Índice

1. Introducción

En esta sesión se van a ver algunos aspectos de JFC (Java Foundation Classes). JFC es un conjunto de componentes para trabajo con interfaces gráficas de usuario en Java.

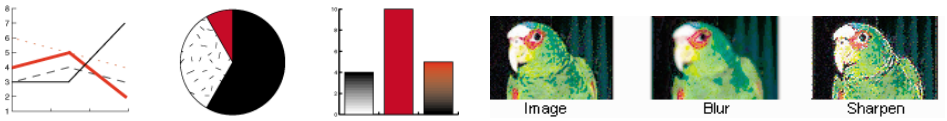
Contiene:

- Abstract Window Toolkit (AWT)

  API para el diseño de interfaces gráficas de usuario que se integran en el sistema de ventanas nativo del sistema donde se ejecutan, incluyendo APIs para arrastrar y soltar.

- Java 2D

- *APIs para trabajar con gráficos 2D, trabajo con imágenes, texto e impresión.*



- Swing

- *APIs que extienden AWT para proporcionar una biblioteca de componentes para el diseño de interfaces gráficas de usuario enteramente realizadas en Java.*

- Accesibilidad

- *APIs para permitir que las aplicaciones sean accesibles a las personas con discapacidades.*

- Internacionalización

- *Todas estas APIs incluyen soporte para crear aplicaciones que puedan ser utilizadas independientemente de la localización del usuario.*

Aquí vamos a ver algo de Swing y de AWT.

A grandes rasgos, los pasos para crear una interfaz gráfica de usuario son

- Crear una ventana
- Colocar componentes en la ventana
- Organizar los componentes en los contenedores
- Tratar los eventos

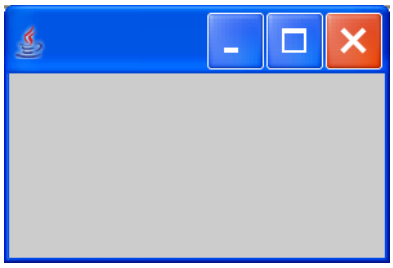
2. Crear una ventana

Se puede crear una ventana (que servirá como contenedor de componentes) utilizando la clase JFrame.

El siguiente código muestra cómo se puede crear una ventana con tamaño 300 por 200 pixels.

```
import javax.swing.*;

public class EjemploVentana{
   public static void main(String[] args){
     Ventana ven = new Ventana();
     ven.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     ven.show();
   }
}

class Ventana extends JFrame{
   public Ventana(){
      setSize(300,200);
   }
}
```

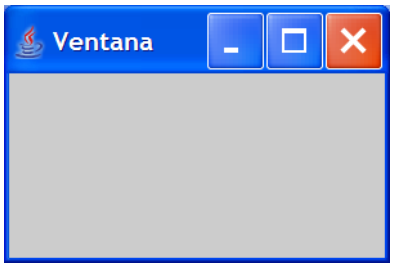
El resultado al ejecutar esta aplicación es el siguiente:



Si se desea que en la ventana aparezca un título se puede poner como primera sentencia en el constructor de Ventana la siguiente instrucción:

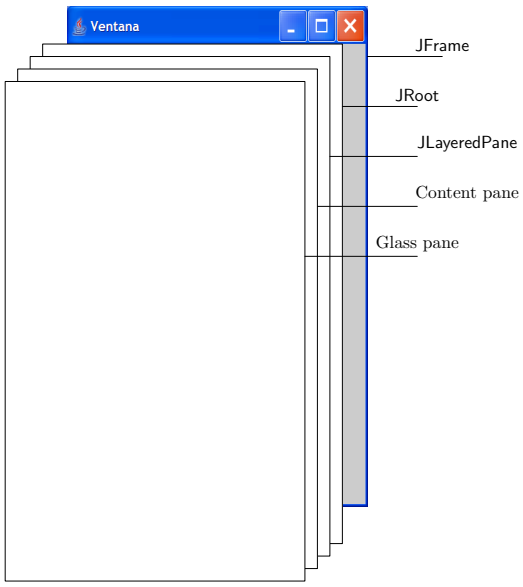
```
   super("Ventana");
```

En este caso se verá lo siguiente:



Las ventanas son contenedores de otros componentes tales como barras de menú, campos de texto, botones, etc.

De hecho una ventana está constituida por una serie de capas:



Para añadir componentes a la ventana, primero se debe obtener el contenedor content pane y a continuación añadir los componentes a éste.

Por ejemplo, supongamos que deseamos dibujar un rectángulo en la ventana. El rectángulo no se puede dibujar directamente sobre un objeto del tipo JFrame. En su lugar procederemos del siguiente modo:

1. Crearemos una clase que extienda a JPanel
2. Sobreescribiremos su método paintComponent(Graphics g)
3. Añadiremos un objeto de este tipo a la ventana.

El código se muestra a continuación:

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

class MiPanel extends JPanel{
   public void paintComponent(Graphics g){
      super.paintComponent(g);
      g.drawRect(20,20,80,80);
   }
}

class Ventana extends JFrame{
   public Ventana(){
      getContentPane().add(new MiPanel());
      setSize(300,200);
   }
}


public class EjemploVentana2{
   public static void main(String[] args){
     Ventana ven = new Ventana();
```

```
        ven.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ven.show();
    }
}
```

Se ha sobreescrito un método de la clase JPanel donde especificamos qué se debe realizar cuando haya que mostrar este componente, pero... ¿donde llamamos a este método?.

La respuesta es: en ningún sitio. Este método es llamado automáticamente cada vez que hay que pintar el componente y esto ocurre cada vez que:

- Cuando hay que mostrarlo por primera vez
- Cuando se modifica el tamaño de la ventana
- Cuando la ventana estaba minimizada y se vuelve a mostrar.
- Cuando otra aplicación que cubre a la ventana se mueve.
- ...

3. Componentes swing

Un componente es un objeto que tiene una representación gráfica y que puede ser mostrado por pantalla y que puede utilizado por el usuario. Ejemplos de componentes son: JButton, JTextField, JScrollPane, JTextArea, [1]

Utilizan como base la clase java.awt.Component que está definida como abstracta. Todos los componentes (excepto los menús) extienden a esta clase.

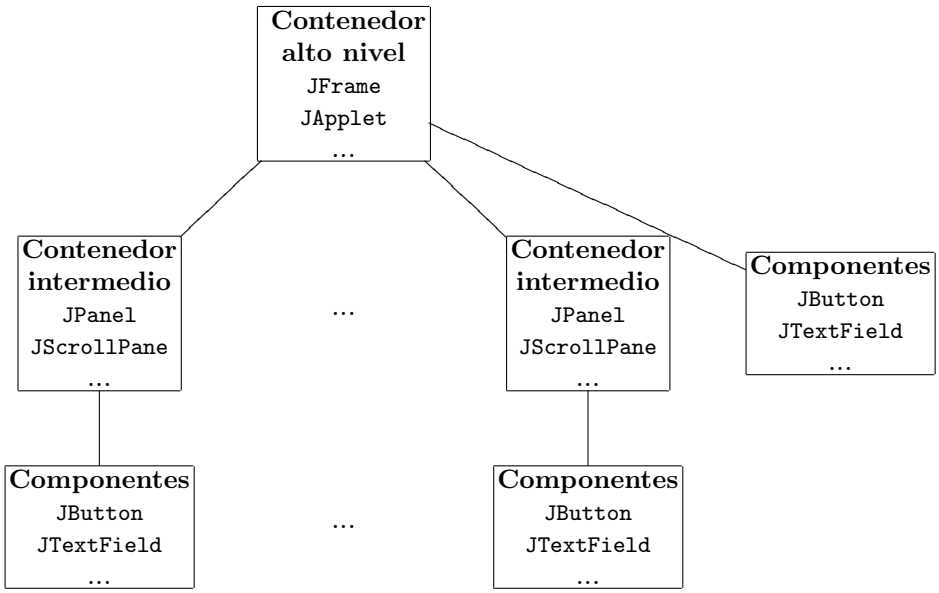
Los componentes se pueden dividir en dos categorías:

Un conjunto de componentes está formado por widgets [2].

Otro conjunto está formado por contenedores. Estos componentes extienden a la clase java.awt.Container (que es una clase abstracta que extiende a Component). Los contenedores son componentes que pueden incluir otros componentes.

La siguiente figura muestra la relación entre componentes y contenedores

[1] Hay otra serie de clases que no empiezan por J: Button, TextField, TextArea,... que pertenecen a AWT.
[2] Contracción de Window y gadget. Una representación visible de un componente que puede ser manipulada por el usuario. Botones, campos de texto y barras de desplazamiento son ejemplos de *widgets*



Vamos a ver algunos de los componentes que ofrece Swing.

3.1. JPanel y JLabel

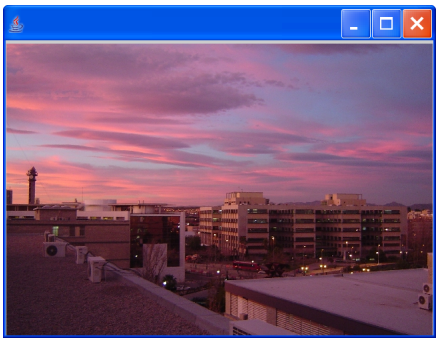
Un objeto de la clase JPanel sirve para contener otros componentes. La clase JLabel se utiliza para crear etiquetas de texto.

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

public class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      JPanel p = new JPanel();
      p.add(new JLabel("Ejemplo de JPanel"));

      c.add(p);

      setSize(200,200);
      setVisible(true);
   }
   public static void main(String[] args){
      new Ventana();
   }
}
```

3.2. ImageIcon

Objetos de esta clase se pueden utilizar para mostrar imágenes.

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

public class Ventana extends JFrame{
   public Ventana(String fich){
      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      ImageIcon ii = new ImageIcon(fich);

      c.add(new JLabel("",ii,JLabel.CENTER));

      setSize(650,500);
      setVisible(true);
   }
   public static void main(String[] args){
      new Ventana(args[0]);
   }
}
```



3.3. JTextField

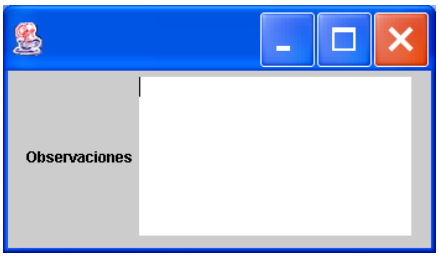
Objetos de esta clase se utilizan para que el usuario pueda introducir datos a la aplicación.

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

public class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      JTextField campoTexto = new JTextField(20);

      c.add(new JLabel("Nombre"));
      c.add(campoTexto);

      setSize(350,200);
      setVisible(true);
   }
   public static void main(String[] args){
      new Ventana();
   }
}
```

3.4. JTextArea

Objetos de esta clase se utilizan para que el usuario pueda introducir datos tipo texto de gran tamaño.

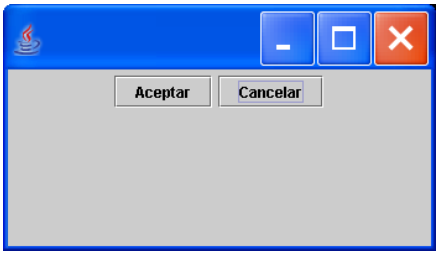
```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;

public class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      JTextArea area = new JTextArea(8,20);

      c.add(new JLabel("Observaciones"));
      c.add(area);

      setSize(350,200);
      setVisible(true);
   }
   public static void main(String[] args){
      new Ventana();
   }
}
```



3.5. JButton

Un objeto de esta clase representa un botón.

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

public class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      JButton b1 = new JButton("Aceptar");
      JButton b2 = new JButton("Cancelar");

      c.add(b1);
      c.add(b2);

      setSize(350,200);
      setVisible(true);
   }
   public static void main(String[] args){
      new Ventana();
   }
}
```



3.6. JCheckBox

Sirve para seleccionar elementos.

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

public class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      JCheckBox cb = new JCheckBox("Pizarra");
      cb.setFont(new Font("Arial",Font.PLAIN,20));

      c.add(cb);

      setSize(200,200);
      setVisible(true);
   }
   public static void main(String[] args){
      new Ventana();
   }
}
```

3.7. JRadioButton

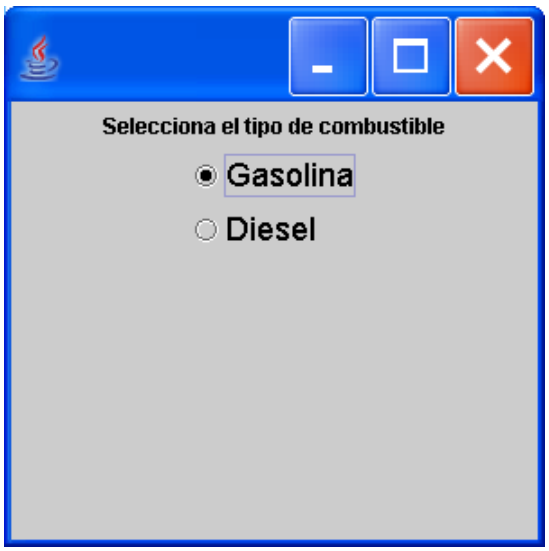
Sirve para seleccionar elementos.

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

public class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      JRadioButton rb=new JRadioButton("Pizarra");

      rb.setFont(new Font("Arial",Font.PLAIN,20));

      c.add(rb);

      setSize(200,200);
      setVisible(true);
   }
   public static void main(String[] args){
      new Ventana();
   }
}
```

3.8. ButtonGroup

Se pueden agrupar una serie de JRadioButton de forma que sólo pueda estar seleccionado uno de ellos.

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

public class Ventana extends JFrame{

   public Ventana(){
      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      c.add(new JLabel("Selecciona el tipo de combustible"));

      Font fuente = new Font("Arial",Font.PLAIN,18);

      JRadioButton gas = new JRadioButton("Gasolina");
      gas.setFont(fuente);

      JRadioButton die = new JRadioButton("Diesel");
      die.setFont(fuente);

      // Agrupamos los botones
      ButtonGroup grupo = new ButtonGroup();
      grupo.add(gas);
      grupo.add(die);

      JPanel radioPanel = new JPanel();
      radioPanel.setLayout(new GridLayout(0, 1));
      radioPanel.add(gas);
      radioPanel.add(die);

      c.add(radioPanel);

      setSize(300,300);
      setVisible(true);
   }
   public static void main(String[] args){
      new Ventana();
   }
}
```



3.9. JComboBox

Sirve para mostrar una lista desplegable de elementos.

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

public class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      JComboBox cb = new JComboBox();
      cb.setFont(new Font("Arial",Font.PLAIN,20));
      cb.addItem("Pizarra");
      cb.addItem("Pantalla");
      cb.addItem("Proyector");

      c.add(cb);
      setSize(200,200);
      setVisible(true);
   }
   public static void main(String[] args){
      new Ventana();
   }
}
```

3.10. JList

Objetos de esta clase sirven para mostrar una lista con elementos.

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

public class Ventana extends JFrame{

    public Ventana(){
       Container c = getContentPane();
       c.setLayout(new FlowLayout());

       String[] datos = {"Pizarra", "Pantalla", "
           Proyector"};
       JList lista = new JList(datos);

       c.add(lista);

       setSize(200,200);
       setVisible(true);
    }
    public static void main(String[] args){
       new Ventana();
    }
}
```

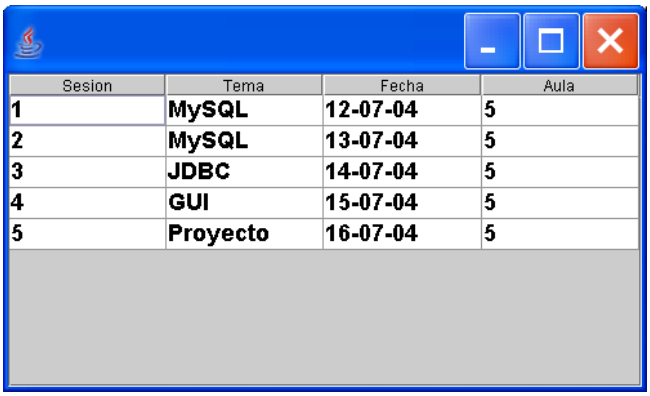
3.11. JTable y JScrollPane

Objetos del tipo JTable sirven para mostrar información en forma tabular.

Los objetos del tipo JScrollPane sirven para contener componentes y mostrar barras de desplazamiento.

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

public class Ventana extends JFrame{
   public Ventana(){

      Container cp = getContentPane();
      cp.setLayout(new BorderLayout());

      // Nombres de las columnas
      final String[] nombreCol = {"Sesion","Tema","Fecha","Aula"};

      // Datos
      Object[][] datos = {
       {"1","MySQL","12-07-04","5"},
       {"2","MySQL","13-07-04","5"},
       {"3","JDBC","14-07-04","5"},
       {"4","GUI","15-07-04","5"},
       {"5","Proyecto","16-07-04","5"}};

      JTable tabla = new JTable(datos,nombreCol);

      tabla.setFont(new Font("Arial",Font.BOLD,18));
      tabla.setRowHeight(24);

      JScrollPane jsp = new JScrollPane(tabla); //,ver,hor);

      cp.add(jsp,BorderLayout.CENTER);

      setSize(500,300);
      setVisible(true);
```

```
    }

    public static void main(String[] args){
        new Ventana();
    }
}
```



Ejercicio 1
Realizar una consulta en la que se muestren las familias del Reino Unido donde el precio de estancia por día sea menor o igual a 18 euros, se debe seleccionar el nombre de la familia, la ciudad y el tipo de casa.
El resultado se debe mostrar en una JTable.

3.12. JTree

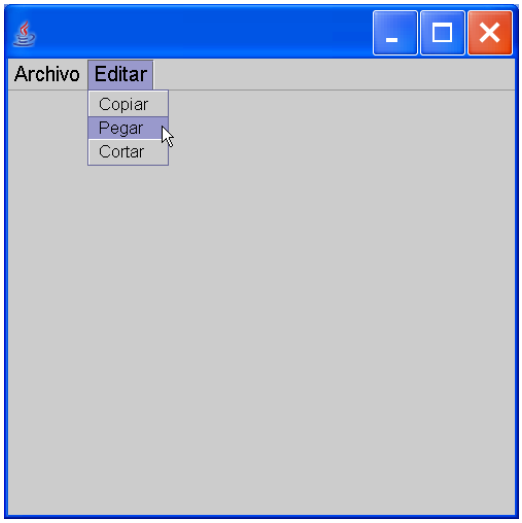
Objetos de este tipo sirven para mostrar la información en forma de árbol.

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

public class Ventana extends JFrame{
    private JTree arbol;
    public Ventana(){


        Container c = getContentPane();
        c.setLayout(new BorderLayout());

        // Construccion del arbol

        DefaultMutableTreeNode asig = new DefaultMutableTreeNode("Enlaces");

        DefaultMutableTreeNode tema = null;
        DefaultMutableTreeNode seccion = null;

        tema = new DefaultMutableTreeNode("Buscadores");
        asig.add(tema);

        seccion = new DefaultMutableTreeNode("Google");
```

```
        tema.add(seccion);

        seccion = new DefaultMutableTreeNode("Yahoo");
        tema.add(seccion);


        tema = new DefaultMutableTreeNode("Java");
        asig.add(tema);

        seccion = new DefaultMutableTreeNode("Sun");
        tema.add(seccion);

        seccion = new DefaultMutableTreeNode("IBM");
        tema.add(seccion);

        seccion = new DefaultMutableTreeNode("JavaWorld");
        tema.add(seccion);


        arbol = new JTree(asig);

        arbol.setFont(new Font("Arial",Font.BOLD,20));

        c.add(arbol,BorderLayout.CENTER);

        setSize(400,600);
        setVisible(true);
    }
    public static void main(String[] args){
        new Ventana();
    }
}
```

3.13. JMenu

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

class Ventana extends JFrame{
   private JMenuBar mb;

   Ventana(){

      // Se crea una barra de menús
      mb = new JMenuBar();

      // Creamos un elemento del menú
      JMenu archivo = new JMenu("Archivo");
      archivo.setFont(new Font("Arial",Font.PLAIN,20));

      // Creamos y añadimos submenús

      JMenuItem nuevo = new JMenuItem("Nuevo");
      nuevo.setFont(new Font("Arial",Font.PLAIN,16));
      archivo.add(nuevo);

      JMenuItem abrir = new JMenuItem("Abrir");
      abrir.setFont(new Font("Arial",Font.PLAIN,16));
      archivo.add(abrir);

      JMenuItem ver = new JMenuItem("Ver todos");
      ver.setFont(new Font("Arial",Font.PLAIN,16));
      archivo.add(ver);

      // Ahora añadimos archivo a la barra de menus
      mb.add(archivo);

      // Creamos otro elemento del menú
      JMenu editar = new JMenu("Editar");
      editar.setFont(new Font("Arial",Font.PLAIN,20));

      // Creamos y añadimos submenús

      JMenuItem copiar = new JMenuItem("Copiar");
      copiar.setFont(new Font("Arial",Font.PLAIN,16));
      editar.add(copiar);

      JMenuItem pegar = new JMenuItem("Pegar");
      pegar.setFont(new Font("Arial",Font.PLAIN,16));
      editar.add(pegar);

      JMenuItem cortar = new JMenuItem("Cortar");
      cortar.setFont(new Font("Arial",Font.PLAIN,16));
      editar.add(cortar);

      // Añadimos editar a la barra de menu
      mb.add(editar);


      setJMenuBar(mb);

      setSize(500,500);
      setVisible(true);

   }

   public static void main(String[] args){
      new Ventana();
   }
}
```



4. Organización de los componentes

Cuando en una ventana hay muchos componentes hay que organizarlos de algún modo.

Java proporciona diversos esquemas de organización (*layout managers*) que pueden ser utilizados para organizar los componentes dentro de los contenedores.

Los gestores de organización se encargan de reorganizar los componentes en caso de que el usuario cambie el tamaño de la ventana.

Los gestores de organización que ofrece Java son:

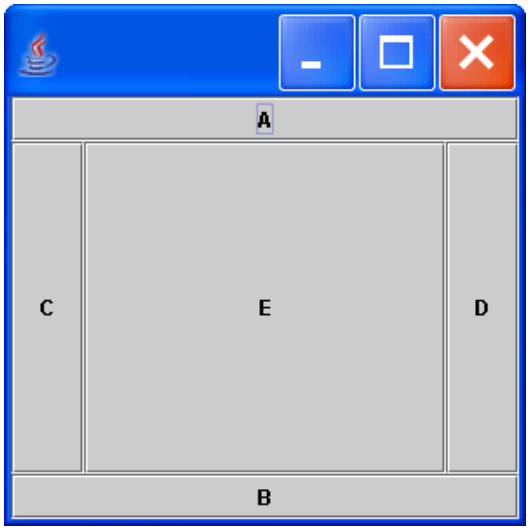
BorderLayout, FlowLayout, BoxLayout, CardLayout, GridLayout, GridBagLayout

El procedimiento es siempre el mismo, se crea un objeto de alguna de estas clases y se le indica al contenedor que organice los componentes utilizando el objeto (para ello los contenedores disponen del método setLayout(LayoutManager m).

4.1. BorderLayout

Se puede utilizar para colocar en un contenedor cinco componentes como máximo ya que proporciona cinco posiciones donde colocar los componentes, estas son: NORTH (arriba), SOUTH (abajo), WEST (izquierda), EAST (derecha) y CENTER (en el centro).

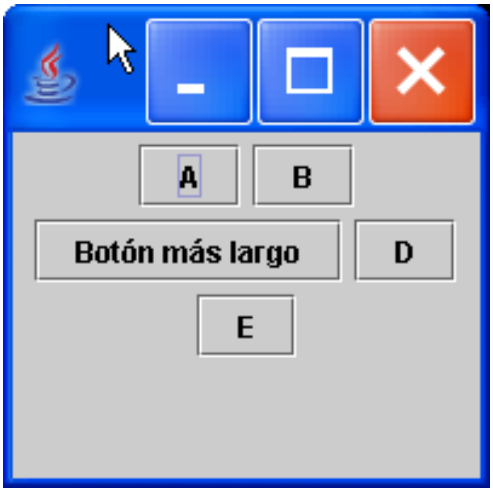
```
import java.awt.*;
import javax.swing.*;

class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();

      JButton b1 = new JButton("A");
      JButton b2 = new JButton("B");
      JButton b3 = new JButton("C");
      JButton b4 = new JButton("D");
      JButton b5 = new JButton("E");

      c.setLayout(new BorderLayout());

      c.add(b1,BorderLayout.NORTH);
      c.add(b2,BorderLayout.SOUTH);
      c.add(b3,BorderLayout.WEST);
      c.add(b4,BorderLayout.EAST);
      c.add(b5,BorderLayout.CENTER);
   }
   public static void main(String[] args){
      Ventana v = new Ventana();
      v.setSize(300,300);
      v.show();
   }
}
```



4.2. FlowLayout

Coloca los componentes de izquierda a derecha conforme se van añadiendo a la ventana. El tamaño de los componentes se ajusta a su contenido.

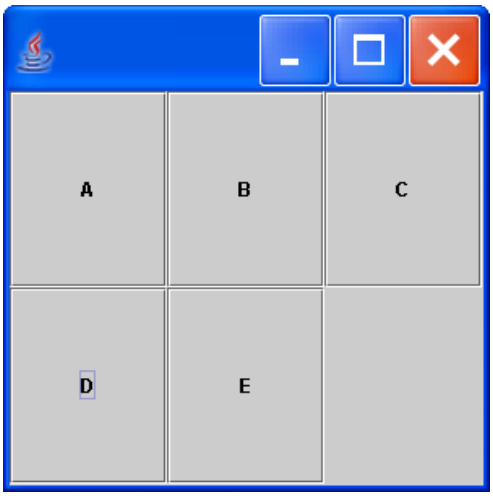
```
import java.awt.*;
import javax.swing.*;

class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();

      JButton b1 = new JButton("A");
      JButton b2 = new JButton("B");
      JButton b3 = new JButton("Botón más largo");
      JButton b4 = new JButton("D");
      JButton b5 = new JButton("E");

      c.setLayout(new FlowLayout());
      c.add(b1);
      c.add(b2);
      c.add(b3);
      c.add(b4);
      c.add(b5);
   }

   public static void main(String[] args){
      Ventana v = new Ventana();
      v.setSize(200,200);
      v.show();
   }
}
```



4.3. GridLayout

Coloca los componentes en filas y columnas en función de los valores pasados al constructor. Todas las celdas tendrán el mismo tamaño.

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

class Ventana extends JFrame{
   public Ventana(){
      Container c = getContentPane();
      JButton b1 = new JButton("A");
      JButton b2 = new JButton("B");
      JButton b3 = new JButton("C");
      JButton b4 = new JButton("D");
      JButton b5 = new JButton("E");

      c.setLayout(new GridLayout(2,3));
      c.add(b1);
      c.add(b2);
      c.add(b3);
      c.add(b4);
      c.add(b5);
   }
   public static void main(String[] args){
      Ventana v = new Ventana();
      v.setSize(300,300);
      v.show();
   }
}
```



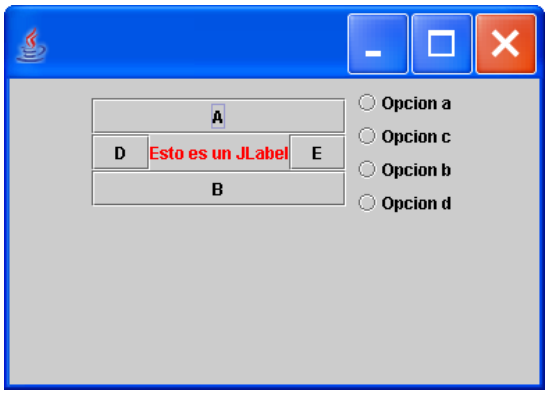
4.4. CardLayout

Se puede utilizar para mostrar de forma condicional unos elementos u otros, de forma que se puede controlar qué elementos serán visibles. Una ilustración es una pila de cartas en las que sólo la superior es visible en un instante dado.

4.5. GridBagLayout

Este es un organizador complejo que permite ajustar la posición de los componentes. Al colocar los componentes en los contenedores se especifican las restricciones que se deben cumplir (posición del componente, anchura y altura del componente, separación entre los componentes, posición dentro del espacio que ocupa, ...).

Ejercicio 2
Se pide construir la interfaz gráfica que se muestra en la siguiente figura.



5. Tratamiento de eventos

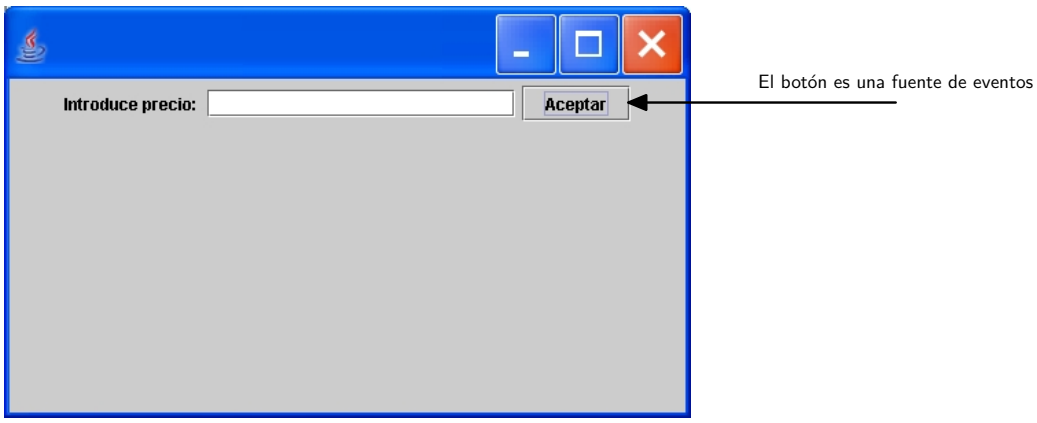
Hasta ahora las interfaces gráficas que se han mostrado tienen poca utilidad ya que no responden a las acciones del usuario.

¿Qué ocurre si pulsamos sobre un botón?

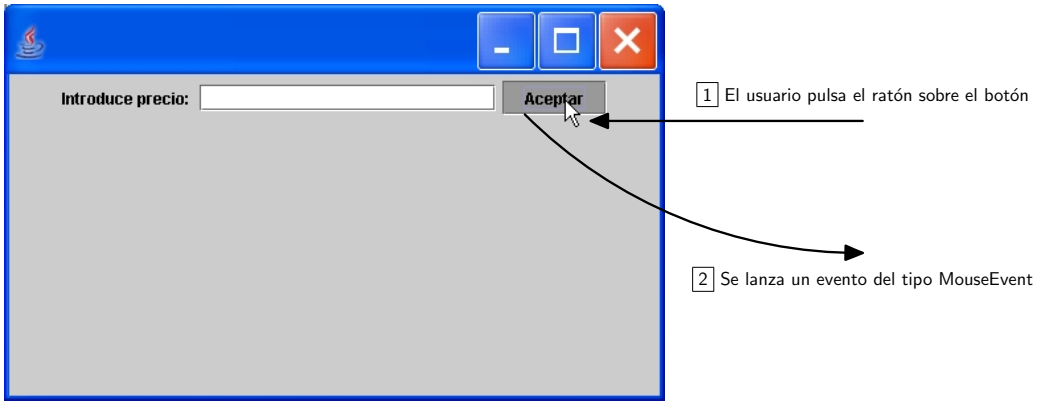
¿Qué ocurre si pulsamos sobre una celda de la tabla?

¿Qué ocurre si pulsamos sobre un elemento de un menú?

Pues con lo que hemos hecho hasta ahora, aparentemente no sucede nada, no se obtiene ninguna respuesta.



En una GUI colocamos una serie de componentes entre los cuales se encuentra un JButton. Este componente es una fuente de eventos de ratón.



Si el usuario pulsa con el ratón sobre el botón se lanza un evento del tipo MouseEvent. Si no hay ningún objeto que recoja ese evento no sucede nada.

¿Qué es lo que falta?

Falta especificar qué es lo que se debe realizar cuando se produzcan determinados eventos sobre los componentes que se coloquen en la ventana.

Esta tarea es la que se conoce como tratamiento de eventos.

Cualquier sistema operativo que soporte GUI's monitoriza los eventos que se producen, como por ejemplo pulsaciones sobre las teclas o pulsaciones con un botón del ratón.

El sistema operativo informa sobre estos eventos a los programas que están en ejecución.

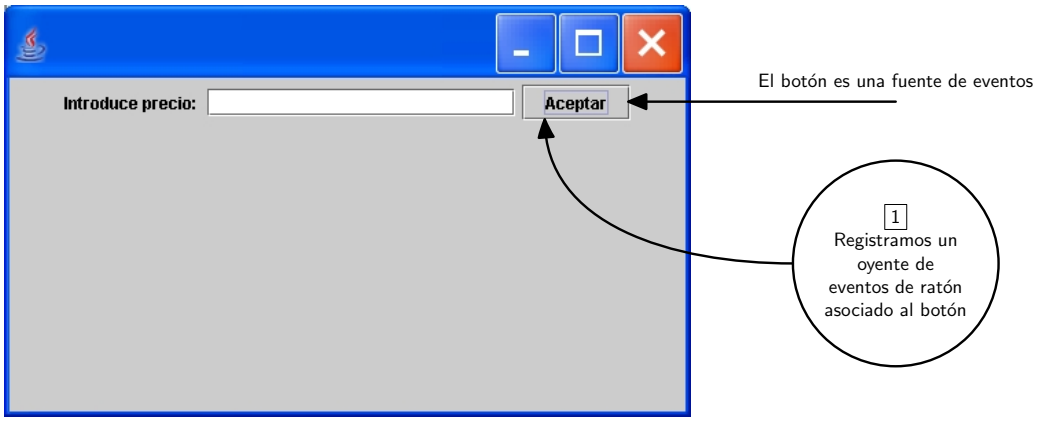
Cada programa decide qué hacer (si es que debe hacer algo) en respuesta a estos eventos.

En Java se utiliza un modelo conocido como *modelo de delegación de eventos*. s

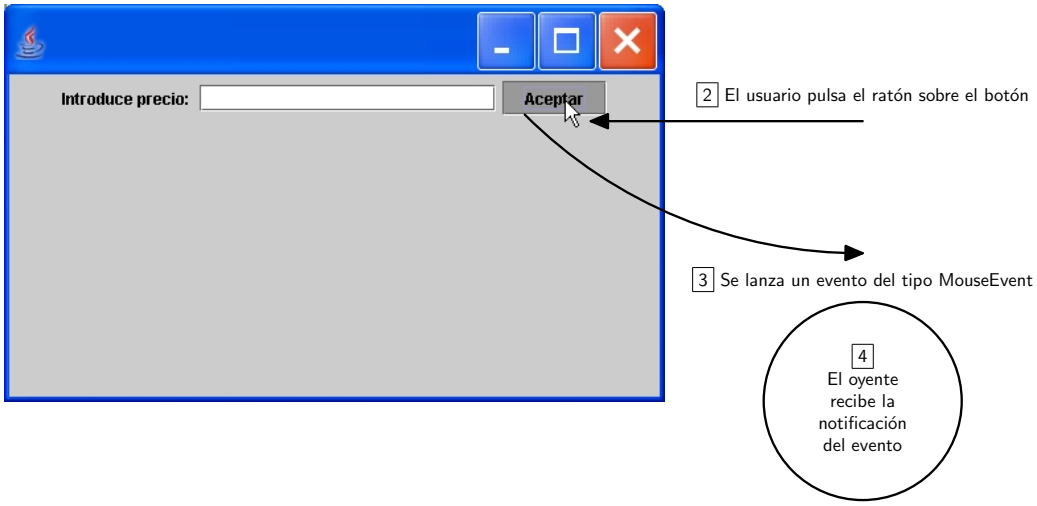
El modelo de delegación de eventos se basa en que los componentes disparan eventos que pueden ser tratados por escuchadores (o manipuladores).

Los escuchadores se registran en un componente. Por ejemplo, en un botón podemos registrar un escuchador de eventos de ratón.

Una vez que el escuchador ha sido añadido al componente, cuando se produzca un evento, los métodos apropiados del manipulador (que han sido especificados en la interfaz) serán llamados.



En una GUI colocamos una serie de componentes entre los cuales se encuentra un JButton. Este componente es una fuente de eventos de ratón. Ahora registramos un oyente de eventos de ratón en el botón.



Si el usuario pulsa con el ratón sobre el botón se lanza un evento del tipo MouseEvent. Los oyentes que se hayan registrados son notificados de que se ha producido un evento.

La clase EventObject del paquete java.util es la clase padre de todos los eventos.

Su constructor recibe una referencia al objeto que genera el evento.

Esta clase tiene dos métodos: getSource() que devuelve el objeto que generó el evento y toString().

Los paquetes relacionados con los eventos en AWT son java.awt.event.

La clase abstracta AWTEvent definida en el paquete java.awt es una subclase de EventObject.

Es la superclase de todos los eventos basados en AWT utilizados por el modelo de delegación de eventos.

Hay dos clases de eventos:

- **Eventos de componente o de bajo nivel** ocurren cuando ocurre algo específico en un componente. Por ejemplo al moverse, entrar o salir el ratón sobre un componente, al ganar o perder la atención,...

- **Eventos semánticos** no son tan específicos como los anteriores y no son disparados necesariamente por una acción atómica tal y como una pulsación del ratón. Las acciones que disparan estos eventos depende del objeto: por ejemplo en una lista se disparan cuando sus elementos son pulsados dos veces, en un campo de texto se disparan cuando se pulsa la tecla *enter*.

Los eventos de componente o de bajo nivel son:

ComponentEvent, ContainerEvent, FocusEvent, InputEvent, KeyEvent, MouseEvent, MouseWheelEvent y WindowEvent

Los eventos semánticos son:

ActionEvent ,AdjustmentEvent , ItemEvent, TextEvent

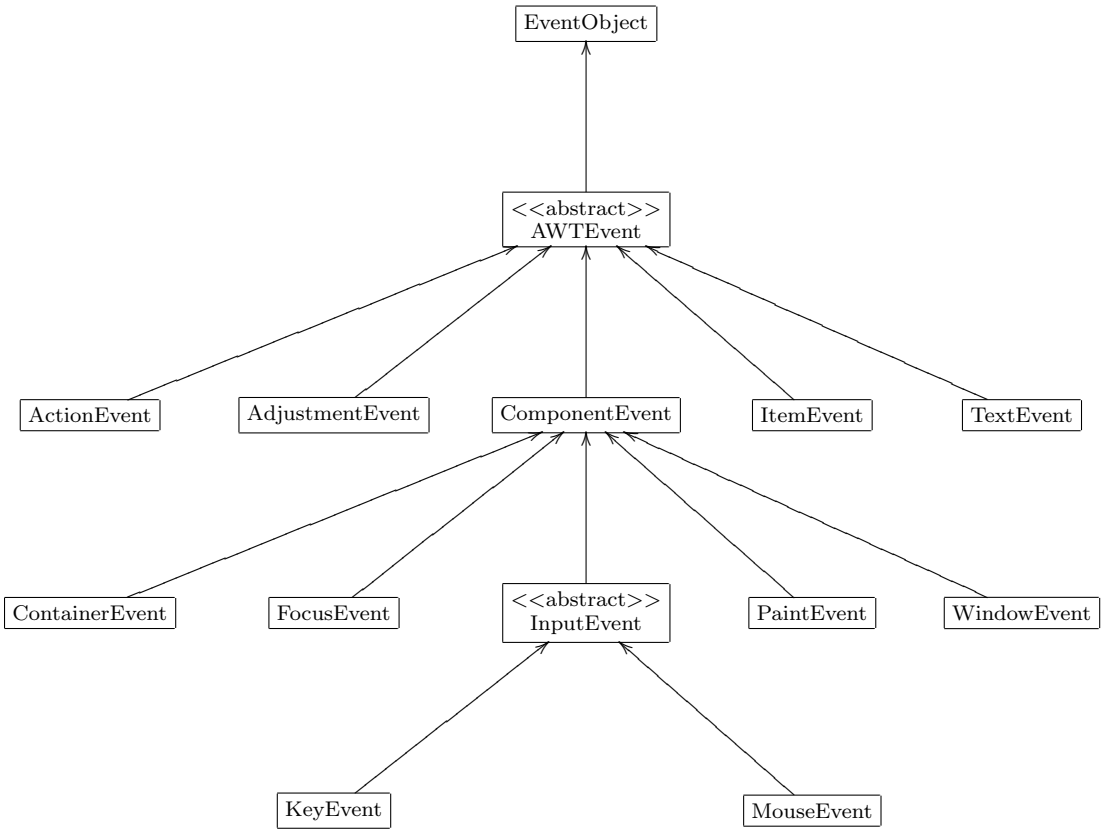
A continuación se muestran los eventos que se definen en AWT y cual es la acción que los produce

Eventos AWT	**Descripción**
`ActionEvent`	Se genera cuando el usuario pulsa un botón, pulsa *Return* en un campo de texto, selecciona un elemento de un menú o cuando un elemento de una lista se pulsado 2 veces.
`AdjustmentEvent`	Se genera cuando se manipula una barra de deslizamiento.
`ItemEvent`	Evento que indica que un elemento de una lista se ha seleccionado o ha dejado de estar seleccionado. Los siguientes componentes generan eventos de este tipo: `CheckBox, CheckBoxMenuItem, Choice, List`.
`TextEvent`	Se genera cuando se cambia el valor de un área de texto o de un campo de texto. Los objetos fuente de este evento son: `TextField` y `TextArea`.

Eventos AWT	Descripción
`ComponentEvent`	Un evento que indica que un componente ha sido movido, ha cambiado de tamaño o ha sido ocultado. AWT maneja este evento (es decir que aunque explícitamente tratemos este evento, AWT también lo hará).
`ContainerEvent`	Se genera cuando se añade o se elimina un componente de un contenedor. AWT trata este evento.
`FocusEvent`	Se genera cuando un componente gana o pierde la atención. Un componente tiene la atención al pulsar sobre él con el ratón o por que se ha llegado a él pulsando la tecla de tabulación. El componente que tiene la atención recibe los eventos de teclado.

Eventos AWT	Descripción
`KeyEvent`	Es una subclase de `InputEvent`. Se genera cuando se pulsa una tecla o libera una tecla.
`MouseEvent`	Es una subclase de `InputEvent`. Se genera cuando el ratón se mueve, se pulsa, se arrastra, o cuando entra o sale el ratón de un componente.
`MouseWheelEvent`	Un evento que indica que la rueda del ratón se ha movido en un componente.
`WindowEvent`	Se genera cuando una ventana se activa, se desactiva, se cierra, se minimiza se maximiza o se sale de ella.

La jerarquía de estas clases se muestra en el siguiente diagrama:



Vamos a ver ahora algunas de las interfaces que se ofrecen en `java.awt` con el fin de especificar los métodos que deben poseer los objetos oyentes o auditores para cada uno de los eventos.

5.1. MouseListener

```
public interface MouseListener extends EventListener
```

Esta interfaz la deben implementar aquellas clases que estén interesadas en escuchar eventos del tipo MouseEvent

El objeto de esta clase debe registrarse en un componente utilizando su método addMouseListener.

Los métodos que define esta interfaz son:

```
// Metodo llamado cuando se pulsa y libera un boton del raton
// sobre un componente
void mouseClicked(MouseEvent e)

// Metodo llamado cuando el raton entra en un componente
void mouseEntered(MouseEvent e)

// Metodo llamado cuando el raton sale de un componente
void mouseExited(MouseEvent e)

// Metodo llamado al pulsar un boton del raton sobre un componente
void mousePressed(MouseEvent e)

// Metodo llamado al liberar un boton del raton sobre un componente
void mouseReleased(MouseEvent e)
```

En el siguiente ejemplo se trata el evento: *el usuario ha pulsado con el ratón sobre el botón*. Lo que se realiza en el manipulador del evento es obtener lo que el usuario haya escrito en el campo de texto y mostrarlo por la salida estándar.

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

class EventosRaton extends JFrame{
   private JButton boton;
   private JTextField campoTexto;

   public EventosRaton(){

      class ManipulaMouseEvent implements MouseListener{
         public void mouseEntered(MouseEvent e) {}
         public void mouseExited(MouseEvent e) {}
         public void mouseClicked(MouseEvent e){}
         public void mouseReleased(MouseEvent e){}
         public void mousePressed(MouseEvent e){
            System.out.println(campoTexto.getText());
         }
      }

      Container cp = getContentPane();

      cp.setLayout(new FlowLayout());
```

```
        cp.add(new JLabel("Introduce precio: "));

        campoTexto = new JTextField(20);

        cp.add(campoTexto);

        boton = new JButton("Aceptar");

        cp.add(boton);

        boton.addMouseListener(new ManipulaMouseEvent());

        setSize(500,300);
    }

    public static void main(String[] args){
      EventosRaton ven = new EventosRaton();
      ven.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      ven.show();
    }
}
```

A continuación se muestra un ejemplo en el que se realiza una consulta a la base de datos utilizando un dato introducido por el usuario en un campo de texto.

La consulta a la base de datos es la del ejercicio 2 de la sesión de JDBC pero ahora se utiliza un PreparedStatement.

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

/** Clase de apoyo. En ella encapsulamos la conexion a la base de datos y
  * las consultas
  */
class Consultas{

    private Connection con;
    private PreparedStatement ps;

    // Constructor
    Consultas(){

        String url = "jdbc:mysql://localhost:3306/estancias?user=usuario&password=clave";

        try{

            Class.forName("com.mysql.jdbc.Driver");

            con = DriverManager.getConnection(url);

            ps = con.prepareStatement("select familias.NombreF, casas.Ciudad, casas.Tipo from
                familias,casas where (familias.IdCasa=casas.IdCasa) and (casas.precioHabDia <= ?)
                ");

        }catch(SQLException ex){
            ex.printStackTrace();
        }catch(ClassNotFoundException ex){
            ex.printStackTrace();
        }
    }

    /** Metodo que realiza una consulta a la base de datos y devuelve un
      * el ResultSet con el resultado.
      *@param valor un float necesario para construir la consulta
```

```
     *@return un objeto del tipo ResultSet con el resultado de la consulta
     */
   public ResultSet consultaCasas(float valor){

      ResultSet resultado = null;

      try{

         ps.setFloat(1,valor);
         resultado = ps.executeQuery();
      }catch(SQLException ex){
         ex.printStackTrace();
      }

      return resultado;
   }

   /** Metodo para cerrar la conexion con la base de datos */
   public void cierraConexion(){
      try{
         con.close();
      }catch(SQLException ex){
         ex.printStackTrace();
      }
   }
}


/** Esta es la ventana */
class ConsultaGUI extends JFrame{
   private JButton boton;
   private JTextField campoTexto;
   private Consultas cons = new Consultas();

   public ConsultaGUI(){

      // Creamos una clase para manipular eventos de raton
      class ManipulaMouseEvent implements MouseListener{
         public void mouseEntered(MouseEvent e) {}
         public void mouseExited(MouseEvent e) {}
         public void mouseClicked(MouseEvent e){}
         public void mouseReleased(MouseEvent e){}
         public void mousePressed(MouseEvent e){

            String parametro = campoTexto.getText();

            try{
               float precio = Float.parseFloat(parametro);

               System.out.println("\n Resultado de la busqueda:");

               ResultSet r = cons.consultaCasas(precio);

               r.beforeFirst();

               while (r.next()){
                  System.out.print("Familia: " + r.getString(1) + "\t");
                  System.out.print("Ciudad:  " + r.getString(2) + "\t");
                  System.out.print("Tipo:    " + r.getString(3) + "\n");
               }

            }catch(NumberFormatException ex){
               System.out.println("No es un número valido");
            }catch(SQLException ex){
               ex.printStackTrace();
            }

         }
      }
```

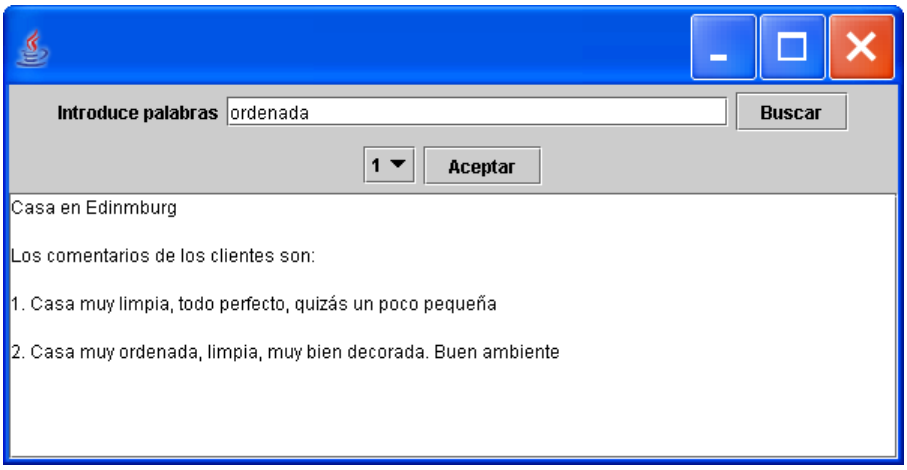
```
        // Se obtiene el content pane de la ventana
        Container cp = getContentPane();

        // Indicamos la forma de organizar los componentes en el
        // content pane
        cp.setLayout(new FlowLayout());

        // Se añade una etiqueta de texto
        cp.add(new JLabel("Introduce precio: "));

        // Se crea y añade un campo de texto
        campoTexto = new JTextField(20);
        cp.add(campoTexto);

        // Se crea y añade un boton
        boton = new JButton("Aceptar");
        cp.add(boton);

        // Registramos un objeto de ManipulaMouseEvent como un escuchador
        // de eventos de raton para el boton
        boton.addMouseListener(new ManipulaMouseEvent());

        setSize(500,300);
    }

    public static void main(String[] args){
      ConsultaGUI ven = new ConsultaGUI();
      ven.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      ven.show();
    }
}
```

Ejercicio 3

En una de las tablas se podían introducir comentarios de los clientes sobre las casas. Se desea realizar una GUI para realizar buscar por una palabra (o palabras) dentro del comentario. La GUI debe contener una etiqueta, un campo de texto donde introducir la(s) palabra(s) y un botón. El resultado de esta búsqueda serán las casas que tienen un comentario que incluya la(s) palabra(s) introducidas.

Este resultado se inserta en un JComboBox de forma que el usuario pueda seleccionar una casa. A su lado aparecerá un botón para lanzar una búsqueda concreta para ver todos los comentarios concernientes a la casa seleccionada. Los comentarios deben aparecer en un área de texto.

La siguiente figura muestra un posible diseño y su ejecución.



5.2. KeyListener

```
public interface KeyListener extends EventListener
```

Esta interfaz la deben implementar aquellas clases que estén interesadas en escuchar eventos del tipo KeyEvent

El objeto de esta clase debe registrarse en un componente utilizando su método addKeyListener.

Los métodos que define esta interfaz son:

```
//Metodo llamado cuando se pulsa una tecla
void keyPressed(KeyEvent e)

//Metodo llamado cuando se libera una tecla
void keyReleased(KeyEvent e)

//Metodo llamado cuando se pulsa y libera una tecla
void keyTyped(KeyEvent e)
```

En el siguiente ejemplo, cada vez que se pulsa una tecla sobre un área de texto se trata el evento que se lanza. Se realiza una estadística para comprobar el número de letras y números frente al uso de otras teclas.

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


class EventosTeclado extends JFrame{
   private JTextArea a;
   private int contadorLetras=0;
   private int contadorOtros=0;
   private long t1;
   private long t2;

   EventosTeclado(){
      a = new JTextArea();

      class ManipulaKeyEvent implements KeyListener{
         public void keyPressed(KeyEvent e) {
            char car;
            car = e.getKeyChar();
            if ( !(Character.isLetter(car)) & !(Character.isDigit(car))){
               String tecla = e.getKeyText(e.getKeyCode());
               if (tecla.compareTo("Retroceso")==0)
                  contadorLetras--;
               System.out.print(tecla);
               contadorOtros++;
            }
         }

         public void keyReleased(KeyEvent e) {}

         public void keyTyped(KeyEvent e) {
            char car;
            car = e.getKeyChar();
            if ( (Character.isLetter(car)) | (Character.isDigit(car))){
               System.out.print(car);
```

```
                contadorLetras++;
            }
        }
    }

    class ManipulaMouseEventInicio implements MouseListener{
        public void mouseEntered(MouseEvent e) {}
        public void mouseExited(MouseEvent e) {}
        public void mouseClicked(MouseEvent e){}
        public void mouseReleased(MouseEvent e){}
        public void mousePressed(MouseEvent e){
            contadorLetras=0;
            contadorOtros=0;

            t1 = Calendar.getInstance().getTimeInMillis();
        }
    }

    class ManipulaMouseEventFin implements MouseListener{
        public void mouseEntered(MouseEvent e) {}
        public void mouseExited(MouseEvent e) {}
        public void mouseClicked(MouseEvent e){}
        public void mouseReleased(MouseEvent e){}
        public void mousePressed(MouseEvent e){

            a.setText("");

            t2 = Calendar.getInstance().getTimeInMillis();

            long tiempo = t2-t1;

            System.out.println("\nLetras y numeros: " + contadorLetras);
            System.out.println("Otras teclas: " + contadorOtros);
            System.out.println("Tiempo (milisegundos): " + tiempo);
         }
    }

    ManipulaKeyEvent mce = new ManipulaKeyEvent();
    a.addKeyListener(mce);

    getContentPane().add(a,BorderLayout.CENTER);

    JButton botonInicio = new JButton("Inicio");
    getContentPane().add(botonInicio,BorderLayout.NORTH);
    botonInicio.addMouseListener( new ManipulaMouseEventInicio());

    JButton botonFin = new JButton("Resultado");
    botonFin.addMouseListener( new ManipulaMouseEventFin());
    getContentPane().add(botonFin,BorderLayout.SOUTH);

    setSize(400,400);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    show();
  }

  public static void main(String[] args){
    new EventosTeclado();
  }
}
```

5.3. WindowListener

```
public interface WindowListener extends EventListener
```

Esta interfaz la deben implementar aquellas clases que estén interesadas en escuchar eventos del tipo WindowEvent

El objeto de esta clase debe registrarse en un componente utilizando su método addWindowListener.

Los métodos que define esta interfaz son:

```
// Metodo llamado cuando se activa la ventana
void windowActivated(WindowEvent e)

// Metodo llamado cuando se ha cerrado la ventana
void windowClosed(WindowEvent e)

// Metodo llamado cuando el usuario cierra la ventana
void windowClosing(WindowEvent e)

// Metodo llamado cuando la ventana deja de estar activa
void windowDeactivated(WindowEvent e)

// Metodo llamado cuando la ventana pasa de icono a su estado normal
void windowDeiconified(WindowEvent e)

// Metodo llamado cuando se iconifica la ventana
void windowIconified(WindowEvent e)

// Metodo llamado la primera vez que se muestra la ventana
void windowOpened(WindowEvent e)
```

5.4. ActionListener

```
public interface ActionListener extends EventListener
```

Esta interfaz la deben implementar aquellas clases que estén interesadas en escuchar eventos del tipo ActionEvent

El objeto de esta clase debe registrarse en un componente utilizando su método addActionListener.

Los métodos que define esta interfaz son:

```
// Metodo llamado cuando ocurre una accion
void actionPerformed(ActionEvent e)
```

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

class EventosAccion extends JFrame implements ActionListener{
   private TextField c1;
   private TextField c2;
   private Button b;

   EventosAccion(){

      // Se crea una barra de menús
      JMenuBar mb = new JMenuBar();
```

```
        // Creamos un elemento del menú
        JMenu archivo = new JMenu("Archivo");
        archivo.setFont(new Font("Arial",Font.PLAIN,20));

        // Creamos y añadimos submenús

        JMenuItem nuevo = new JMenuItem("Nuevo");
        nuevo.setFont(new Font("Arial",Font.PLAIN,16));
        nuevo.addActionListener(this);
        archivo.add(nuevo);


        JMenuItem abrir = new JMenuItem("Abrir");
        abrir.setFont(new Font("Arial",Font.PLAIN,16));
        abrir.addActionListener(this);
        archivo.add(abrir);

        mb.add(archivo);

        setJMenuBar(mb);

        setSize(300,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        show();

    }

    public void actionPerformed(ActionEvent e) {
        // Comprobamos si la fuente del evento es un JMenuItem
        if (e.getSource() instanceof JMenuItem){
            JMenuItem source = (JMenuItem)(e.getSource());
            String seleccionado = source.getText();

            // Si pulsa sobre abrir mostramos una ventana para abrir ficheros
            if (seleccionado.compareTo("Abrir")==0){
                JFileChooser pideFichero = new JFileChooser();
                int returnVal = pideFichero.showOpenDialog(EventosAccion.this);
                if(returnVal == JFileChooser.APPROVE_OPTION) {
                    System.out.println("Has seleccionado el fichero: " +
                    pideFichero.getSelectedFile().getName());
                }
            }
        }
    }

    public static void main(String[] args){
        new EventosAccion();
    }
}
```

5.5. TextListener

```
public interface TextListener extends EventListener
```

Esta interfaz la deben implementar aquellas clases que estén interesadas en escuchar eventos del tipo TextEvent

El objeto de esta clase debe registrarse en un componente utilizando su método addTextListener.

Los métodos que define esta interfaz son:

```
// Metodo llamado cuando el texto ha sido modificado
void textValueChanged(TextEvent e)
```

5.6. ItemListener

```
public interface ItemListener extends EventListener
```

Esta interfaz la deben implementar aquellas clases que estén interesadas en escuchar eventos del tipo ItemEvent

El objeto de esta clase debe registrarse en un componente utilizando su método addItemListener.

Los métodos que define esta interfaz son:

```
// Metodo llamado cuando el texto ha sido modificado
void itemStateChanged(ItemEvent e)
```

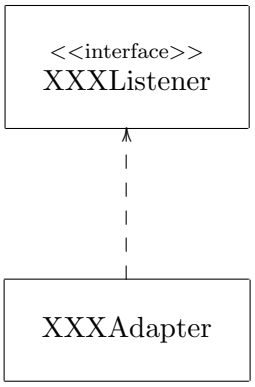
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class EventosItem extends JFrame{
   private JCheckBox check;
   private JComboBox comb;
   private boolean sel;
   private String val1;

   class ManipulaMouseEvent implements MouseListener{
      public void mousePressed(MouseEvent e){
         System.out.println(sel);
         System.out.println(val1);
      }
      public void mouseReleased(MouseEvent e){}
      public void mouseClicked(MouseEvent e) {}
      public void mouseEntered(MouseEvent e) {}
      public void mouseExited(MouseEvent e) {}
   }

   class ManipulaItemEvent implements ItemListener{
      public void itemStateChanged(ItemEvent e){

         if (e.getSource() instanceof JCheckBox){
            if (e.getStateChange() == ItemEvent.SELECTED)
               sel =true;
            else
               sel = false;
         }
         else // es que la fuente es el JComboBox
            val1 = (String)e.getItem();
      }
   }

   // El constructor
   EventosItem(){

      Container cp = getContentPane();
```

```
        check = new JCheckBox("A");

        comb = new JComboBox();
        comb.addItem("Casa");
        comb.addItem("Piso");
        comb.addItem("Adosado");

        JButton b = new JButton("Aceptar");

        check.addItemListener(new ManipulaItemEvent());
        comb.addItemListener(new ManipulaItemEvent());

        b.addMouseListener(new ManipulaMouseEvent());

        cp.setLayout(new FlowLayout());

        cp.add(check);
        cp.add(comb);
        cp.add(b);

        setSize(400,400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        show();
    }
    public static void main(String[] args){
        new EventosItem();
    }
}
```

Hay ocasiones en las que puede resultar incómodo trabajar con estas interfaces si uno está interesado en implementar un único método, ya que hay que escribir el resto de los métodos dejándolos vacíos.

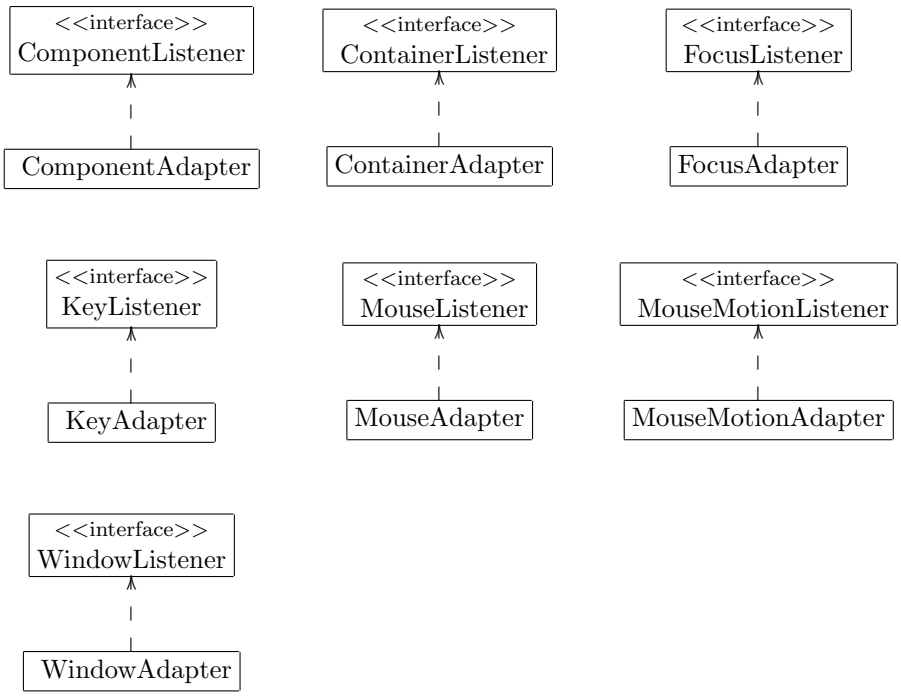
Para algunas de estas interfaces se ofrecen clases que las implementan pero dejando vacíos todos los métodos.

De esta forma, puede resultar más cómodo extender estas clases que implementar las interfaces.



donde XXX es el tipo del evento.

Las clases adaptadoras que se ofrecen son:



El código de la derecha y el de la izquierda son equivalentes:

```
Button b = new Button("Aceptar");

class Oyente implementes MouseListener{

    public void mouseClicked(MouseEvent e) {}

    public void mouseEntered(MouseEvent e) {}

    public void mouseExited(MouseEvent e) {}

    public void mousePressed(MouseEvent e) {
      System.out.println("Boton pulsado");
    }

    public void mouseReleased(MouseEvent e) {}
}

b.addMouseListener(new Oyente());
```

```
Button b = new Button("Aceptar");

class Oyente extends MouseAdapter{

    public void mousePressed(MouseEvent e) {
      System.out.println("Boton pulsado");
    }
}

b.addMouseListener(new Oyente());
```

La única diferencia a nivel de código es que en el caso de la derecha la clase implementa a la interfaz MouseListener y en el de la derecha la clase extiende a la clase MouseAdapter que a su vez implementa a la interfaz MouseListener.

Cuando una clase oyente de eventos se va a utilizar en un punto muy específico del código y no se va a reutilizar en otras partes (o en otras clases) existe la posibilidad de realizarla mediante una **clase anónima**.

Una clase anónima (como cabe esperar) es aquella a la que no se asigna un nombre.

La creación del objeto y la especificación de la clase se realiza en el mismo momento, el este caso en el argumento de un método.

Vamos a ver cómo se puede utilizar una clase anónima con el ejemplo anterior:

```
Button b = new Button("Aceptar");

b.addMouseListener(new MouseAdapter(){
    public void mousePressed(MouseEvent e){
      System.out.println("Boton pulsado");
    }
}
```

Debido a la duración del curso se han quedado algunas sin ver como por ejemplo:

- Pluggable Look and Feel.
- El patrón Model-View-Controller que utiliza Swing.
- Unos cuantos componentes Swing.
- Algunos gestores de organización.