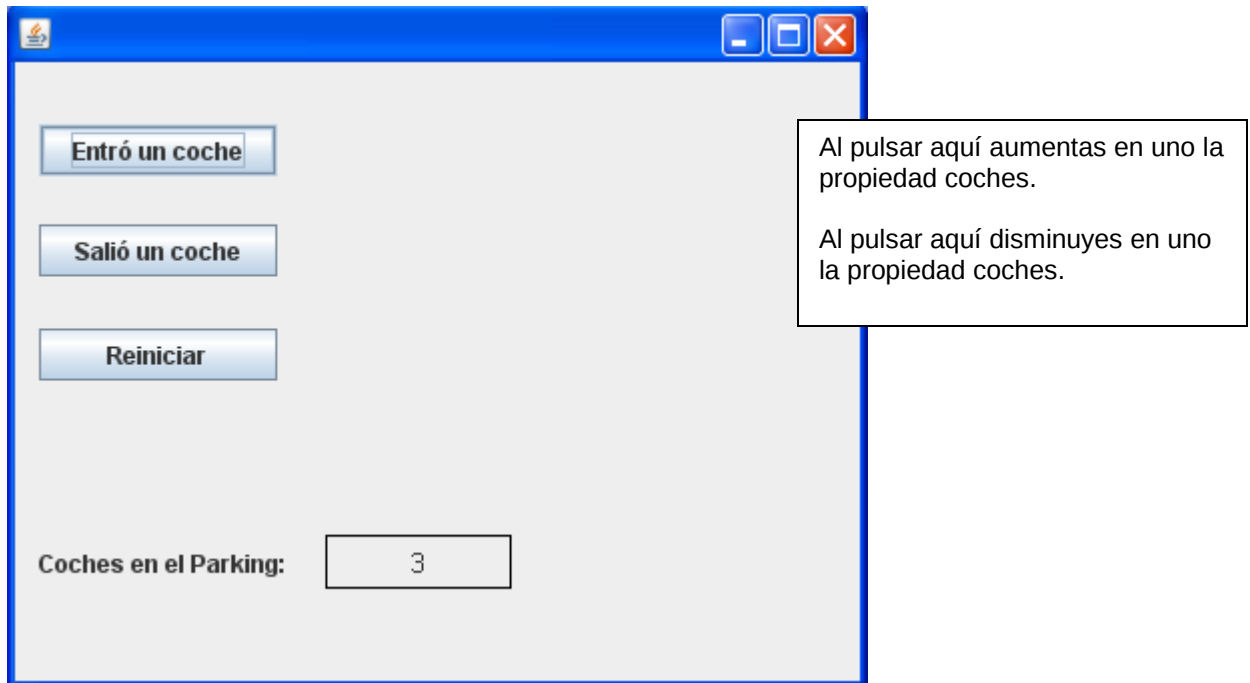
- Ahora programaremos el botón “Salió un coche” de la siguiente forma:

```
if (coches>0) {  
    coches=coches-1;  
    etiCoches.setText(""+coches);  
}
```

Como ves, se accede igualmente a la propiedad *coches* pero esta vez para restarle una unidad. Luego se muestra el valor actual de *coches* en la etiqueta correspondiente.

Se usa un if para controlar que no pueda restarse un coche cuando el parking esté vacío. (Si hay cero coches en el parking no se puede restar uno)

- Ejecuta el programa y prueba los dos botones. Observa como la cantidad de coches del parking aumenta o disminuye.



Lo realmente interesante de esto es que desde dos eventos distintos (desde dos botones) se puede usar la variable *coches*. Esto es así gracias a que ha sido creada como variable global, o dicho de otro modo, ha sido creada como propiedad de la clase *Parking*.

- Finalmente se programará el botón *Reiniciar*. Este botón, al ser pulsado, debe colocar la propiedad *coches* a cero. Programa dentro de su *actionPerformed* lo siguiente:

```
coches=0;  
etiCoches.setText("0");
```

Simplemente introduzco el valor cero en la variable global y actualizo la etiqueta.

- Ejecuta el programa y comprueba el funcionamiento de este botón.

CONCLUSIÓN

Las variables globales, también llamadas propiedades de la clase, son variables que pueden ser usadas desde cualquier evento del programa. Estas variables mantienen su valor hasta que otro evento lo modifique.

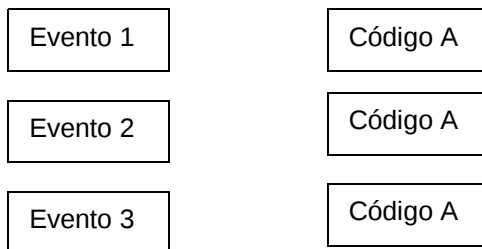
Las variables globales se declaran justo después de la línea *public class*.

La inicialización de estas variables se realiza en el constructor.

EJERCICIO GUIADO. JAVA: CENTRALIZAR CÓDIGO

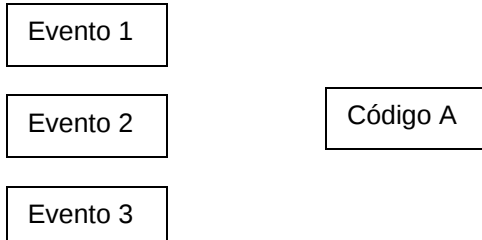
El problema de la repetición de código

Es muy habitual en Java que varios eventos tengan que ejecutar el mismo código. En este caso se plantea la necesidad de “copiar y pegar” ese código en los distintos eventos a programar:



Esta es una mala forma de programación, ya que se necesitara modificar el código, sería necesario realizar la modificación en cada copia del código. Es muy fácil que haya olvidos y aparezcan errores en el programa que luego son muy difíciles de localizar.

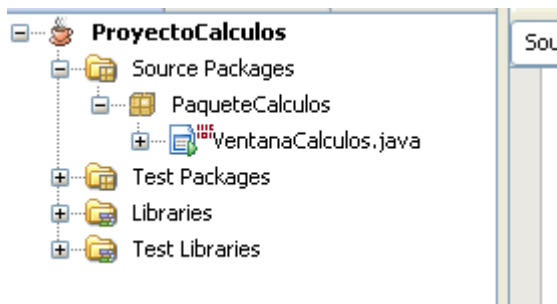
Lo mejor es que el código que tenga que ser ejecutado desde distintos eventos aparezca solo una vez, y sea llamado desde cada evento:



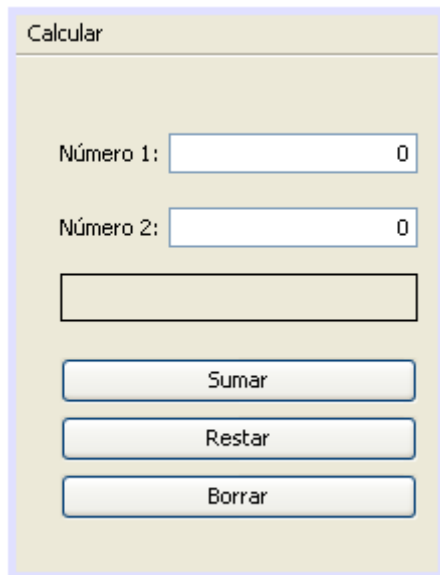
Veamos algunos ejemplos en los que el código se puede repetir y como evitar esta repetición.

Ejercicio guiado 1

- Crea un nuevo proyecto en java que se llame *ProyectoCalculos*. Este proyecto tendrá un paquete llamado *PaqueteCalculos*. Y dentro de él creará un JFrame llamado *VentanaCalculos*. El proyecto tendrá el siguiente aspecto:

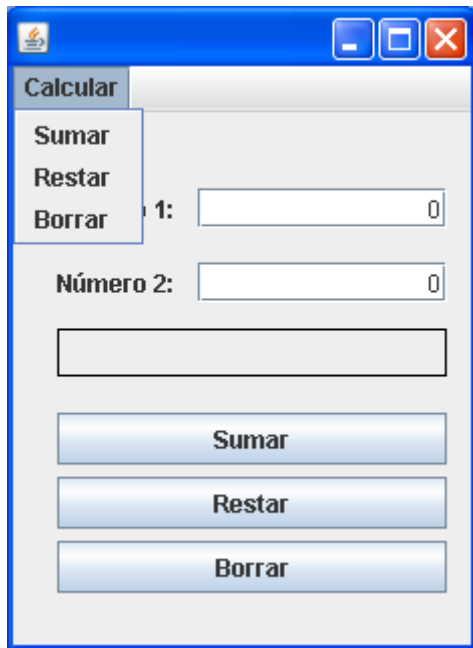


- La *VentanaCalculos* debe estar diseñada de la siguiente forma:



Esta ventana contiene los siguientes elementos:

- Una barra de menús a la que puede llamar *menuBarra*.
 - La barra de menús contiene un JMenu con el texto "Calcular" y que se puede llamar *menuCalcular*
 - El *menuCalcular* contendrá tres JMenuItem, llamados respectivamente: *menuSumar*, *menuRestar*, *menuBorrar* y con los textos "Sumar", "Restar" y "Borrar".
 - Una etiqueta con el texto "Número 1". (no importa su nombre)
 - Una etiqueta con el texto "Número 2". (no importa su nombre)
 - Un cuadro de texto con un 0 y con el nombre *txtNumero1*.
 - Un cuadro de texto con un 0 y con el nombre *txtNumero2*.
 - Una etiqueta con el nombre *etiResultado*.
 - Un botón "Sumar" con el nombre *btnSumar*.
 - Un botón "Restar" con el nombre *btnRestar*.
 - Un botón "Borrar" con el nombre *btnBorrar*.
- Aquí puedes ver la ventana en ejecución con el menú "Calcular" desplegado:



- El objetivo de programa es el siguiente:
 - o El usuario introducirá dos números en los cuadros de texto.
 - o Si pulsa el botón Sumar, se calculará la suma.
 - o Si pulsa el botón Restar, se calculará la resta.
 - o Si pulsa el botón Borrar, se borrarán ambos cuadros de texto.
 - o Si elige la opción del menú Calcular-Sumar entonces se calculará la suma.
 - o Si elige la opción del menú Calcular-Restar entonces se calculará la resta.
 - o Si elige la opción del menú Calcular-Borrar entonces se borrarán ambos cuadros de texto.
 - o Si se pulsa enter en alguno de los dos cuadros de texto se debería calcular la suma.
- Este es un ejemplo en el que al activarse uno de varios eventos distintos se tiene que ejecutar el mismo código. Observa el caso de la suma:

Pulsar Botón Sumar

Activar Calcular – Sumar en el menú

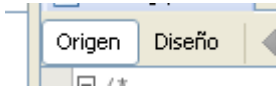
Pulsar enter en el primer cuadro de texto

Pulsar enter en el segundo cuadro de texto

Calcular la suma y mostrarla en la etiqueta de resultado

- Para que el código esté “centralizado”, es decir, que aparezca solo una vez, será necesario construir en la clase un *método*. Un *método* en java es el equivalente de una función o procedimiento en C. Veamos como hacerlo:

- Accede al código de tu programa a través del botón *Origen*.



- Un buen sitio para programar tus procedimientos puede ser debajo del constructor. Puedes distinguir fácilmente al constructor porque tiene el mismo nombre que la clase que estás programando, o dicho de otro modo, tiene el mismo nombre que la ventana que estás programando: *VentanaCalculos*.

```

    /**
    */
    public class VentanaCalculos extends javax.swing.JFrame {

        /**
         * Creates new form VentanaCalculos
         */
        public VentanaCalculos() {
            initComponents();
        }

        /** This method is called from within the constructor to
         * initialize the form.
         * WARNING: Do NOT modify this code. The content of this m

```

Este es el constructor

Este es un buen sitio para crear
tus propios procedimientos

- Se va a programar un procedimiento que se encargue de recoger los valores de los cuadros de texto. Calculará la suma de dichos valores, y luego mostrará la suma en la etiqueta de resultados.

Los procedimientos en java tienen prácticamente la misma estructura que en C. Programe lo siguiente:

```

public class VentanaCalculos extends javax.swing.JFrame {

    /**
     * Creates new form VentanaCalculos
     */
    public VentanaCalculos() {
        initComponents();
    }

    void Sumar() {
        String cad1, cad2;
        int a,b,s;

        cad1 = txtNumerol.getText();
        cad2 = txtNumero2.getText();
        a = Integer.parseInt(cad1);
        b = Integer.parseInt(cad2);
        s=a+b;
        etiResultado.setText(""+s);
    }

    /** This method is called from within the constructor to

```

Este es el procedimiento que tienes que introducir en el programa.

- Si observas el código, es el típico procedimiento de C, cuya cabecera comienza con *void* y el nombre que le hayas asignado (en nuestro caso *Sumar*)

```

void Sumar() {
    ....
}

```

Si estudias las líneas del código, verás que lo que hace es recoger el contenido de los dos cuadros de texto en dos variables de cadena llamadas *cad1* y *cad2*.

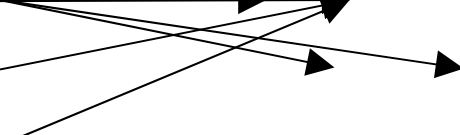
Luego convierte dichas cadenas en números que almacena en dos variables enteras llamadas *a* y *b*.

Finalmente calcula la suma en una variable *s* y presenta el resultado en la etiqueta *etiResultado*.

- Hay que destacar que este código no pertenece ahora mismo a ningún evento en concreto, por lo que no tiene efecto ninguno sobre el programa. Será necesario pues asociar los eventos correspondientes con este procedimiento.
- Interesa que al pulsar el botón "Sumar" se ejecute la suma, así pues entre en el evento *actionPerformed* del botón "Sumar" y añada la siguiente línea:

```
Sumar();
```

- Como también interesa que al pulsar la opción del menú "Calcular-Sumar" se ejecute la suma, entre en el evento *actionPerformed* de la opción del menú "Sumar" y añade de nuevo la siguiente línea:



```
Sumar();
```

- También se quiere que al pulsar la tecla enter en el cuadro de texto del número 1 se ejecute la suma. Por lo tanto, en el evento *actionPerformed* del cuadro de texto txtNumero1 hay que añadir la siguiente línea:

```
Sumar();
```

- Y como también se quiere que al pulsar la tecla enter en el cuadro de texto del número 2 se ejecute la suma, también habrá que introducir en su *actionPerformed* la siguiente línea:

```
Sumar();
```

- Antes de continuar, ejecute el programa, introduzca dos números, y compruebe como se calcula la suma al pulsar el botón Sumar, o al activar la opción del menú Calcular–Sumar, o al pulsar Enter en el primer cuadro de texto, o al pulsar Enter en el segundo cuadro de texto.

En cada uno de los eventos hay una llamada al procedimiento *Sumar*, que es el que se encarga de realizar la suma.

```
actionPerformed btnSumar
```

```
actionPerformed menuSumar
```

```
actionPerformed txtNumero1
```

```
actionPerformed txtNumero2
```

Procedimiento

Sumar()

- En el caso de la resta sucede igual. Tenemos que varios eventos distintos deben provocar que se realice una misma operación. En este caso tenemos lo siguiente:

```
Pulsar Botón Restar
```

```
Activar Calcular – Restar en el menú
```

Calcular la resta y
mostrar el resultado.

- Para centralizar el código, crearemos un método *Restar* que se encargará de hacer la resta de los números introducidos en los cuadros de texto. Este método se puede colocar debajo del anterior método *Sumar*:

```

void Sumar() {
    String cad1, cad2;
    int a,b,s;

    cad1 = txtNumerol.getText();
    cad2 = txtNumero2.getText();
    a = Integer.parseInt(cad1);
    b = Integer.parseInt(cad2);
    s=a+b;
    etiResultado.setText(""+s);
}

void Restar() {
    String cad1, cad2;
    int a,b,r;

    cad1 = txtNumerol.getText();
    cad2 = txtNumero2.getText();
    a = Integer.parseInt(cad1);
    b = Integer.parseInt(cad2);
    r=a-b;
    etiResultado.setText(""+r);
}

```

Programa este procedimiento.

- El código de este procedimiento es prácticamente idéntico al del procedimiento Sumar, así que no se comentará.
- Ahora, es necesario que cuando se activen los eventos indicados antes, estos hagan una llamada al procedimiento Restar para que se efectúe la resta. Así pues, entre en el evento *actionPerformed* del botón "Restar" y añada esta línea de código:

Restar();

- Igualmente, entre en el evento *actionPerformed* de la opción del menú "Calcular – Restar" y añada la misma llamada:

Restar();

- Ejecute el programa y compruebe como funciona el cálculo de la resta, da igual que lo haga pulsando el botón "Restar" o la opción del menú "Restar". Ambos eventos llaman al mismo método:

actionPerformed btnRestar

actionPerformed menuRestar

Procedimiento

Restar()

- Finalmente se programará el borrado de los cuadros de texto a través del botón “Borrar” y de la opción del menú “Borrar”. En primer lugar, programa el siguiente método (puedes hacerlo debajo del método “Restar”):

```

}

void Restar() {
    String cad1, cad2;
    int a,b,r;

    cad1 = txtNumerol.getText();
    cad2 = txtNumero2.getText();
    a = Integer.parseInt(cad1);
    b = Integer.parseInt(cad2);
    r=a-b;
    etiResultado.setText(""+r);
}

```

```

void Borrar() {
    txtNumerol.setText("");
    txtNumero2.setText("");
}

```

Programa el
procedimiento Borrar...

- Ahora programa las llamadas al procedimiento borrar desde los distintos eventos. En el evento *actionPerformed* del botón “Borrar” y en el evento *actionPerformed* de la opción del menú “Borrar” programa la siguiente llamada:

```
Borrar();
```

- Ejecuta el programa y prueba su funcionamiento.

CONCLUSIÓN

En java se pueden programar procedimientos al igual que en C. Normalmente, estos procedimientos se programarán debajo del constructor, y tienen la misma estructura que en C.

Se puede llamar a un mismo procedimiento desde distintos eventos, evitando así la repetición de código.

EJERCICIO GUIADO. JAVA: LAYOUTS

El problema de la distribución de elementos en las ventanas

Uno de los problemas que más quebraderos de cabeza da al programador es el diseño de las ventanas y la situación de los distintos componentes en ellas.

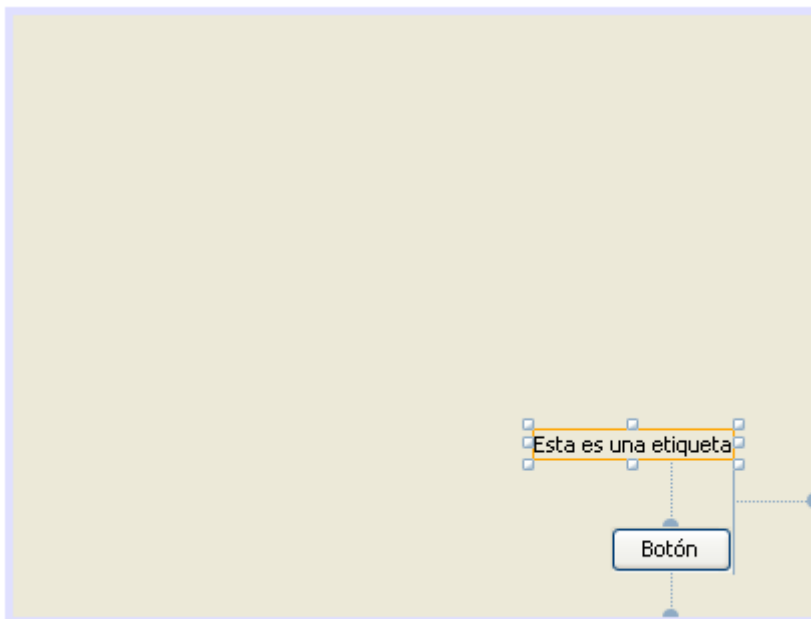
Para diseñar más cómodamente las ventanas, Java proporciona una serie de objetos denominados Layouts, que definen la forma que tendrán los elementos de situarse en las ventanas.

Así pues, un Layout define de qué forma se colocarán las etiquetas, botones, cuadros de textos y demás componentes en la ventana que diseñamos.

Ejercicio guiado

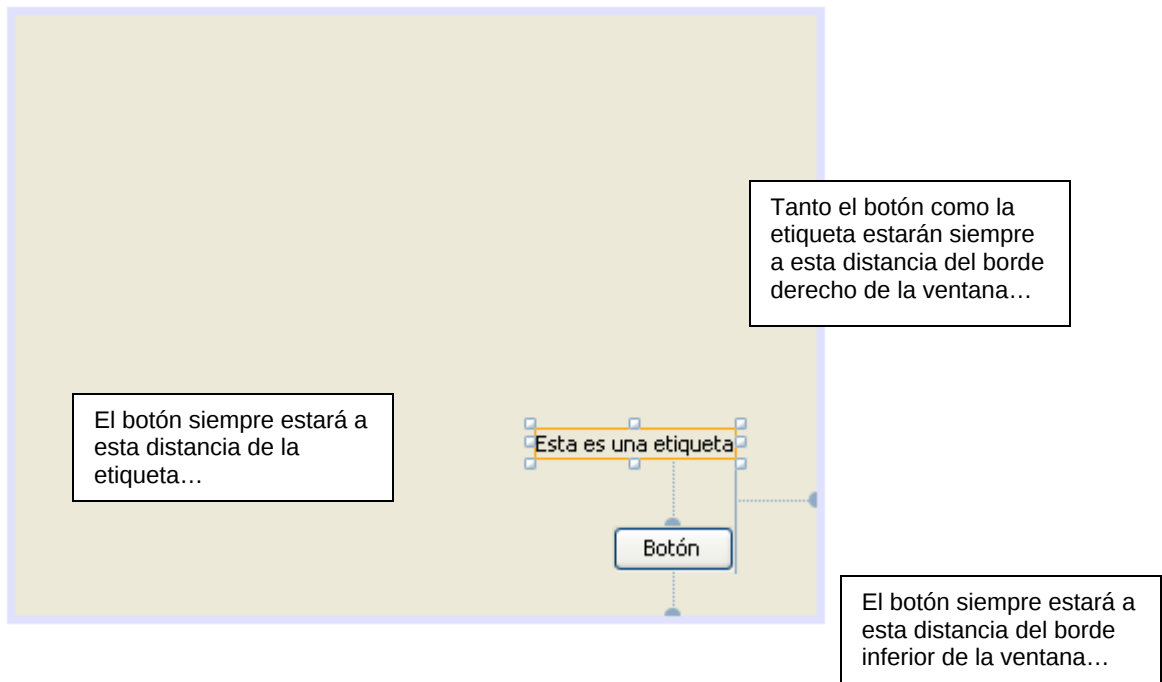
“Diseño Libre”

- Crea un nuevo proyecto en java.
- Añade una etiqueta y un botón. Muévelos a la posición que se indica en la imagen. Deben aparecer las líneas “guía” de color azul que se muestran:

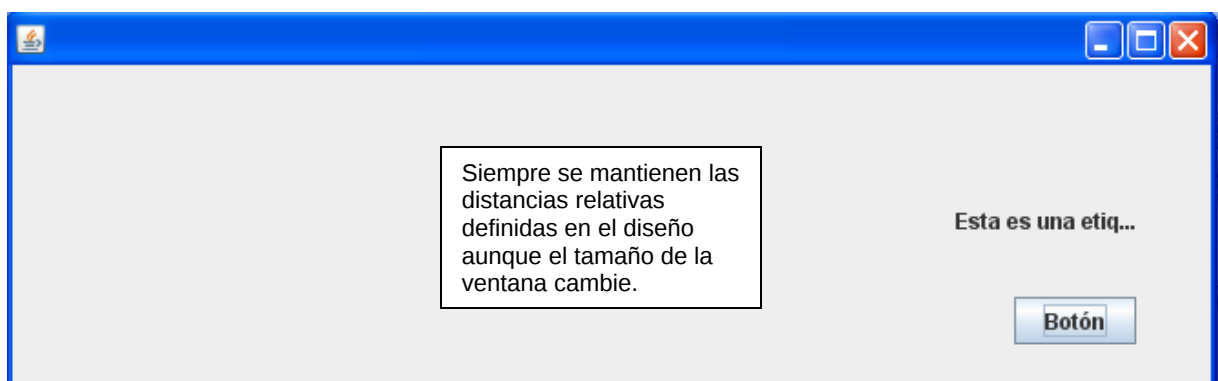


- Las líneas azules que aparecen indican con qué otro elemento está relacionado un componente de la ventana. La situación de un elemento dependerá siempre de la situación del otro.

Dicho de otra forma, las líneas azules indican las distancias que siempre se respetarán. Observa la siguiente imagen:



- Ejecuta el programa y prueba a ensanchar (o achicar) la ventana por el lado derecho y por el lado inferior. Debes observar como la etiqueta y el botón mantienen sus distancias relativas entre sí y con los bordes derecho e inferior de la ventana.



- Este comportamiento de los elementos en la ventana viene dado por una opción del NetBeans llamada *Diseño Libre* (Free Design)

El Diseño Libre permite que los elementos de una ventana mantengan una distribución relativa da igual el tamaño que tenga la ventana. Dicho de otra forma, los elementos se redistribuyen al cambiar el tamaño de la ventana.

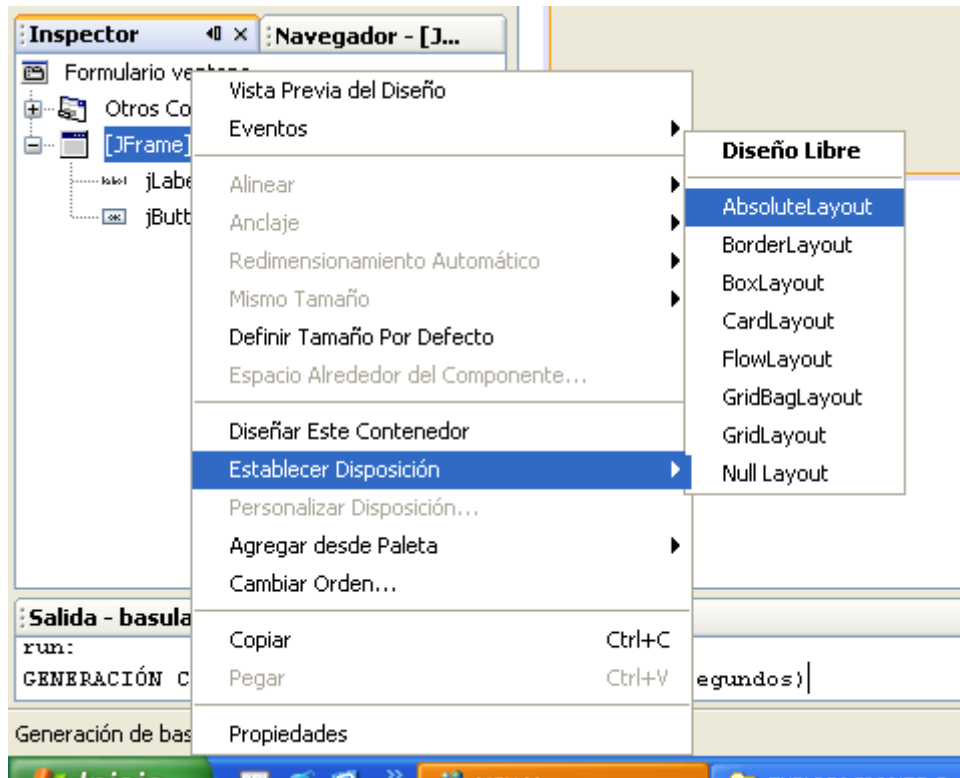
El problema del Diseño Libre es el poco control que se tiene sobre los elementos que se añaden a la ventana.

Se puede observar como a veces los elementos no se colocan en la posición que deseamos o como cambian de tamaño de forma inesperada. Todo esto es debido a la necesidad de dichos elementos de mantener unas distancias relativas con otros

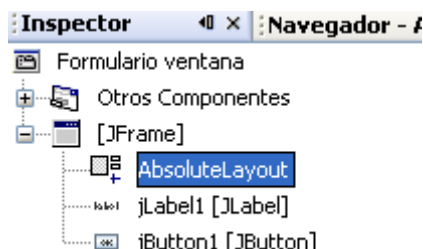
elementos de la ventana. Cuantos más elementos tengamos en una ventana, más difícil será el colocarlos usando el Diseño Libre.

“AbsoluteLayout. Posiciones Absolutas”

- El Diseño Libre es la opción que está activada por defecto cuando se crea un proyecto en NetBeans. Sin embargo, esta opción se puede cambiar por distintos “Layouts” o “Distribuciones”.
- En el *Inspector* de tu proyecto pulsa el botón derecho del ratón sobre el objeto JFrame y activa la opción *Establecer Disposición – AbsoluteLayout*.



- El *Inspector* tendrá la siguiente forma ahora:

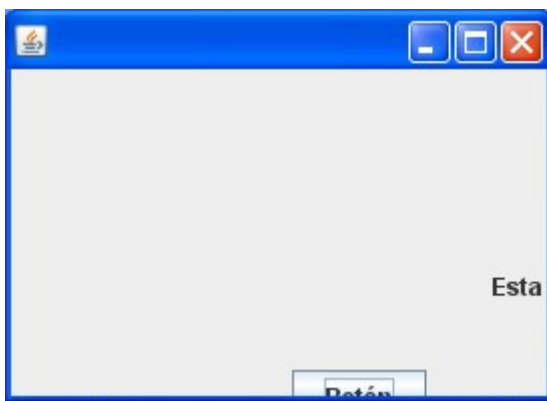


Como ves, aparece un objeto dentro del JFrame llamado AbsoluteLayout. Este objeto define otra forma de situar los elementos en la ventana. Concretamente, la distribución AbsoluteLayout permite al programador colocar cada elemento donde él quiera, sin restricciones, sin tener en cuenta distancias relativas.

- Sitúa la etiqueta y el botón donde quieras. Observa que no aparece ninguna línea guía que defina distancias relativas:



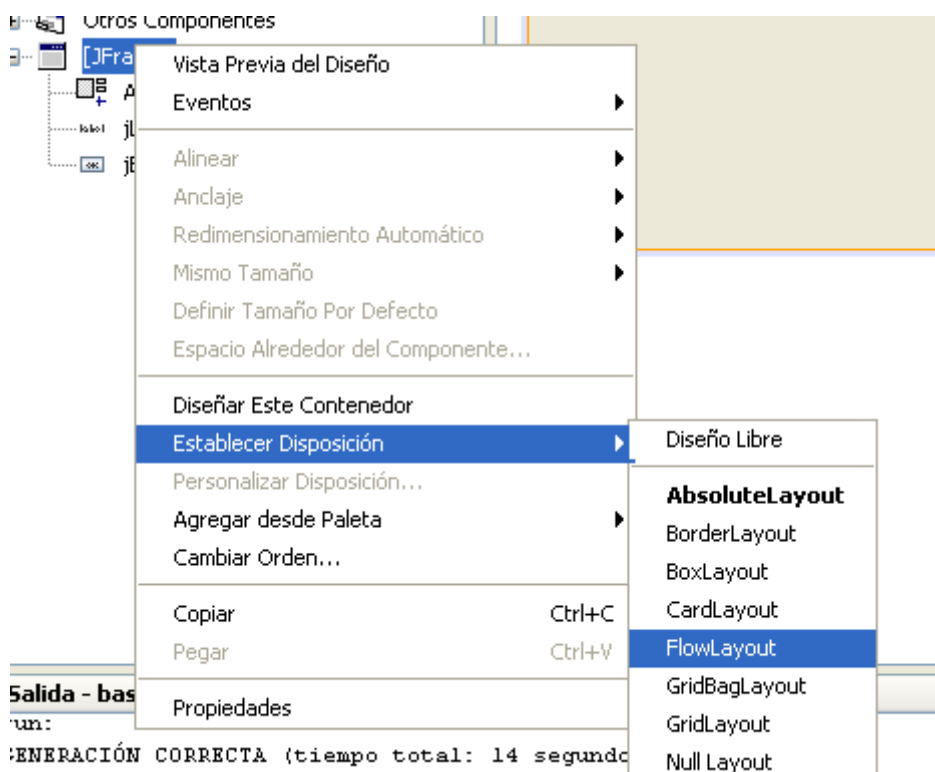
- La ventana de definir una distribución `AbsoluteLayout` es la facilidad para colocar cada elemento en la ventana (no tendrás los problemas del Diseño Libre). Sin embargo, la desventaja es que los elementos no mantienen una distribución relativa respecto al tamaño de la ventana.
- Ejecuta el programa y reduce su ancho. Observa lo que ocurre:



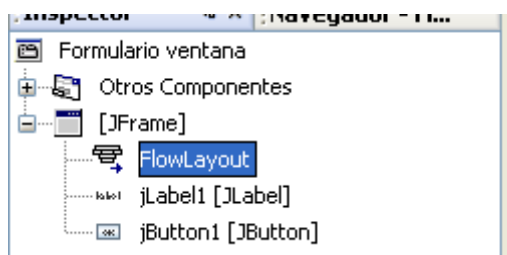
Verás que los elementos de la ventana son inamovibles aunque la ventana cambie de tamaño. En cambio, en el Diseño Libre los elementos intentaban siempre estar dentro de la ventana.

“Distribución en línea. `FlowLayout`”

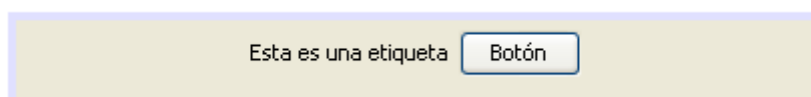
- Practiquemos ahora con otro tipo de distribución. Accede al *Inspector* y pulsa el botón derecho del ratón sobre el objeto JFrame. Activa la opción *Establecer Disposición – FlowLayout*.



- Observa como el layout “AbsoluteLayout” es sustituido por la distribución “FlowLayout”. Una elemento solo puede tener un tipo de distribución a la vez.

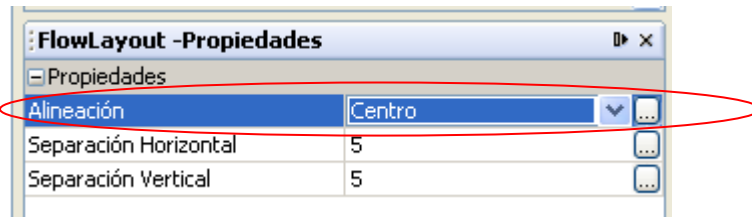


- Observa la ventana. Los elementos se han colocado uno detrás de otro. Se han colocado “en línea”. Esto es lo que hace el “FlowLayout”. Fuerza a los distintos elementos a que se coloquen en fila.

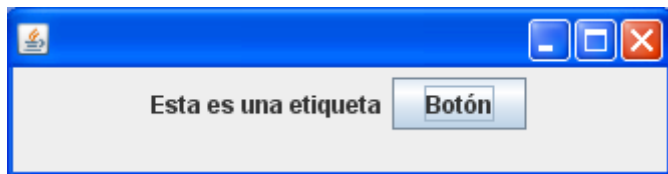


- Si seleccionas el FlowLayout en el *Inspector*, podrás acceder a sus propiedades (los layout son objetos como los demás) Una de las propiedades del FlowLayout se llama

alineación y permite que los elementos estén alineados a la izquierda, derecha o centro. El FlowLayout tiene una alineación centro por defecto.



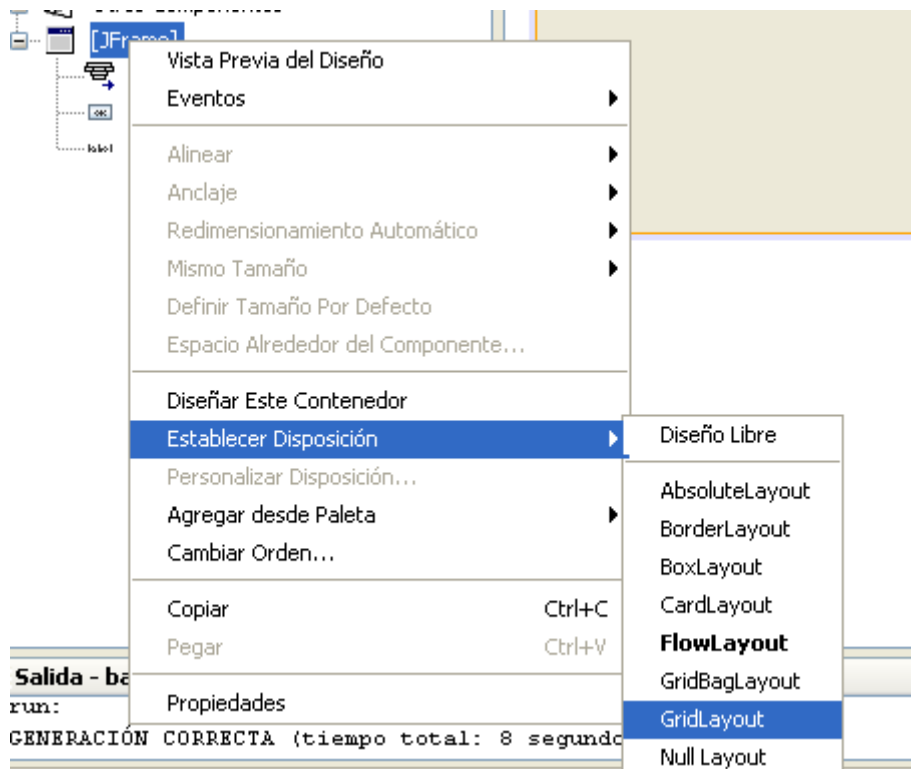
- El FlowLayout no permite controlar la posición de los elementos en la ventana, pero sí procura que los elementos estén siempre visibles aunque la ventana se cambie de tamaño. Ejecuta el programa y observa el comportamiento del FlowLayout al agrandar o achicar la ventana:



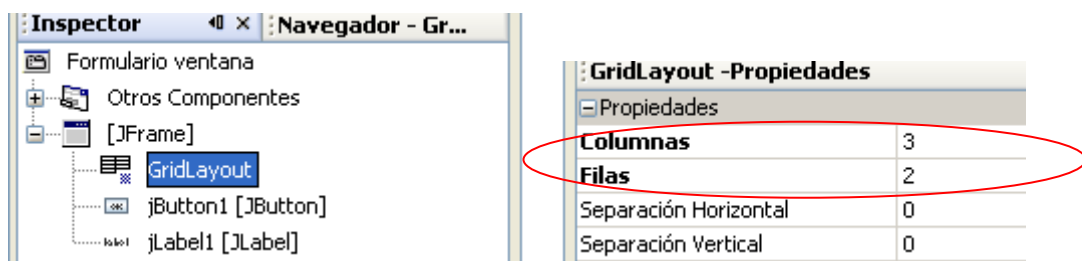
En el FlowLayout, los elementos intentan siempre estar dentro de la ventana, aunque esta se cambie de tamaño...

“Distribución en rejilla. GridLayout”

- Otra distribución que se puede usar es la distribución GridLayout. Esta distribución coloca a los elementos en filas y columnas, como si formaran parte de una tabla. Al añadir esta distribución es necesario indicar cuantas filas y columnas tendrá la rejilla.
- Cambia el layout del JFrame por un GridLayout:



- Marca el GridLayout y cambia sus propiedades Filas y Columnas. Asigna a la propiedad Filas un 2 y a la propiedad Columnas un 3.



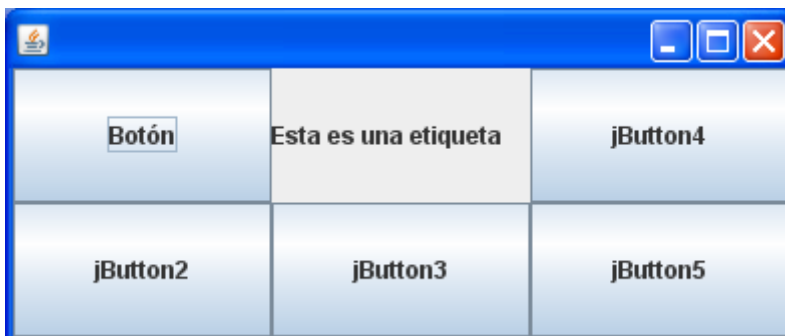
- Al asignar 2 filas y 3 columnas al GridLayout, conseguiremos que los elementos de la ventana se distribuyan en una tabla como la siguiente:

Los distintos elementos se adaptarán al espacio que tienen asignado, y cambiarán de tamaño.

- Ya que solo tienes dos elementos en la ventana (una etiqueta y un botón), añade otros cuatro elementos más (cuatro botones) para observar como se distribuyen en la cuadrícula.



- En un GridLayout, los elementos estarán situados siempre en una casilla de la rejilla, ocupando todo su espacio. El programador no tiene mucho control sobre la disposición de los elementos.
- Ejecuta el programa y agranda y achica la ventana. Observa como los elementos siempre mantienen su disposición en rejilla y siempre aparecen dentro de la ventana aunque el tamaño de esta varíe.



Con un GridLayout los elementos aparecen en filas y columnas.

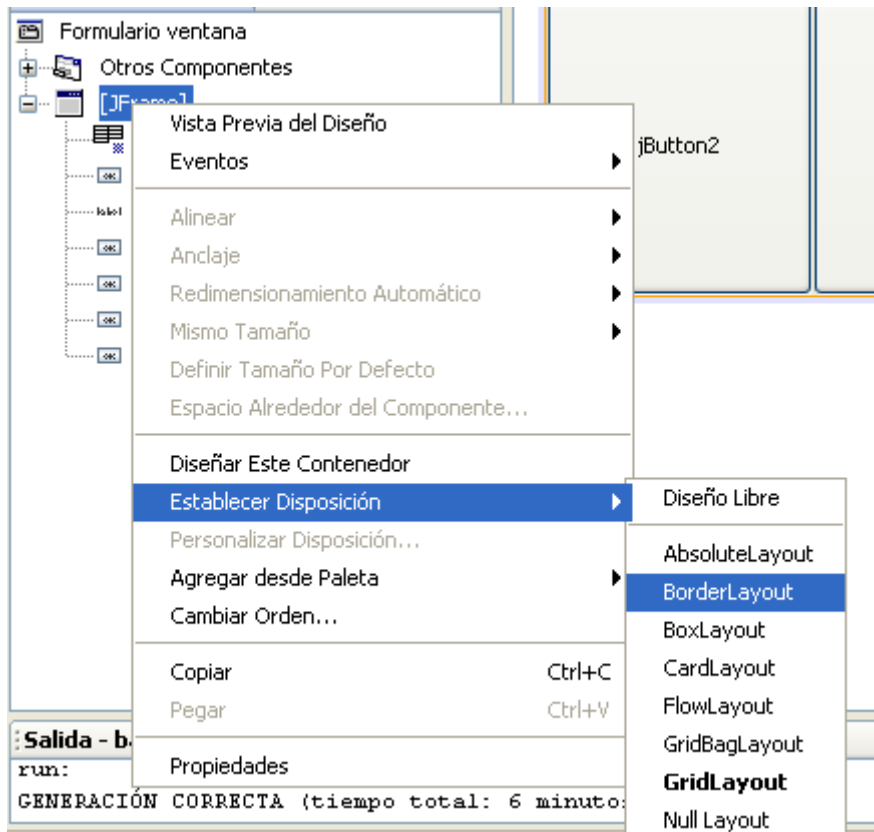
Siempre aparecen dentro de la ventana aunque el tamaño de esta cambie.

“BorderLayout”

- Otra de las distribuciones posibles es la llamada BorderLayout. Esta distribución coloca los elementos de la ventana en cinco zonas:
 - Zona norte (parte superior de la ventana)
 - Zona sur (parte inferior de la ventana)
 - Zona este (parte derecha de la ventana)

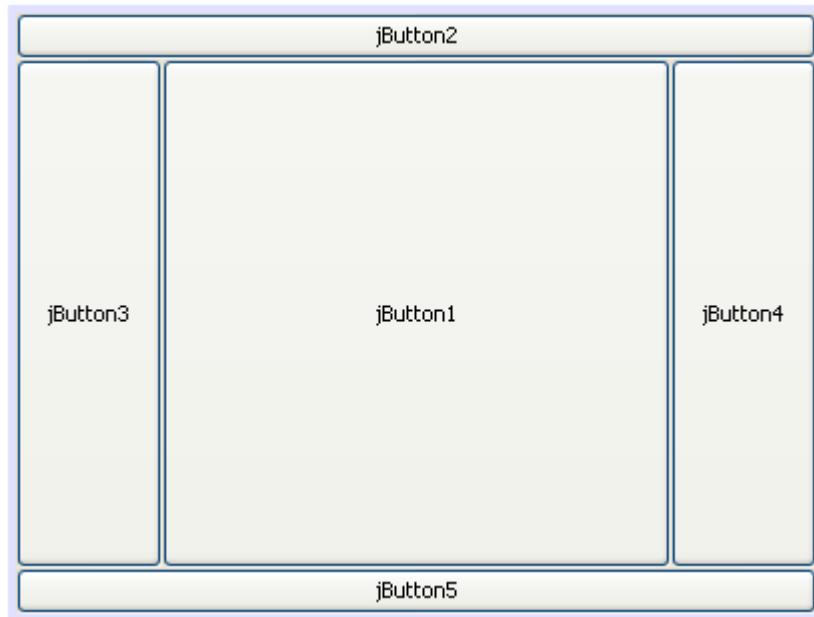
- Zona oeste (parte izquierda de la ventana)
- Zona centro.

- Haz clic con el derecho sobre el JFrame y asigna una distribución “BorderLayout”.



- Para poder entender el funcionamiento del BorderLayout, se recomienda que el JFrame contenga únicamente 5 botones (elimine los elementos que tiene ahora y añada cinco botones)

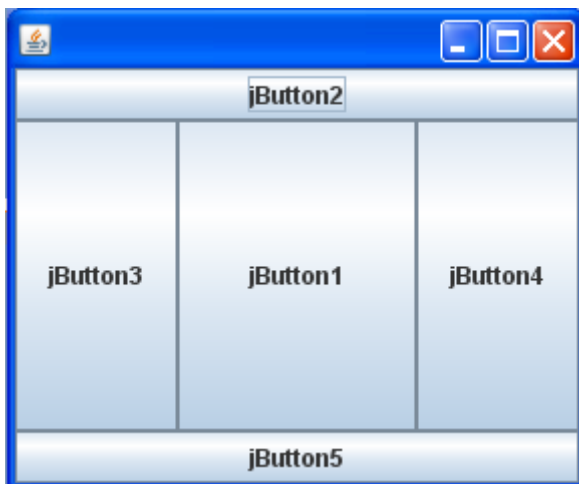
La ventana tendrá un aspecto parecido al siguiente:



- Como se puede observar, cada botón se ha colocado en una zona, y su tamaño ha variado hasta ocupar la zona entera. Tenemos un botón en el norte, otro al sur, uno al este, otro al oeste y uno en el centro.

El programador no tiene mucho control sobre la disposición de los elementos en la ventana al usar esta distribución.

- Ejecuta el programa y observa como los elementos siempre se mantienen dentro de la ventana aunque esta cambie de tamaño.



Con un GridLayout los elementos aparecen zonas.

Siempre aparecen dentro de la ventana aunque el tamaño de esta cambie.

CONCLUSIÓN

El diseño de la ventana viene definido por los Layouts o distribuciones.

Diseño Libre – Esta distribución viene activada por defecto en el NetBeans, y define una distribución de componentes en la que se respetan las distancias entre ellos cuando la ventana cambia de tamaño.

AbsoluteLayout – En esta distribución el programador puede colocar cada elemento en la posición que desee de la ventana. Los distintos elementos mantienen su posición aunque la ventana cambie de tamaño, lo que puede hacer que si la ventana se reduce de tamaño algunos elementos no se vean.

FlowLayout – En esta distribución los elementos se colocan uno detrás de otro. Los elementos intentarán estar dentro de la ventana aunque esta se reduzca de tamaño.

GridLayout – Esta distribución coloca a los elementos en filas y columnas. Los elementos siempre estarán dentro de la ventana aunque esta se reduzca de tamaño.

BorderLayout – Esta distribución coloca a los elementos en zonas. Los elementos siempre estarán dentro de la ventana aunque esta se reduzca de tamaño.

EJERCICIO GUIADO. JAVA: LAYOUTS Y PANELES

Técnicas de distribución de elementos en las ventanas

A la hora de diseñar una ventana se tienen en cuenta dos cosas:

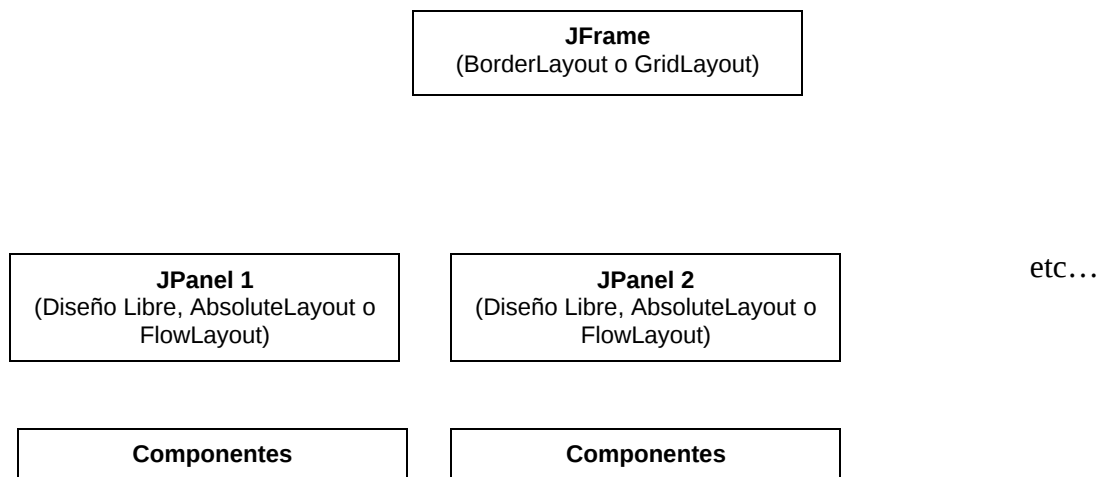
- La facilidad a la hora de colocar muchos componentes en la ventana.
- Que dichos componentes estén siempre visibles independientemente del tamaño de la ventana.

La distribución `AbsoluteLayout` por ejemplo nos da mucha facilidad a la hora de colocar los elementos en la ventana, pero sin embargo los componentes no se adaptan a los cambios de tamaño.

El Diseño Libre en cambio permite crear ventanas en las que sus componentes se “recolocan” según el tamaño de estas pero a cambio crece la dificultad del diseño.

Para aprovechar las ventajas de los distintos layouts y minimizar sus inconvenientes, es habitual en java crear una estructura de paneles cada uno de ellos con un layout distinto, según nuestras necesidades.

Normalmente, al `JFrame` se le asigna un layout que lo divida en zonas, como puede ser el `BorderLayout` o el `GridLayout`. Luego, dentro de cada una de estas zonas se introduce un panel (objeto `JPanel`). Y a cada uno de estos paneles se le asigna el layout que más le convenga al programador (`FlowLayout`, Diseño Libre, `AbsoluteLayout`, etc...) Finalmente, dentro de cada panel se añaden los componentes de la ventana.

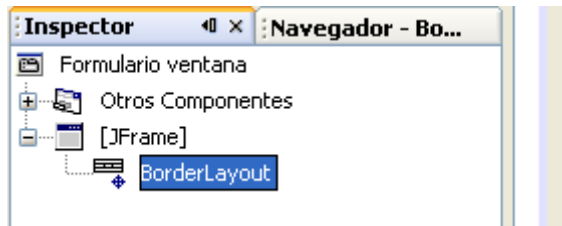


Ejercicio guiado

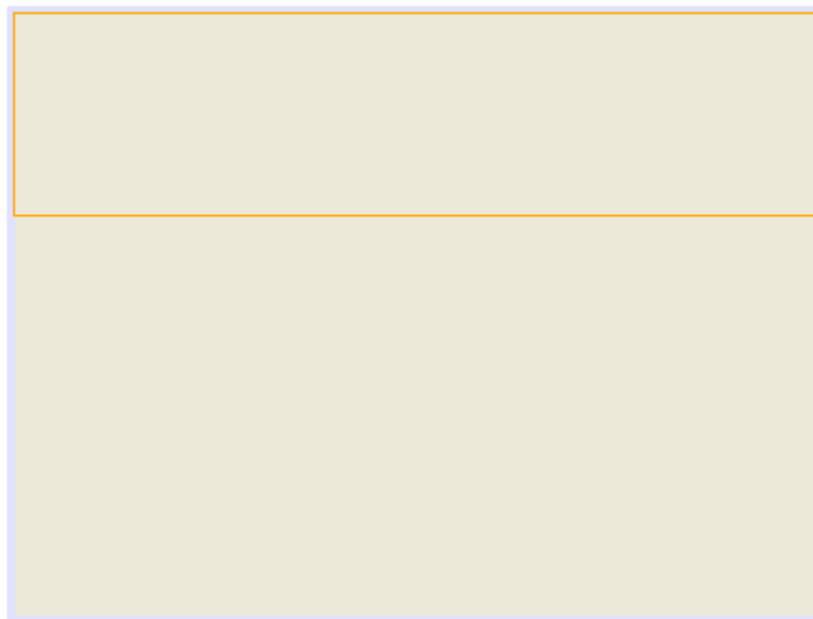
- Crea un nuevo proyecto en java.

Se pretende crear un proyecto con una ventana de diseño complejo. Para ello sigue los siguiente pasos:

- En primer lugar, asigna un BorderLayout al JFrame:

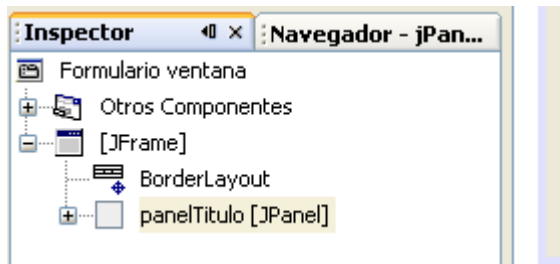


- El BorderLayout divide la ventana principal en zonas. Ahora añade un panel (JPanel) a la zona norte de la ventana.

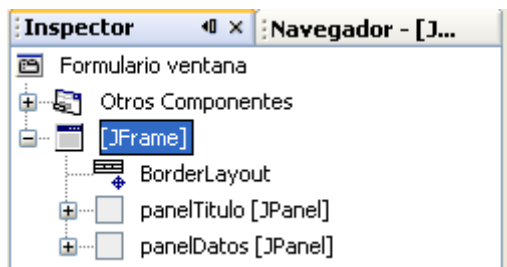
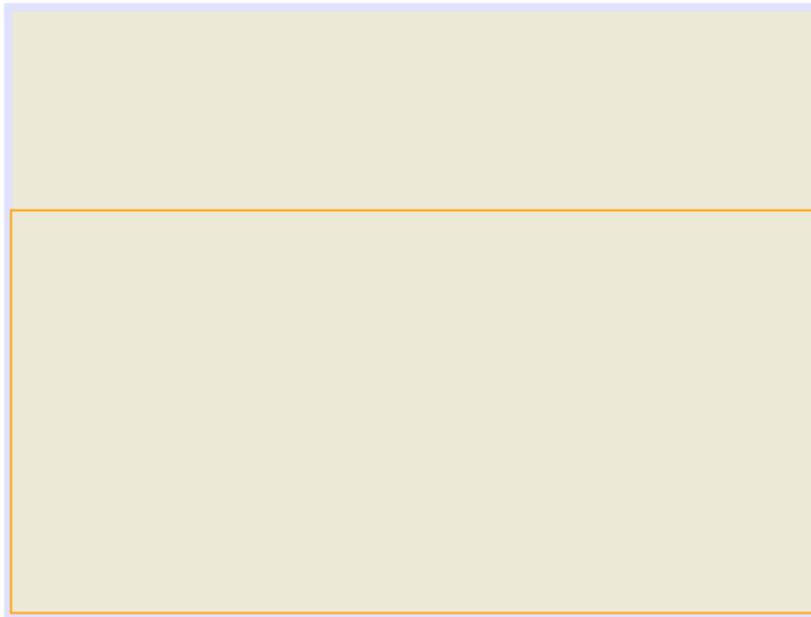


Panel en la zona norte.

- Cambia el nombre a este panel y llámalo *panelTitulo*, ya que contendrá el nombre del programa.

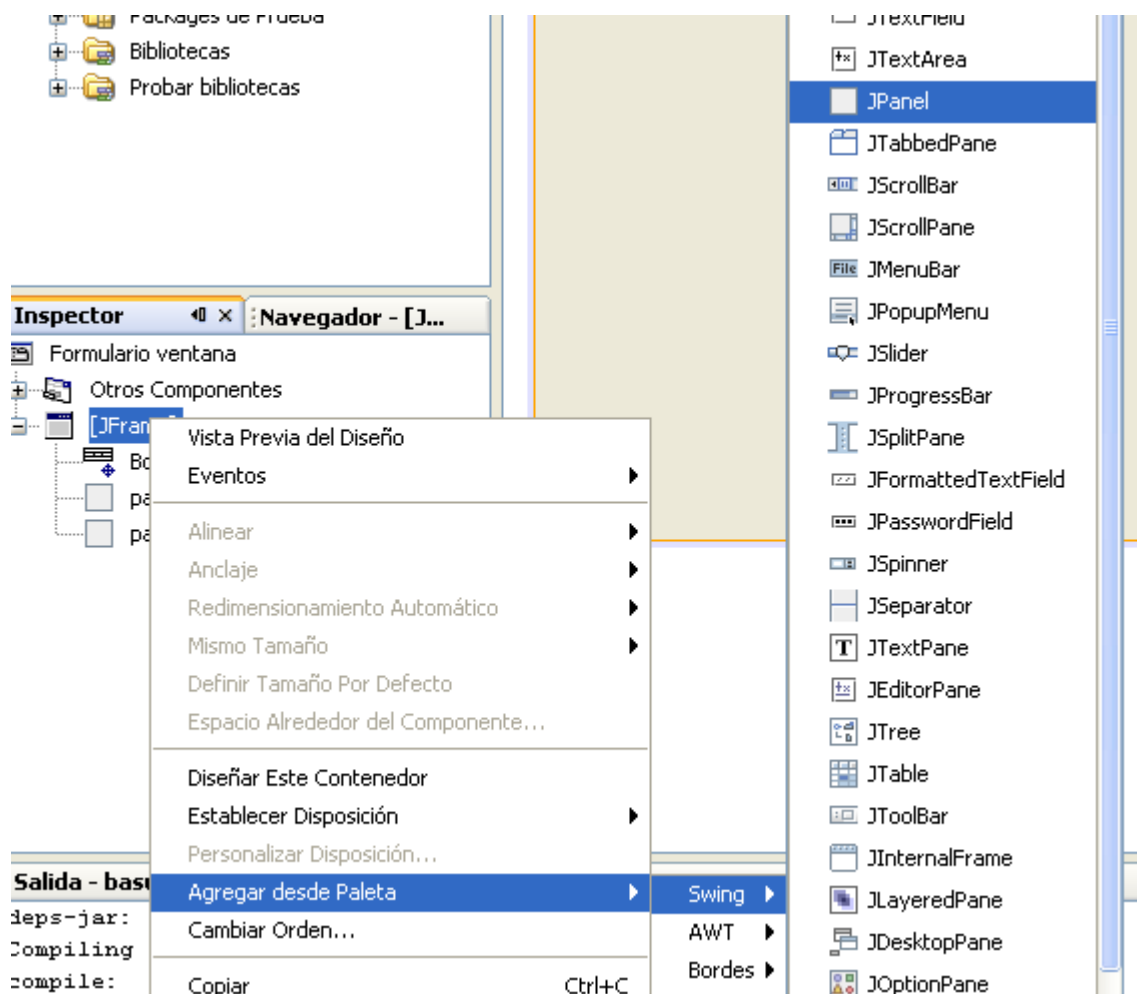


- Añade otro panel, esta vez a la parte central. El panel se llamará *panelDatos*:

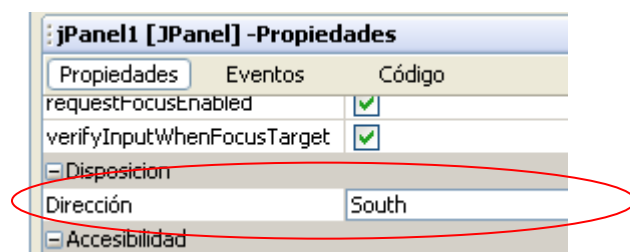


- Añade un nuevo panel en la parte sur de la ventana. Su nombre será *panelEstado*.

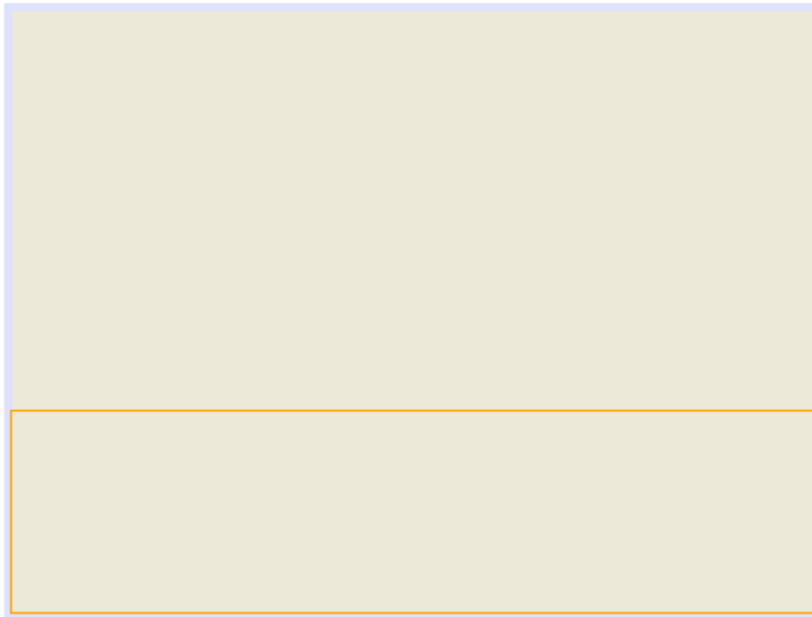
NOTA. A veces resulta complicado agregar un panel en una zona de la ventana cuando tenemos un BorderLayout. Puedes entonces hacer clic con el derecho sobre JFrame en el *Inspector* y activar la opción *Agregar desde paleta – Swing – JPanel*.



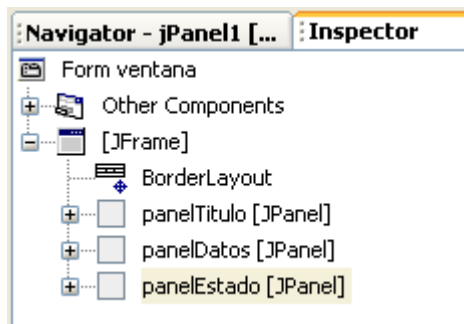
- Si el panel no se coloca en el sitio deseado, se puede seleccionar en el *Inspector* y activar su propiedad *Dirección*, e indicar la zona donde se quiere colocar:



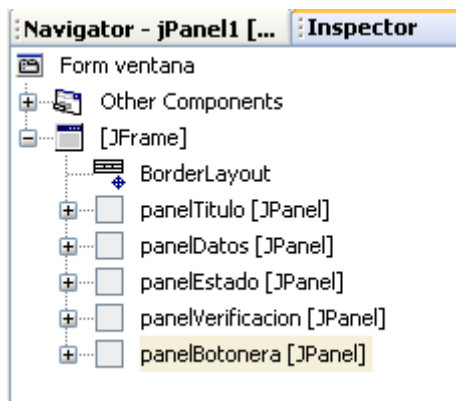
El panel debería estar situado finalmente en el sur de la ventana:



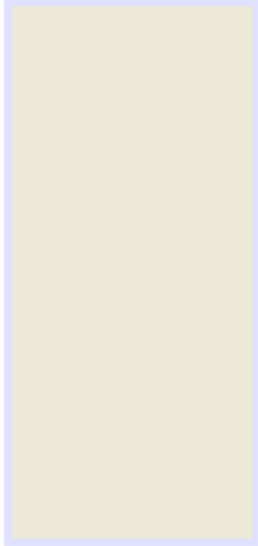
- El *Inspector* tendrá la siguiente forma ahora:



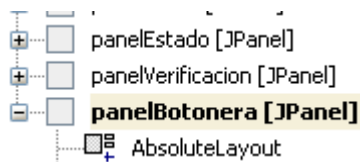
- Añade ahora tu solo un panel en la zona oeste llamado *panelBotonera* y otro en la zona esta llamado *panelVerificacion*. El *Inspector* debería tener la siguiente forma al finalizar:



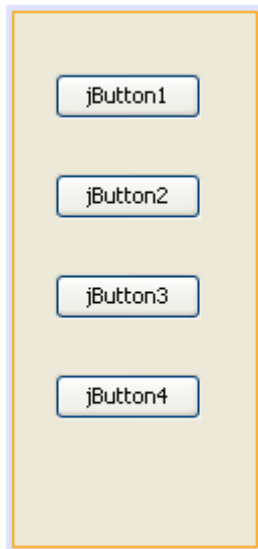
- Cada panel puede ser diseñado de forma individual, simplemente haciendo doble clic sobre él. Así pues, empezaremos diseñando el panel *panelBotonera*. Haz doble clic sobre él.
- En la parte izquierda del NetBeans aparecerá únicamente el *panelBotonera*. Agrándalo para que tenga la siguiente forma:



- A cada panel se le puede asignar un Layout distinto. A este panel le asignaremos un *AbsoluteLayout* para poder colocar cada elemento donde quiera. Asigna un *AbsoluteLayout* al panel haciendo clic con el derecho sobre él en el *Inspector*. El *Inspector* debería quedar así:

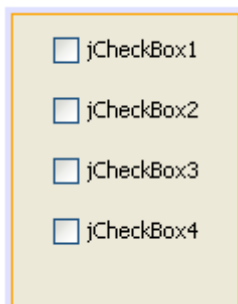


- Ahora añade cuatro botones al panel. Observa como tienes libertad total para colocar cada botón donde quieras. Procura que el panel quede así:

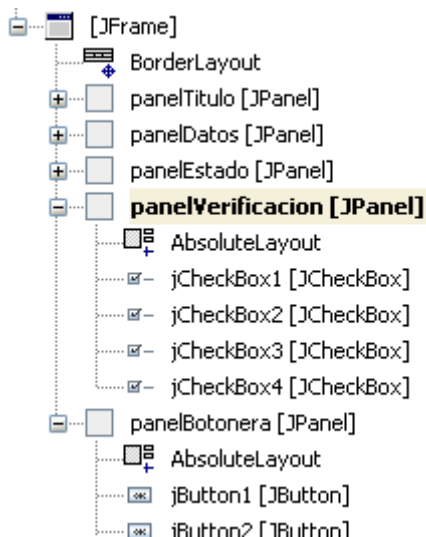


(No nos vamos a preocupar en este ejercicio de los nombres de los componentes)

- Ahora diseña el panel *panelVerificación* haciendo doble clic sobre él.
- Asígnale también un layout *AbsoluteLayout*.
- Coloca en él cuatro casillas de verificación. El aspecto del panel al terminar debe ser parecido al siguiente:



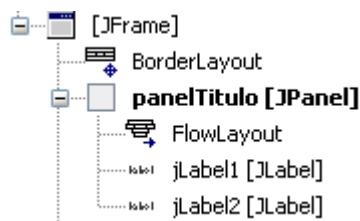
Y el *Inspector* debe tener un estado similar a este:



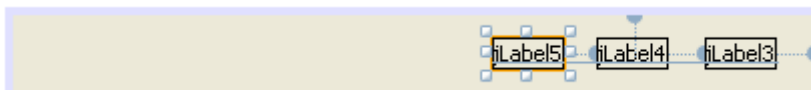
- Ahora se diseñará el *panelTitulo*. Haz doble clic sobre él.
- En este caso se le añadirá un FlowLayout. Recuerda que este layout hace que cada elemento se coloque uno detrás de otro.
- Añade al panel dos etiquetas como las que siguen. Ponle un borde a cada una:

Ejercicio de distribución de paneles y Layout Por Fulanito Pérez

El *Inspector* tendrá este aspecto en lo que se refiere al *panelTitulo*...



- El *panelEstado* lo diseñaremos sin asignar ningún layout, es decir, usando el *Diseño Libre*. En él añadiremos tres etiquetas de forma que estas mantengan una distancia relativa con respecto al límite derecho del panel. Dicho de otra forma, que siempre estén pegadas a la parte derecha del panel:

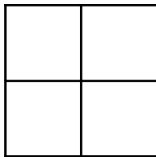


Observa las líneas “guía”. Indican que las etiquetas dependen de la parte derecha del panel. A su vez cada una depende de la otra. Es como si estuvieran “enganchadas”, como los vagones de un tren.

- El *panelDatos* lo vamos a complicar un poco. Haz doble clic sobre él para diseñarlo y asígnale un GridLayout.

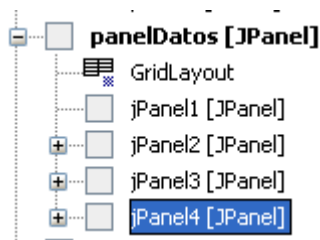


- Marca el GridLayout y asígnale 2 filas y 2 columnas, para que interiormente tenga forma de una rejilla como esta:

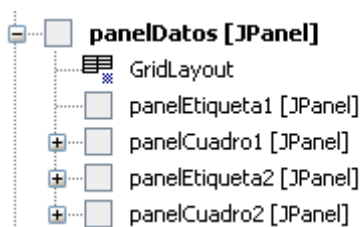


- A cada una de las divisiones del GridLayout del *panelDatos* le asignaremos un nuevo panel. Añade al *panelDatos* cuatro paneles. Esto lo puedes hacer fácilmente haciendo clic con el botón derecho del ratón sobre el *panelDatos* en el *Inspector* y eligiendo la opción *Añadir desde paleta – Swing – JPanel*.

El aspecto del inspector debería ser como el que sigue, en lo que se refiere al *panelDatos*:



- Asignaremos a cada uno de los cuatro paneles los siguientes nombres: *panelEtiqueta1*, *panelCuadro1*, *panelEtiqueta2*, *panelCuadro2*. El panel quedará así en el *Inspector*.

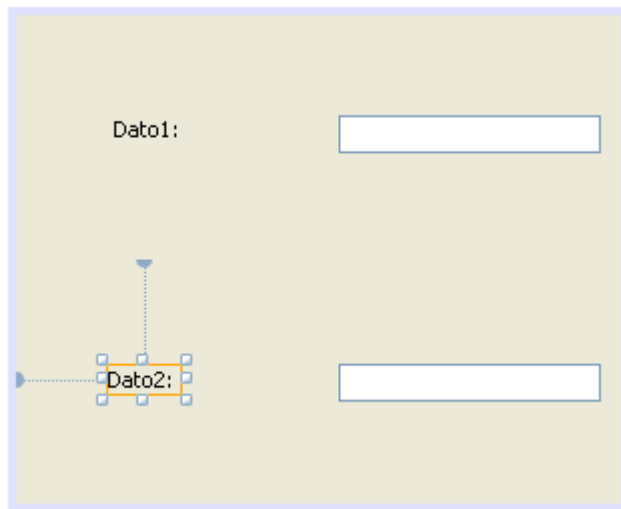


Así pues, el panel *panelDatos* tiene forma de rejilla con cuatro celdas, y en cada celda hay un panel. Puede imaginarse el *panelDatos* así:

panelDatos

PanelEtiqueta1	PanelCuadro1
PanelEtiqueta2	PanelCuadro2

- Ahora añade al panelEtiqueta1 y al panelEtiqueta2 sendas etiquetas. Y al panelCuadro1 y panelCuadro2 sendos cuadros de textos. El panel *panelDatos* debe quedar así:



- Finalmente ejecuta el programa y comprueba como se comportan los elementos según el panel donde se encuentre y el layout asignado a cada uno.

CONCLUSIÓN

Para el diseño de ventanas muy complejas, se suelen definir layouts que dividan en zonas el JFrame, como por ejemplo el BorderLayout o el GridLayout.

Dentro de cada una de dichas zonas se añade un JPanel, al que se le asigna un AbsoluteLayout, un FlowLayout o se mantiene el Diseño Libre.

Es posible asignar a un panel un layout de zonas, como el GridLayout, y, a su vez, introducir en él nuevos paneles, y así sucesivamente.

EJERCICIO GUIADO. JAVA: DIÁLOGOS

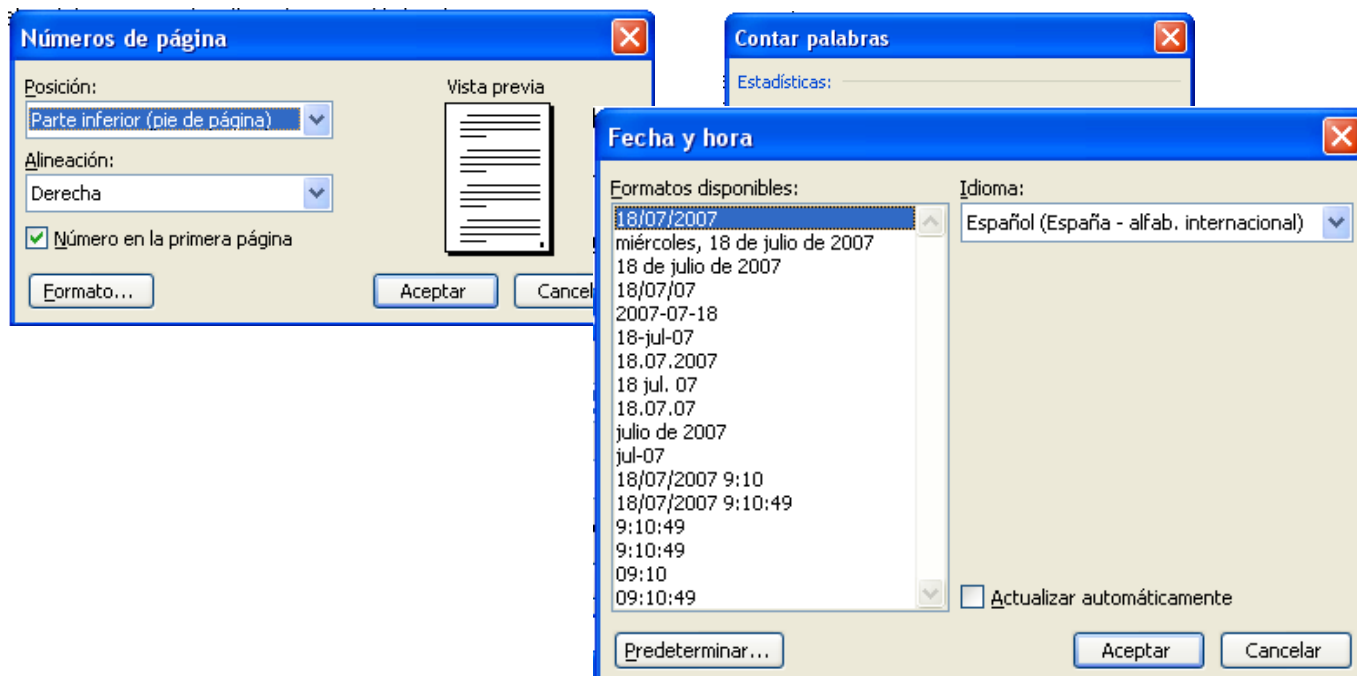
Cuadros de Diálogo

Un cuadro de diálogo es un cuadro con opciones que aparece normalmente cuando se activa una opción del menú principal del programa.

Los cuadros de diálogo tienen forma de ventana aunque no poseen algunas características de estas. Por ejemplo, no pueden ser minimizados ni maximizados.

Los cuadros de diálogo, aparte de las opciones que muestran, suelen contener dos botones típicos: el botón Aceptar y el botón Cancelar. El primero de ellos da por válidas las opciones elegidas y cierra el cuadro de diálogo. El segundo simplemente cierra el cuadro de diálogo sin hacer ninguna modificación.

He aquí algunos ejemplos de cuadros de diálogo del programa Word:



Para crear cuadros de diálogo en Java, se usa un tipo de objetos llamado JDialog. Estos objetos pueden ser diseñados como si fueran ventanas, aunque representan realmente cuadros de diálogo.

Ejercicio guiado

- Crea un nuevo proyecto en java.
- Diseña el JFrame de forma que la ventana tenga el siguiente aspecto:

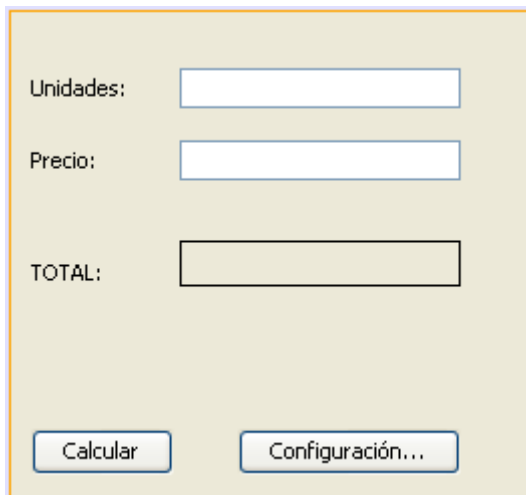


Diagrama de una ventana de Java Swing con un fondo beige. En la parte superior izquierda, hay tres etiquetas: 'Unidades:', 'Precio:' y 'TOTAL:'. A la derecha de 'Unidades:' y 'Precio:' hay cuadros de texto blancos. A la derecha de 'TOTAL:' hay un cuadro de texto gris. En la parte inferior, hay dos botones: 'Calcular' y 'Configuración...'. El botón 'Calcular' está a la izquierda del botón 'Configuración...'.

Los elementos de la ventana tendrán los siguientes nombres:

- Cuadro de texto de unidades: txtUnidades.
 - Cuadro de texto de precio: txtPrecio.
 - Etiqueta con borde del total: etiTotal.
 - Botón Calcular: btnCalcular.
 - Botón Configuración: btnConfiguracion.
-
- Se pretende que cuando se pulse el botón Calcular se calcule el total de la venta (esto se hará luego) Para hacer el cálculo se tendrán en cuenta el IVA y el Descuento a aplicar. Estos dos valores serán variables globales, ya que se usarán en distintos lugares del programa.
 - Así pues entra en el código y declara una variable global *iva* y otra *descuento* tal como se indica a continuación (recuerda que las variables globales se colocan justo después de la línea donde se define la clase principal *public class*):

```

    * @author didact
    */
    public class ventanaprincipal extends javax.swing.JFrame {

        double iva;
        double descuento;

        /** Creates new form ventanaprincipal */
        public ventanaprincipal() {
            initComponents();
        }
    }

```

Variables globales

- Cuando el programa arranque, interesará que el *iva* por defecto sea 0, y que el *descuento* por defecto sea 0 también, así que en el constructor, inicializaremos las variables globales *iva* y *descuento* a 0:

```

    */
    public class ventanaprincipal extends javax.swing.JFrame {

        double iva;
        double descuento;

        /** Creates new form ventanaprincipal */
        public ventanaprincipal() {
            initComponents();
            iva=0;
            descuento=0;
        }
    }

```

Inicialización de variables globales

- Estamos ya preparados para programar el botón btnCalcular. Entra en su `actionPerformed` y allí se programará la realización del cálculo de la siguiente forma:

```

double unidades;
double precio;
double total;    //total
double cantiva;  //cantidad iva
double cantdes;  //cantidad descuento
double totalsiniva; //total sin iva

//Recojo los datos de los cuadros de textos (convirtiendolos a números)
unidades = Double.parseDouble(txtUnidades.getText());
precio = Double.parseDouble(txtPrecio.getText());

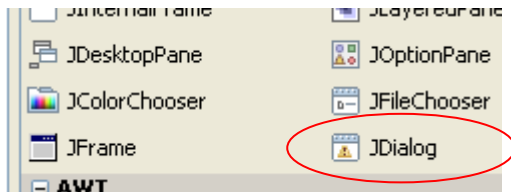
//Calculo el total sin iva, la cantidad de iva y la cantidad de descuento
totalsiniva=precio*unidades;
cantiva=totalsiniva*iva/100;
cantdes=totalsiniva*descuento/100;

```

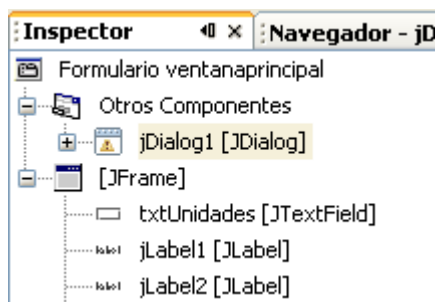
```
//Ahora calculo el precio total:
total = totalsiniva+cantiva-cantdes;

//Coloco el total en la etiqueta:
etiTotal.setText(""+total);
```

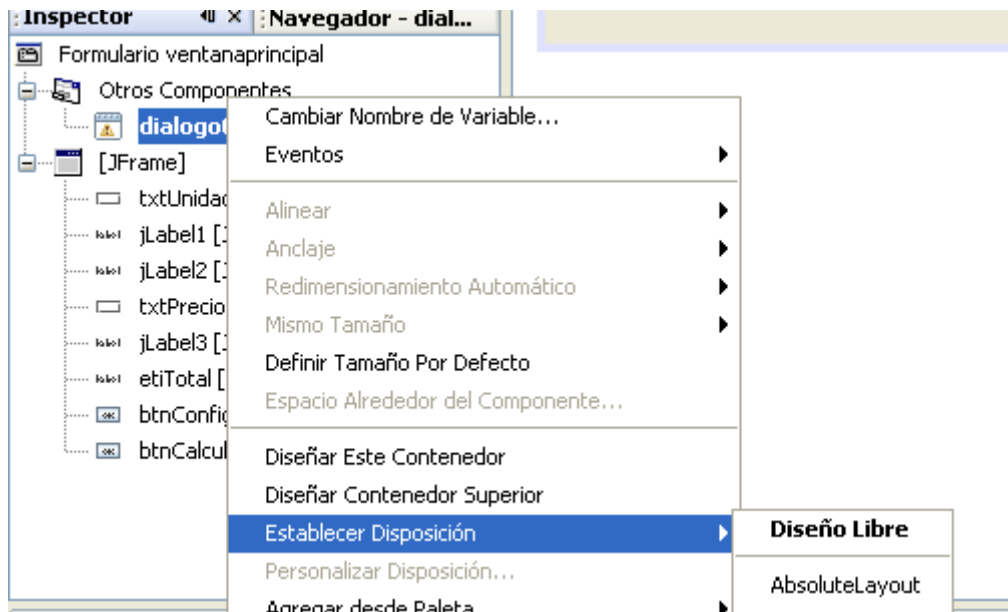
- Puedes ya ejecutar el programa y comprobar que el botón Calcular funciona, aunque el cálculo que realiza lo hace con un iva 0 y un descuento 0.
- A continuación se programará el botón Configuración de forma que nos permita decidir qué iva y qué descuento queremos aplicar. Este botón mostrará un CUADRO DE DIÁLOGO que permita al usuario configurar estos datos.
- Para añadir un cuadro de diálogo al proyecto, se tiene que añadir un objeto del tipo JDialog sobre el JFrame.



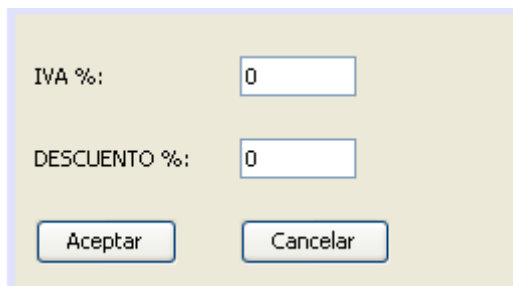
- Los JDialog son objetos ocultos, es decir, objetos que se colocan en la parte del *Inspector* llamada *Otros Componentes*, al igual que sucede con los menús contextuales o los JFileChooser. Observa tu inspector, allí verás el JDialog que has añadido:



- Cámbiale el nombre. Lo llamaremos *dialogoConfiguracion*.
- Los diálogos normalmente traen por defecto el layout BorderLayout. Para nuestro ejemplo cambiaremos el layout del JDialog por el Diseño Libre:



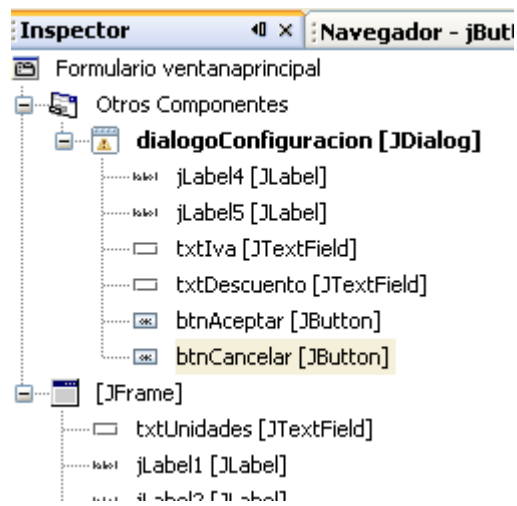
- Los JDialog se pueden diseñar independientemente, al igual que los JPanel. Solo tienes que hacer doble clic sobre el *dialogoConfiguracion* (en el *inspector*) y este aparecerá en el centro de la ventana.
- Así pues debes diseñar el *dialogoConfiguracion* para que quede de la siguiente forma:



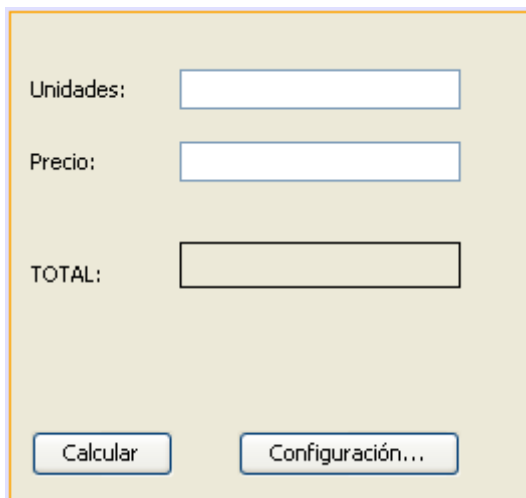
Los elementos del cuadro de diálogo tienen los siguientes nombres:

- El cuadro de texto del Iva: txtIva.
- El cuadro de texto del Descuento: txtDescuento.
- El botón Aceptar: btnAceptar.
- El botón Cancelar: btnCancelar.

Si observas el *Inspector* debe tener el siguiente aspecto:



- Se ha dicho que cuando se pulse el botón Configuración en la ventana principal, debe aparecer el cuadro de diálogo *dialogoConfiguracion*, que acabas de diseñar:

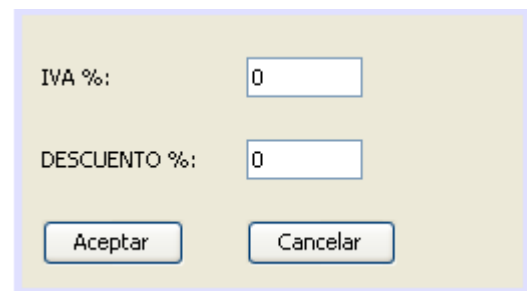


Unidades:

Precio:

TOTAL:

Haces clic sobre Configuración
y aparece el diálogo



IVA %:

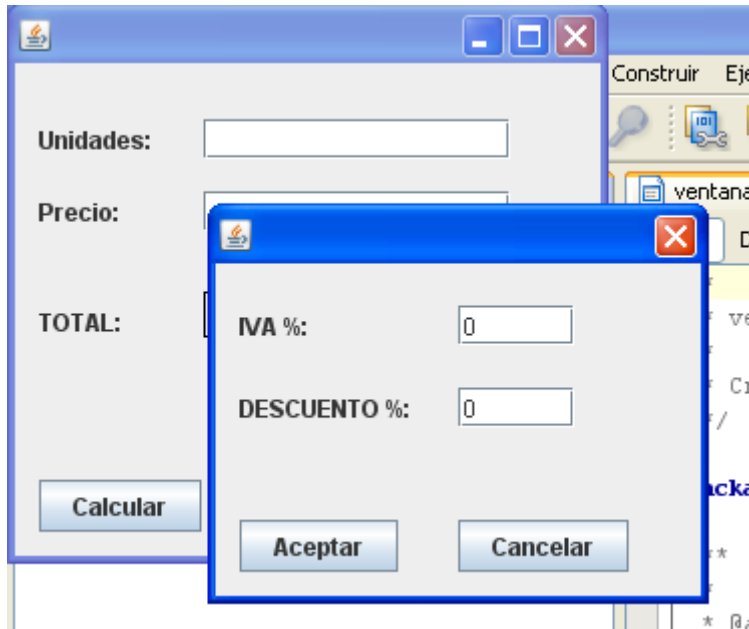
DESCUENTO %:

- Para conseguir esto, debes programar el *actionPerformed* del botón *btnConfiguracion* de la siguiente forma:

```
dialogoConfiguracion.setSize(250,200);  
dialogoConfiguracion.setLocation(100,100);  
dialogoConfiguracion.setVisible(true);
```

- El código anterior hace lo siguiente:
 - A través del método *setSize* se asigna un tamaño de 250 x 200 al cuadro de diálogo.
 - A través del método *setLocation* se determina que el cuadro de diálogo aparecerá en la posición (100, 100) de la pantalla.
 - A través del método *setVisible* hacemos que el cuadro de diálogo se muestre.

- Ejecuta el programa y observa lo que sucede cuando pulsas el botón Configurar. Debería aparecer el cuadro de diálogo en la posición programada y con el tamaño programado:



- Los botones Aceptar y Cancelar del cuadro de diálogo aún no hacen nada. Así que los programaremos. Empezaremos por el más sencillo, el botón Cancelar.
- El botón Cancelar de un cuadro de diálogo simplemente cierra dicho cuadro de diálogo. Para ello, debes añadir el siguiente código en el *actionPerformed* del botón Cancelar del diálogo:

```
dialogoConfiguracion.dispose();
```

El método *dispose* se usa para cerrar un cuadro de diálogo. También se puede usar con un *JFrame* para cerrarlo.

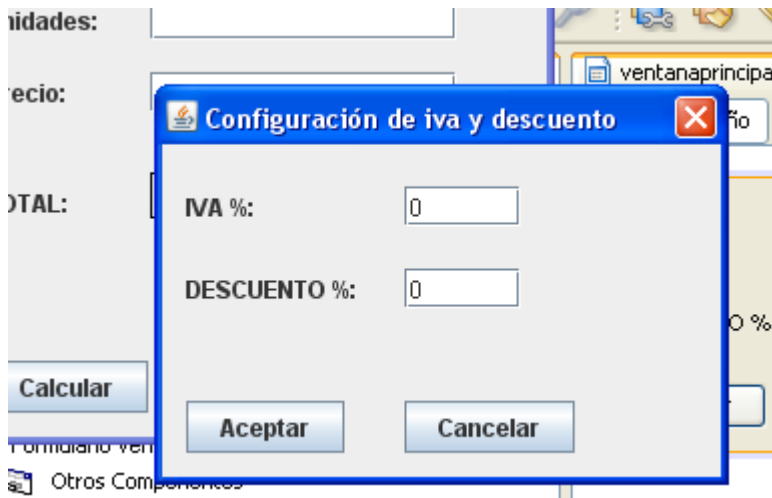
- Ejecuta el programa de nuevo y comprueba el funcionamiento del botón Cancelar del cuadro de diálogo.
- Ahora se programará el botón Aceptar. Cuando el usuario pulse este botón, se confirmará el valor del *iva* y del *descuento* que haya introducido. Es decir, se traspasarán los valores introducidos en los cuadros de texto *txtIva* y *txtDescuento* a las variables globales *iva* y *descuento*.

Una vez que se haya hecho esto, el cuadro de diálogo se debe cerrar.

- Este es el código que hace lo anterior. Debe programarlo en el *actionPerformed* del botón Aceptar:

```
iva = Double.parseDouble(txtIva.getText());
descuento=Double.parseDouble(txtDescuento.getText());
dialogoConfiguracion.dispose();
```

- Observe el código. Primero se traspasa los valores de los cuadros de texto a las variables globales y luego se cierra el cuadro de diálogo.
- Compruebe el funcionamiento del programa de la siguiente forma:
 - o Ejecute el programa.
 - o Introduzca 5 unidades y 20 de precio.
 - o Si pulsa calcular, el total será 100. (No hay ni iva ni descuento al empezar el programa)
 - o Ahora pulse el botón Configuración, e introduzca un iva del 16. El descuento déjelo a 0. Acepte.
 - o Ahora vuelva a calcular. Observe como ahora el total es 116, ya que se tiene en cuenta el iva configurado.
 - o Pruebe a configurar un descuento y vuelva a calcular.
- Se pretende ahora mejorar un poco el cuadro de diálogo, añadiéndole un título. Seleccione el cuadro de diálogo en el *Inspector* y luego busque su propiedad *title*. En ella escriba “Configuración de iva y descuento”.
- Vuelva a ejecutar el programa. Observe la barra de título del cuadro de diálogo:

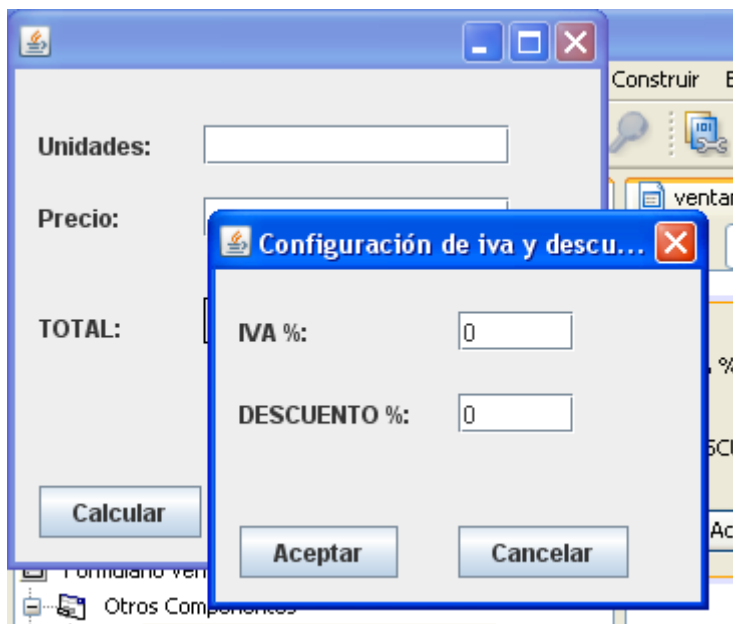


- Ahora se estudiará el concepto de cuadro de diálogo modal y cuadro de diálogo no modal.

1. Un cuadro de diálogo no modal. Es aquel que permite activar la ventana desde la que apareció. Los cuadros de diálogo añadidos a un proyecto son por defecto no modales.

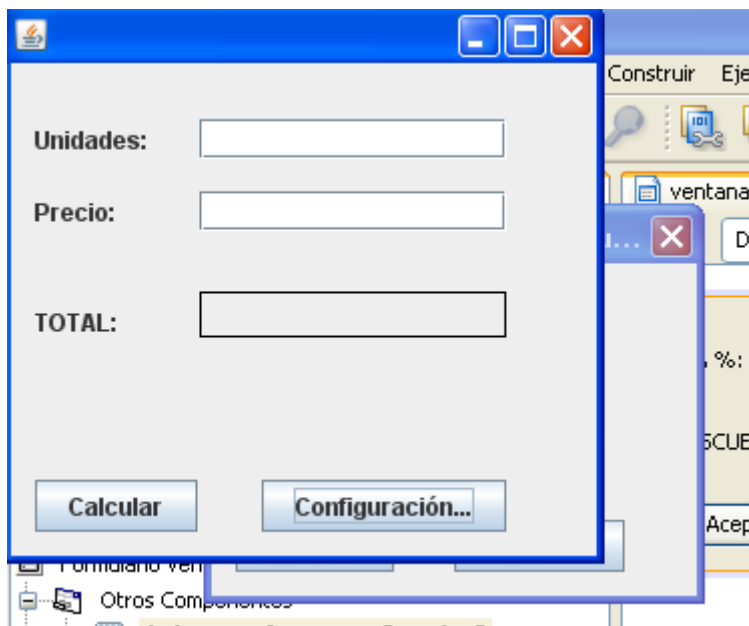
- Ejecuta el programa y prueba a hacer lo siguiente:
 - o Pulsa el botón Configurar. Aparecerá el cuadro de diálogo.

- o Pulsa sobre la ventana.



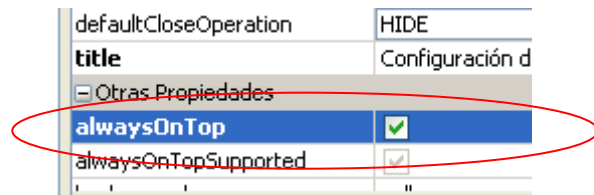
Pulsa sobre la ventana.

- o Observarás que la ventana se activa, colocándose sobre el cuadro de diálogo.

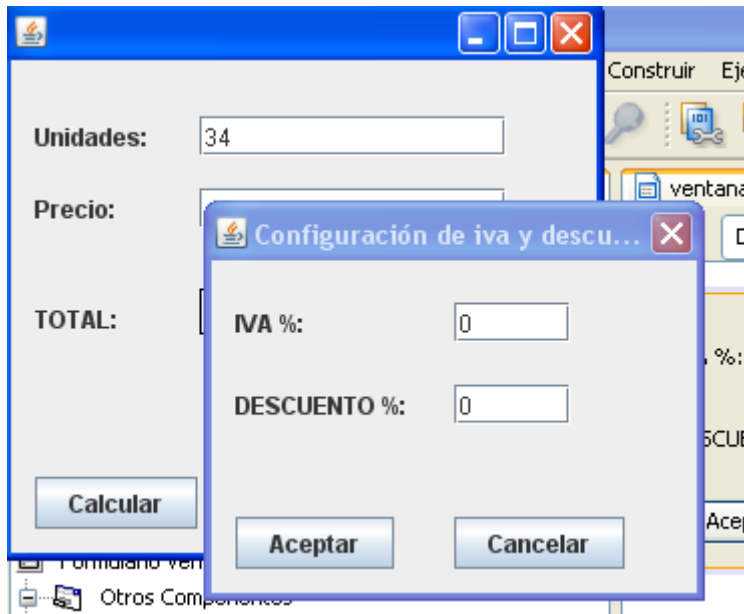


La ventana se activa colocándose por encima del cuadro de diálogo.

- o Esto es posible gracias a que el cuadro de diálogo es *no modal*.
- o A veces, puede ser interesante que se active la ventana pero que el cuadro de diálogo siga delante de ella. Para conseguir esto, es necesario activar la propiedad del cuadro de diálogo llamada *alwaysOnTop*. Activa esta propiedad:



- o Ahora ejecuta el programa de nuevo y haz que se visualice el cuadro de diálogo de configuración. Podrás comprobar que se puede activar la ventana e incluso escribir en sus cuadros de textos, y que el cuadro de diálogo sigue visible:



Se puede activar la ventana trasera, e incluso escribir en ella. Esto es gracias a que el cuadro de diálogo es *no modal*.

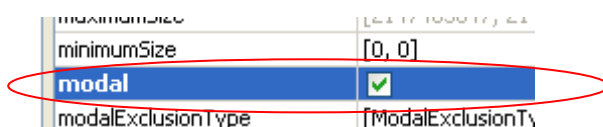
Por otro lado, el cuadro de diálogo sigue mostrándose delante de la ventana. Esto es gracias a la propiedad *alwaysOnTop*

- o Es muy común, cuando tenemos un cuadro de diálogo *no modal*, usar la propiedad *alwaysOnTop*, para que siempre aparezca delante de la ventana.

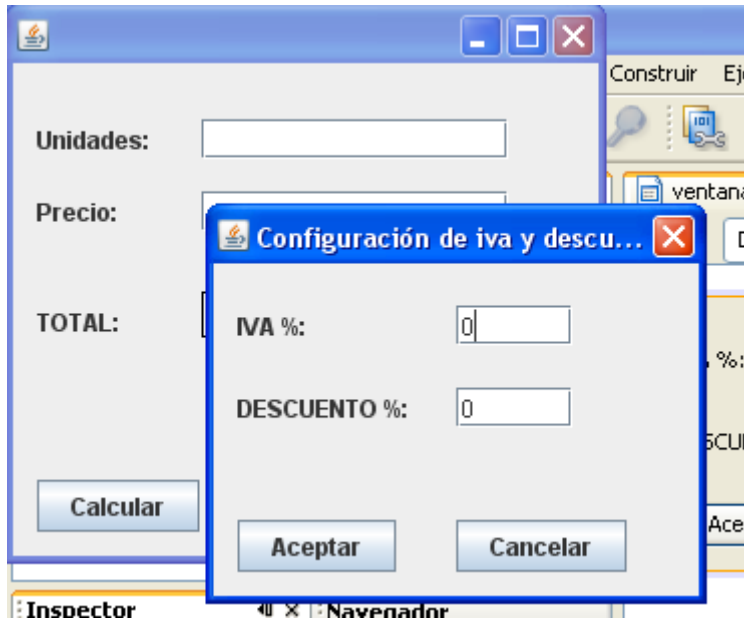
- Ahora se estudiará el concepto de cuadro de diálogo *modal*.

2. Un cuadro de diálogo *modal* es aquel que no permite que se active otra ventana hasta que este no se haya cerrado.

- Para convertir nuestro cuadro de diálogo en *modal*, será necesario que lo selecciones en el *inspector* y busques la propiedad *modal*. Debes activar esta propiedad.



- Ahora ejecuta el programa comprueba lo siguiente:
 - o Haz que se visualice el cuadro de diálogo de configuración.
 - o A continuación intenta activar la ventana haciendo clic sobre ella. Verás como no es posible activarla. Es más, intenta escribir en sus cuadros de texto. No será posible hacerlo. (Incluso observarás un parpadeo en el cuadro de diálogo avisándote de ello). Esto es debido a que ahora nuestro cuadro de diálogo es *modal*.



Aunque intentes activar la ventana o escribir en ella, no podrás, ya que el cuadro de diálogo es *modal*.

Incluso verás un parpadeo en el cuadro de diálogo cuando intentas activar la otra ventana.

Se podría decir que un cuadro de diálogo *modal* es un acaparador, y que no te deja usar otro elemento hasta que no acabes con él.

Solo cuando cierres el cuadro de diálogo podrás seguir trabajando con la ventana.

- o Solo cuando pulses, Aceptar, o Cancelar, o cierres el cuadro de diálogo, podrás seguir trabajando con tu ventana.

CONCLUSIÓN

Los Cuadros de Diálogo son ventanas simplificadas que muestran distintas opciones al usuario.

Los objetos `JDialog` son los que permiten la creación y uso de cuadros de diálogo en un proyecto java.

Para visualizar un `JDialog` será necesario llamar a su método `setVisible`. También son interesantes los métodos `setSize` para asignarles un tamaño y `setLocation` para situar el cuadro de diálogo en la pantalla.

Para cerrar un `JDialog` será necesario invocar a su método `dispose`.

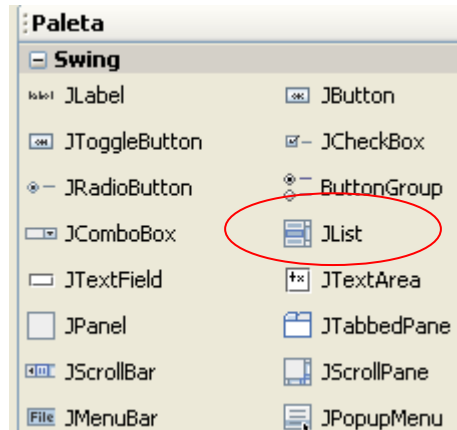
Existen dos tipos de cuadros de diálogo: los modales y no modales.

Los cuadros de diálogo modales no permiten que se active otra ventana hasta que el cuadro de diálogo no se haya cerrado.

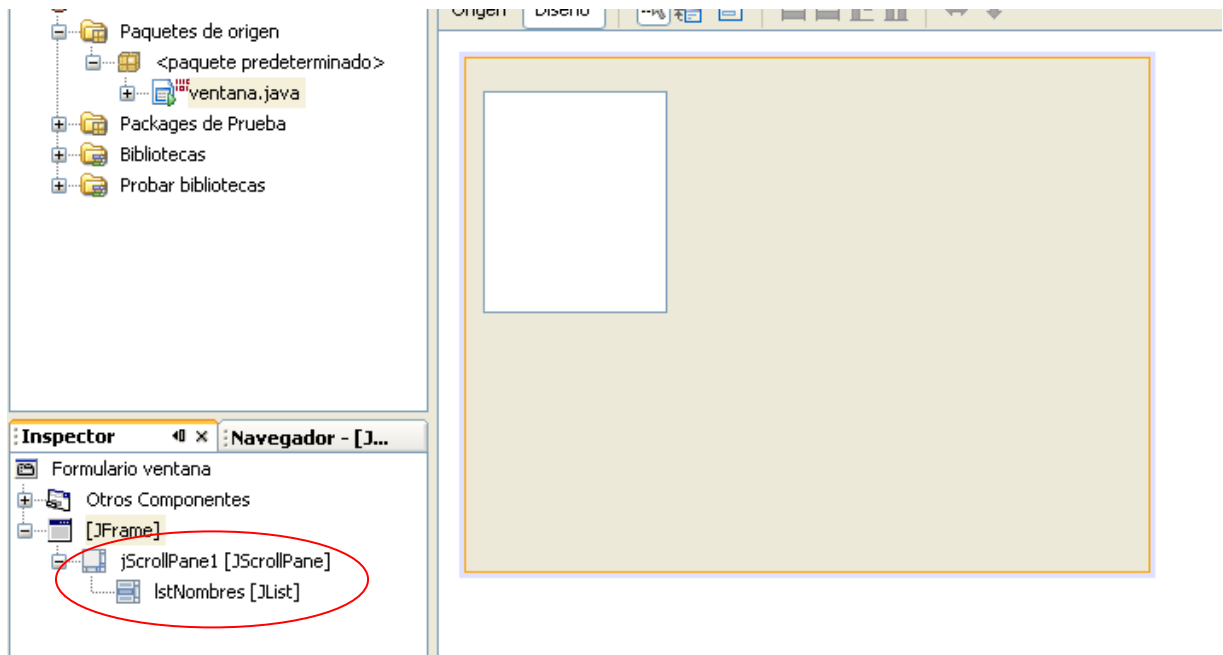
Los cuadros de diálogo no modales permiten trabajar con otra ventana a pesar de que el propio cuadro de diálogo no haya sido cerrado.

EJERCICIO GUIADO. JAVA: MODELOS DE CUADRO DE LISTA

127. Realiza un nuevo proyecto.
128. En la ventana principal debes añadir lo siguiente:
yyyyy. Una etiqueta con borde llamada etiResultado.
130. Añade un cuadro de lista al formulario (JList).



131. Borra todo el contenido de la lista (propiedad model) y cámbiale el nombre a la lista. La lista se llamará *IstNombres*. Recuerda que las listas aparecen dentro de un objeto del tipo JScrollPane.



132. Añade dos botones al formulario. Uno de ellos tendrá el texto “Curso 1” y se llamará btnCurso1 y el otro tendrá el texto “Curso 2” y se llamará btnCurso2.

El diagrama muestra un formulario con un fondo beige. En la parte superior izquierda hay un cuadro de lista rectangular vacío con un borde azul. A la derecha de este cuadro hay un campo de texto rectangular con un borde negro. En la parte inferior izquierda del formulario hay dos botones rectangulares con bordes azules. El primer botón está etiquetado como 'Curso 1' y el segundo como 'Curso 2'.

133. En el evento *actionPerformed* del botón “Curso 1” programa lo siguiente:

```
DefaultListModel modelo = new DefaultListModel();  
modelo.addElement("Juan");  
modelo.addElement("María");  
modelo.addElement("Luis");  
lstNombres.setModel(modelo);
```

134. En el evento *actionPerformed* del botón “Curso 2” programa lo siguiente:

```
DefaultListModel modelo = new DefaultListModel();  
modelo.addElement("Ana");  
modelo.addElement("Marta");  
modelo.addElement("Jose");  
lstNombres.setModel(modelo);
```

135. Explicación de los códigos anteriores:

fffff. Lo que hace cada botón es rellenar el cuadro de lista con una serie de nombres. En el caso del botón “Curso 1”, la lista se rellena con los nombres Juan, María y Luis, mientras que en el caso del botón “Curso 2”, la lista se rellena con los nombres Ana, Marta y Jose.

gggggg. El contenido de un cuadro de lista es lo que se denomina un “modelo”. El “modelo” es un objeto que contiene el listado de elementos de la lista.

hhhhhh. Los modelos de las listas son objetos del tipo *DefaultListModel*.

iiii. Lo que hace el programa es crear un “modelo”. Luego rellena el “modelo” con datos, y finalmente asocia el “modelo” al cuadro de lista. Veamos como se hace todo esto.

jjjjj. Primero se crea el “modelo”, a través de la siguiente instrucción (será necesario añadir el *import* correspondiente, atento a la bombillita):

```
DefaultListModel modelo = new DefaultListModel();
```

kkkkk. El “modelo” tiene un método llamado *addElement* que permite introducir datos dentro de él. Así pues usamos este método para añadir los datos al modelo.

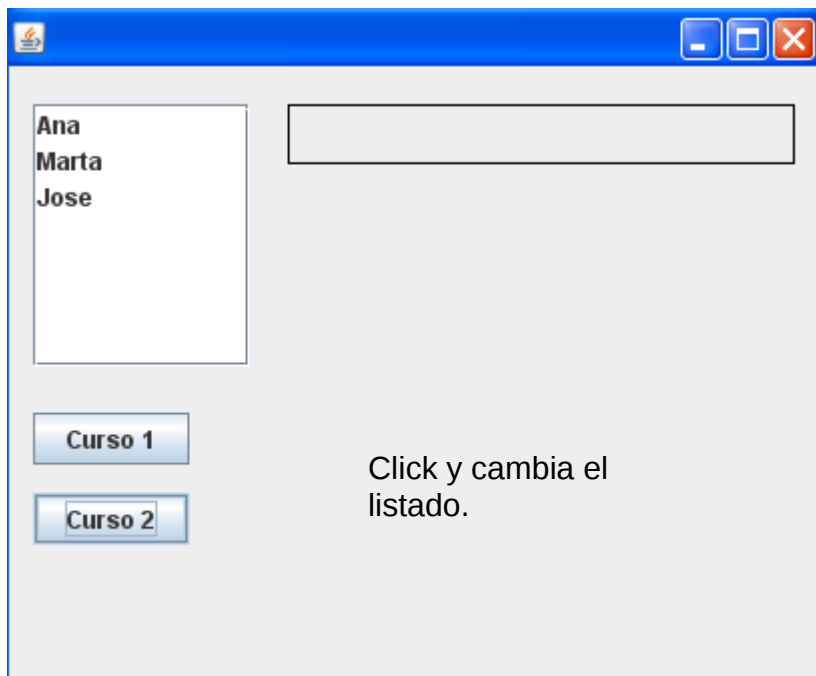
```
modelo.addElement("Ana");  
modelo.addElement("Marta");  
modelo.addElement("Jose");
```

lllll. Finalmente asociamos el “modelo” creado al cuadro de lista de la siguiente forma:

```
lstNombres.setModel(modelo);
```

mmmmm. Así pues, aquí tienes una forma de cambiar el contenido de un cuadro de lista desde el propio programa.

144. Prueba a ejecutar el programa. Observa como cuando pulsas cada botón cambia el contenido de la lista:

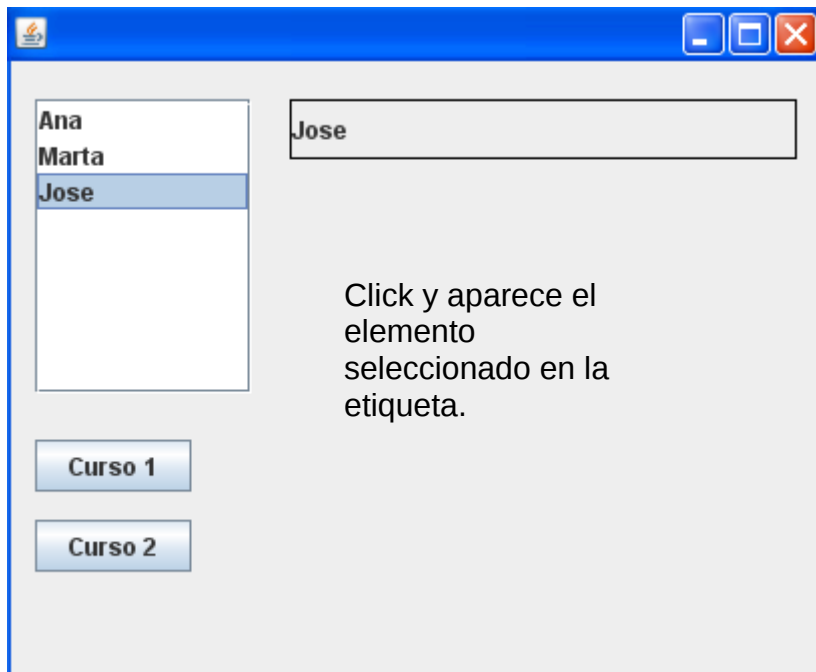


145. Ahora añade el siguiente código al evento *mouseClicked* del cuadro de lista:

```
etiResultado.setText(lstNombres.getSelectedValue().toString());
```

Esta instrucción hace que al seleccionar un elemento del cuadro de lista éste aparezca en la etiqueta etiResultado. Recuerda que el método `getSelectedValue` permite recoger el elemento seleccionado (hay que convertirlo a cadena con `toString`)

146. Ejecuta el programa:



147. Una propuesta. Añada un botón “Vaciar” llamado `btnVaciar`. Este botón vaciará el contenido de la lista. Para esto lo único que tiene que hacer es crear un modelo y, sin introducir ningún valor en él, asociarlo al cuadro de lista.

CONCLUSIÓN

Un cuadro de lista es un objeto que contiene a su vez otro objeto denominado “modelo”.

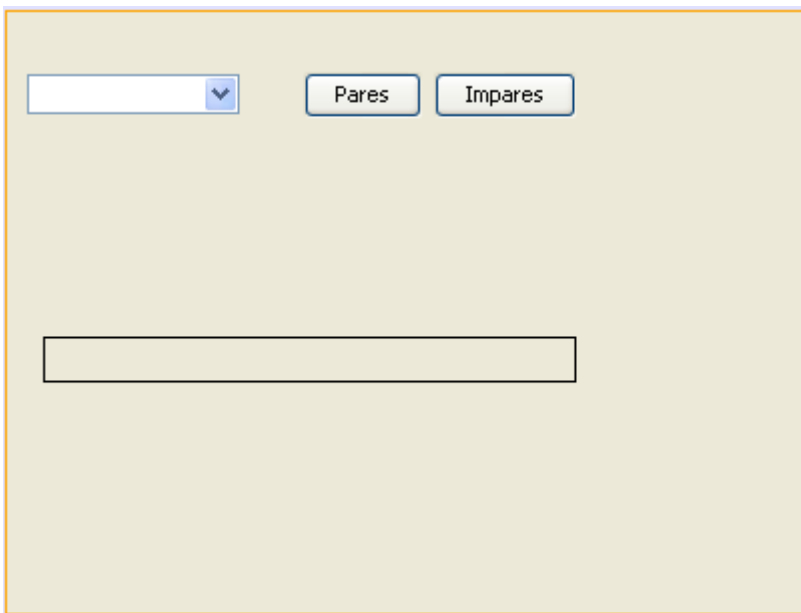
El objeto “modelo” es el que realmente contiene los datos de la lista.

Cuadro de lista → Modelo → Datos

Se puede crear un “modelo” y luego introducir datos en él. Luego se puede asociar ese “modelo” a la lista. De esta manera se puede cambiar el contenido de la lista en cualquier momento.

EJERCICIO GUIADO. JAVA: MODELOS DE CUADRO DE LISTA

148. Realiza un nuevo proyecto.
149. En la ventana principal debes añadir lo siguiente:
- ttttt. Un combo llamado cboNumeros.
 - uuuuuu. Un botón "Pares" llamado btnPares.
 - vvvvvv. Un botón "Impares" llamado btnImpares.
 - wwwwww. Una etiqueta con borde llamada etiResultado.
154. Elimina todos los elementos que contenga el combo. Recuerda, debes usar la propiedad "model" del combo para cambiar sus elementos.
155. Después de haber hecho todo esto, tu ventana debe quedar más o menos así:



156. En el evento *actionPerformed* del botón Pares, programa lo siguiente:

```
int i;

DefaultComboBoxModel modelo = new DefaultComboBoxModel();

for (i=0;i<10;i+=2) {
    modelo.addElement("Nº "+i);
}

cboNumeros.setModel(modelo);
```

157. Observa lo que hace este código:
bbbbbbb. Crea un objeto "modelo" para el combo.

Al igual que pasa con los cuadros de lista, los combos tienen un objeto "modelo" que es el que realmente contiene los datos. En el caso de los combos, para crear un objeto "modelo" se usará esta instrucción:

```
DefaultComboBoxModel modelo = new DefaultComboBoxModel();
```

cccccc. A continuación, se usa el objeto “modelo” creado y se rellena de datos. Concretamente, se rellena con los números pares comprendidos entre 0 y 10.

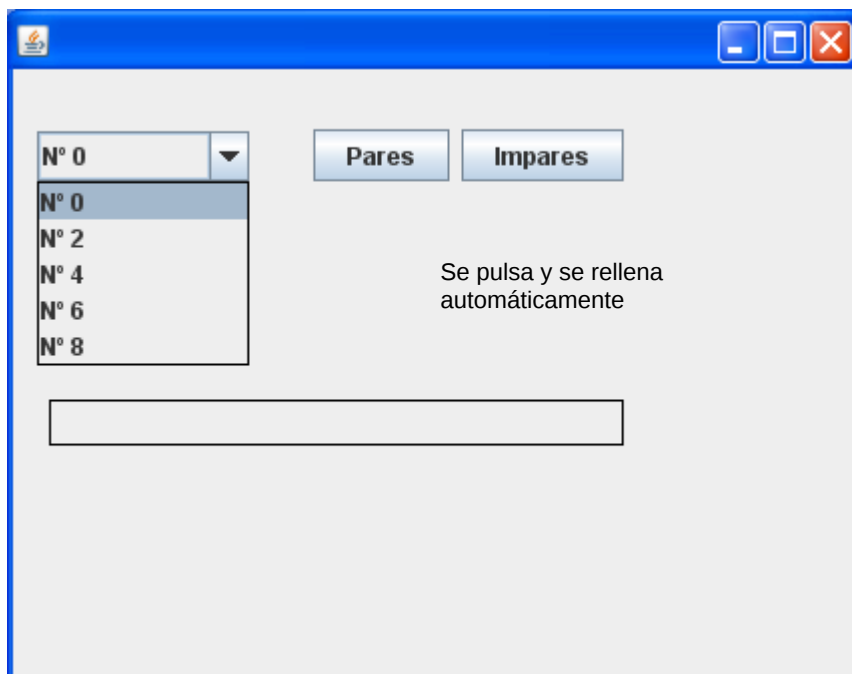
dddddd. Observa el uso de la propiedad `addElement` para añadir un elemento al modelo del combo.

eeeeee. Se ha usado un bucle `for` para hacer la introducción de datos en el modelo más fácil.

ffffff. Finalmente, se asocia el modelo al combo a través de la siguiente línea, con lo que el combo aparece relleno con los elementos del modelo:

```
cboNumeros.setModel(modelo);
```

163. Ejecuta el programa y observa el funcionamiento del botón Pares.



164. El botón Impares es similar. Programa su *actionPerformed* como sigue:

```
int i;
DefaultComboBoxModel modelo = new DefaultComboBoxModel();

for (i=1;i<10;i+=2) {
    modelo.addElement("Nº "+i);
}

cboNumeros.setModel(modelo);
```

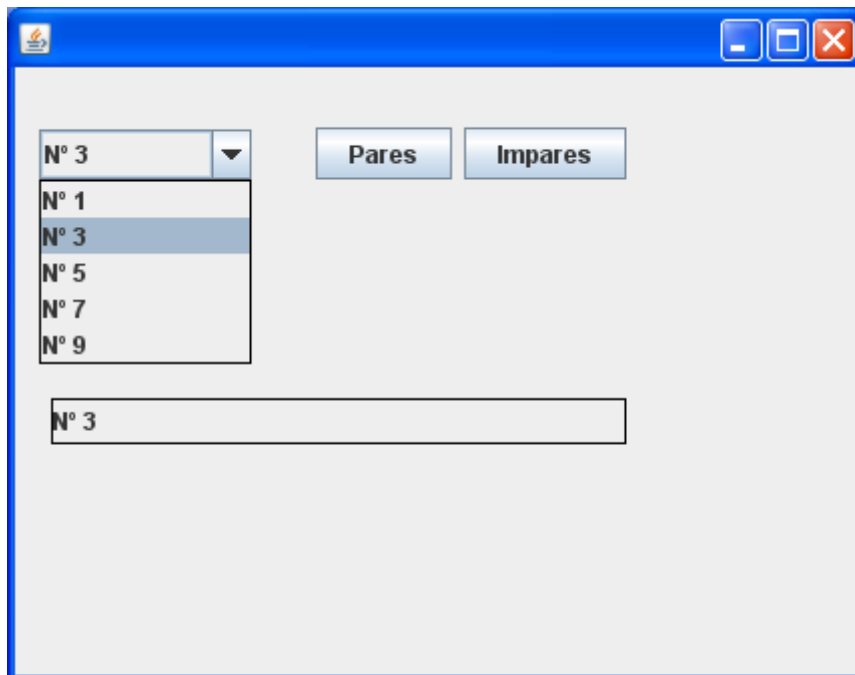
165. La única diferencia de este código es el `for`, que está diseñado para que se introduzcan los números impares comprendidos entre 0 y 10 dentro del modelo.

166. Finalmente se programará el *actionPerformed* del combo para que al seleccionar un elemento este aparezca en la etiqueta. Esto se hace con una simple instrucción:

```
etiResultado.setText(cboNumeros.getSelectedItem().toString());
```

Recuerda el uso de `getSelectedItem()` para recoger el elemento seleccionado, y el uso de `toString()` para convertirlo a texto.

167. Prueba el programa. Prueba los botones Pares e Impares y prueba el combo.



168. Sería interesante añadir un botón “Vaciar” llamado `btnVaciar` que vaciara el contenido del combo. Esto se haría simplemente creando un modelo vacío y asignarlo al combo. Se anima al alumno a que realice esta mejora.

CONCLUSIÓN

Un combo, al igual que los cuadros de lista, es un objeto que contiene a su vez otro objeto denominado “modelo”.

El objeto “modelo” es el que realmente contiene los datos del combo.

Combo → Modelo → Datos

Se puede crear un “modelo” y luego introducir datos en él. Luego se puede asociar ese “modelo” al combo. De esta manera se puede cambiar el contenido del combo en cualquier momento.