9. TEMA 9: Desarrollo de interfaces gráficas de usuario.

Uno de los aspectos más importantes en la programación, a parte del buen desarrollo de código y la buena funcionalidad de la aplicación, es desarrollar una interfaz de usuario que muestre todas las capacidades y funcionalidades de la aplicación.

En este tema se va a conocer los diferentes componentes básicos que componen una interfaz gráfica y aprendemos como crear interfaces sencillas mediante una herramienta de eclipse llamada “Swing Designer”.

Se conocerá el modelo vista/controlador, en que se basa y como diferenciar estas partes.

Además se aprenderá a interactuar con esta interfaz mediante el modelo de eventos, el cual nos va a permitir comunicar la interfaz con el control de la aplicación.

9.1 Introducción.

Para poder seguir este tema debemos instalar en eclipse el complemento Swing Designer. En el momento de elaborar este libro este plug-in se encuentra en la siguiente ubicación:

http://dl.google.com/eclipse/inst/d2wbpro/latest/3.7

Los pasos a seguir para instalarlo son:

- Pinchar en el menu superior en Help > Install New Software.
- En la ventana que aparece pegamos la url puesta arriba en el campo “Work with” y pulsamos intro.
- En el recuadro blanco que tenemos abajo aparecerán varias opciones, entre ellas una que se llama “Swing Designer”, la marcamos y pulsamos en el botón “Siguiente” o “Next”.
- Después de unos segundos en la ventana que aparece pulsamos de nuevo en “Siguiente”
- En la ventana que aparece a continuación marcamos la opción “I accept terms of the license agreement” y para terminar pulsamos “Finish”.
- Puede que durante la instalación nos pida confirmar de nuevo la instalación, en todo caso pulsaremos “Ok”.
- Una vez realizada la instalación se reiniciará eclipse y podremos empezar a usar el nuevo complemento.

9.2 Tipos de interfaces. AWT y Swing.

Tanto en Java como en otros entornos disponemos de una serie de API's, componentes o herramientas que nos van a permitir dar un aspecto visual mas amigable a nuestra aplicación. En Java tenemos una jerarquía muy amplia de estos componentes, se expone aquí un ejemplo básico de la misma:

Las partes en las que podemos dividir una interfaz son las siguientes:

- Contenedores: Es por decirlo de alguna manera la ventana o partes de una ventana en la cual se agrupan los componentes. Pueden ser de diferentes tipos, alto nivel o nivel intermedio.
- Componentes: menús, botones, areas de texto, …
- El modelo de eventos: Aquellas acciones que pueden ser registradas por un componente.

En Java disponemos de 2 kits de interfaces: AWT y Swing.

- AWT (Abstract Window Toolkit): El desarrollar la aplicación con este kit nos aporta la ventaja de que la aplicación va a tener un aspecto visual muy parecido al subyacente nativo del sistema operativo sobre el cual se ejecuta. De esta manera obtenemos aplicaciones en las cuales el aspecto visual puede variar adaptándose al aspecto de ventanas y componentes del sistema operativo sobre el que se ejecuta.
- Swing: Si desarrollamos la aplicación bajo este kit el resultado que vamos a obtener es el mismo en cualquier sistema operativo, por tanto el aspecto visual siempre será el mismo, lo cual nos aporta la ventaja de tener un mayor control sobre el aspecto visual final de la aplicación. Generalmente los componentes en común con AWT suelen llamarse igual con una “J” mayúscula al principio, por ejemplo en AWT es Frame → en Swing es JFrame.

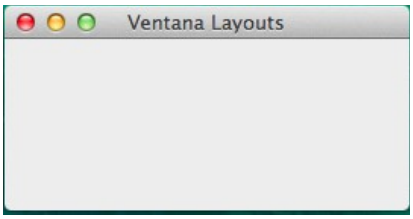
Principalmente en este tema nosotros vamos a trabajar con Swing.

9.3 Contenedores

Los contenedores son aquellos componentes que como su propio nombre indica contienen otros componentes o incluso otros contenedores dentro de ellos mismos.

9.3.1 JFrame

Implementa una ventana. Las ventanas principales de una aplicación deberían ser de este tipo. Un JFrame nos proporciona métodos para agregar componentes dentro de él y también para recoger el contenedor sobre el que agregarlos. Así mismo también nos proporciona múltiples métodos para manejar y personalizar la ventana, tanto sus dimensiones como su situación en la pantalla y las acciones que realizamos sobre ella o los botones de “cerrar”, “minimizar” o “maximizar”.



Veamos como se implementa una sencilla ventana en la cual situamos un elemento etiqueta con un texto “Hola, mundo!!” y le damos unas dimensiones y situación, además también vamos a concretar la acción del botón “cerrar” de la ventana:

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.WindowConstants;
```

```
public class MiFrame {
	public static void main(String[] args) {
		JFrame frame=new JFrame("Mi primera ventana");
		JPanel panel=new JPanel();
		frame.setContentPane(panel);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JLabel lbHola=new JLabel("Hola Mundo!!");
		panel.add(lbHola);
		frame.pack();   // Esto empaqueta la ventana,la ajusta al contenido
		frame.setSize(400, 400);   // Esto le da unas dimensiones 400x400
		frame.setBounds(100, 100, 400, 600); // Esto la pone en las
                                             // coordenadas x:100 y:100 y
                                             // dimensiones 400x600
		frame.setVisible(true);
	}
}
```

Como se puede observar para agregar componentes a la ventana usamos el método "add" como se muestra en el código anterior. Generalmente no los agregaremos directamente sobre el "frame", sino que al "frame" le vamos a asignar un objeto de tipo "JPanel" que esta definido por un "layout" que nos dice como se van a distribuit los componentes asignados a ese panel como veremos más adelante. Por tanto los componentes los agregaremos sobre el "Jpanel".

El ejemplo expuesto arriba puede ser más o menos representativo ver de una manera sencilla como crear ventanas, pero generalmente las lo que se suele hacer es personalizar un tipo de ventana adecuada a nuestras necesidades heredando de "JFrame", veamos el mismo ejemplo expuesto arriba pero heredando de "JFrame":

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class MiFrame extends JFrame{
	public MiFrame() {
		super("Mi primera ventana");
		JPanel panel=new JPanel();
		this.setContentPane(panel);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JLabel lbHola=new JLabel("Hola Mundo!!");
		panel.add(lbHola);
		this.setBounds(100, 100, 400, 600); // Esto la pone en las
                                            // coordenadas x:100 y:100 y
                                            // dimensiones 400x600
		this.setVisible(true);
	}
	public static void main(String[] args) {
		MiFrame miVentana=new MiFrame();
	}
}
```

9.3.2 JDialog

Los “JDialog” son ventanas secundarías que emergen a partir de una ventana principal, por tanto si la ventana principal se cierra las secundarias también.

Existen varios tipos diferentes de “JDialog”, de los cuales veremos 2:

- ***JOptionPane***: Los elementos “JOptionPane” generalmente están relacionados con aquellas ventanas emergentes o auxiliares que usamos en las aplicaciones de entorno ventana, por ejemplo una ventana informativa con un botón “Aceptar”, o una ventana en que nos pregunta una acción a realizar y tenemos los botones “Aceptar” o “Cancelar”, o incluso una ventana emergente para seleccionar un archivo. Dentro de esta encontramos varios tipos diferentes de los cuales nosotros veremos estos 3:
  - *JOptionPane.ShowMessageDialog*: Es una ventana a nivel informativo.


```
JOptionPane.showMessageDialog(getParent(), "Mensaje de aviso",
            "Mi Mensaje", JoptionPane.INFORMATION_MESSAGE);
```

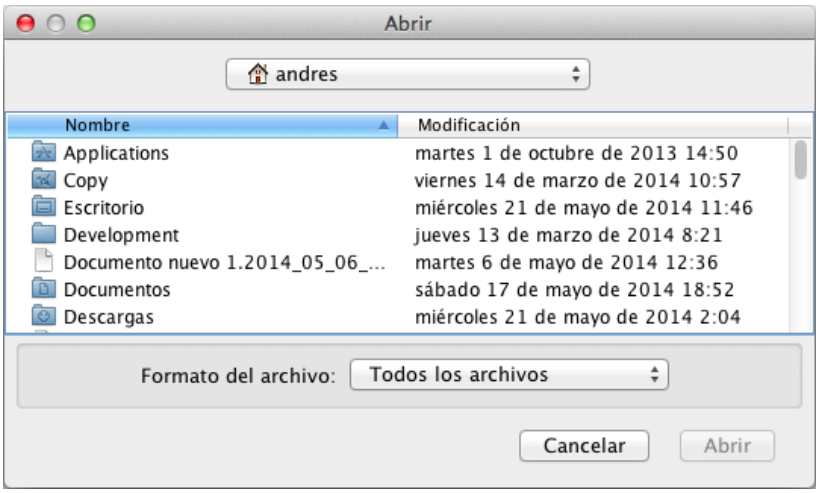
  - *JOptionPane.ShowInputDialog*: Es una ventana que me permite que el usuario introduzca un valor y recogerlo posteriormente.


```
String num = JOptionPane.showInputDialog("Dame un número: ");
if (num!=null){
      JOptionPane.showMessageDialog(null, "El número es: "+num);
}
```

  - *JOptionPane.ShowConfirmDialog*: Es una ventana que nos permite confirmar con botones como “Cancelar”, “Si”, “No”, “Ok”,...

```
int op = JoptionPane.showConfirmDialog(getParent(),
                                "Eres programador de Java?");
if (op == JOptionPane.YES_OPTION){
      JOptionPane.showMessageDialog(null, "Has seleccionado SI.");
}else{
      if (op == JOptionPane.NO_OPTION){
            JOptionPane.showMessageDialog(null, "Has seleccionado NO.");
      }
}
```

- ***JFileChooser***: Se trata de un selector de archivos, el cual nos permite interactivamente seleccionar un fichero o directorio.



Un ejemplo de su uso sería el siguiente:

```
JFileChooser file = new JFileChooser();
file.showOpenDialog(getParent());
File fichero = file.getSelectedFile();
System.out.println("\nEl fichero abierto es:"+fichero.getName());
```

9.3.3 Layouts

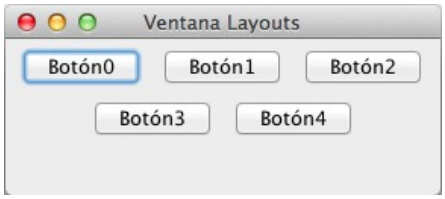
Los Layouts son elementos que nos van a ayudar a definir como se distribuyen los componentes dentro de un contenedor. Por defecto un contenedor sino tiene asignado un layout los componentes que situemos se van a colocar uno por encima del otro, así que es conveniente cada vez que definimos un contenedor al mismo tiempo definir como se van a distribuir los componentes en el mismo. Existen diferentes tipos de layouts, aquí veremos los más básicos y habituales:

- ***FlowLayout***: Es el mas sencillo de todos, sitúa los componentes uno al lado del otro y pasa a la línea siguiente cuando sea necesario, según el tamaño del panel. Un ejemplo de uso sería el siguiente:

```
public static void main(String[] args) {
      JFrame ventana = new JFrame("Ventana Layouts");
      JPanel panel = new JPanel();
      ventana.setContentPane(panel);
      ventana.setBounds(50, 50, 300, 300);
      JButton botones[] = new JButton[5];
      panel.setLayout(new FlowLayout());
      for (int i = 0; i < botones.length; i++) {
```

```
            botones[i] = new JButton("Botón" + i);
            panel.add(botones[i]);
        }
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
}
```

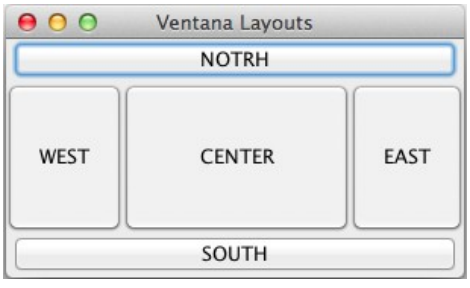
Y el resultado es:



- ***BorderLayout***: Es el layout por defecto para las ventanas, tiene 5 zonas divididas en las cuales podemos situar un solo componente, estas zonas se identifican con: NORTH (Norte), EAST (Este), WEST (Oeste), SOUTH (Sur) y CENTER (Centro). Un ejemplo de uso sería el siguiente:

```
public static void main(String[] args) {
        JFrame ventana = new JFrame("Ventana Layouts");
        JPanel panel = new JPanel();
        ventana.setContentPane(panel);
        ventana.setBounds(50, 50, 300, 300);
        JButton botones[] = new JButton[5];
        panel.setLayout(new BorderLayout());
        botones[0]=new JButton("NOTRH");
        panel.add(botones[0],BorderLayout.NORTH);
        botones[1]=new JButton("EAST");
        panel.add(botones[1],BorderLayout.EAST);
        botones[2]=new JButton("CENTER");
        panel.add(botones[2],BorderLayout.CENTER);
        botones[3]=new JButton("WEST");
        panel.add(botones[3],BorderLayout.WEST);
        botones[4]=new JButton("SOUTH");
        panel.add(botones[4],BorderLayout.SOUTH);
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
}
```

Y el resultado es:

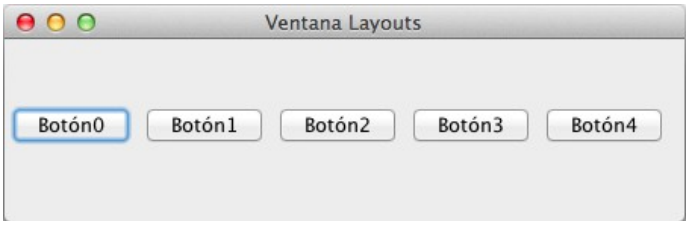


- ***BoxLayout***: Es un Layout parecido al FlowLayout, la diferencia es simple, FlowLayout ubica todos los componentes solo de forma horizontal, mientras que BoxLayout los ubica, tanto horizontal como vertical, por tanto podemos especificar

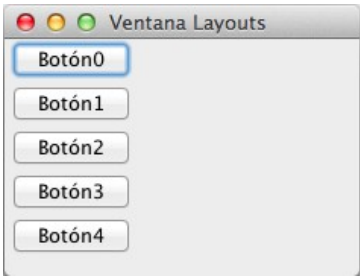
esto con un parámetro que puede ser "*BoxLayout.X_AXIS*" para el eje X, o "*BoxLayout.Y_AXIS*" para el eje Y. Un ejemplo de uso sería el siguiente:

```
public static void main(String[] args) {
	JFrame ventana = new JFrame("Ventana Layouts");
	JPanel panel = new JPanel();
	ventana.setContentPane(panel);
	ventana.setBounds(50, 50, 300, 300);
	JButton botones[] = new JButton[5];
	panel.setLayout(new BoxLayout(panel,BoxLayout.X_AXIS));
	for (int i = 0; i < botones.length; i++) {
		botones[i] = new JButton("Botón" + i);
		panel.add(botones[i]);
	}
	ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	ventana.setVisible(true);
}
```

Y el resultado es:



Si cambiamos la línea de color morado por `panel.setLayout(panel, BoxLayout.Y_AXIS));` el resultado sería:



- ***AbsoluteLayout***: Realmente no es un layout, podríamos decir que es la ausencia de layout, con lo cual al situar el layout del panel pondríamos "panel.setLayout(null);", por tanto si no hay un layout los componentes deben situarse en base a unas coordenadas "x" e "y". El no usar layout tiene el inconveniente de que los componentes no se ajustan ni redimensionan cuando redimensionamos la ventana, por tanto puede ser que se queden componentes fuera de la ventana y no sean visibles, esto no pasa con los anteriores layouts en los cuales podemos observar que si redimensionamos la ventana se van ajustando. Un ejemplo de uso en el que los botones se situan aleatoriamente en base al alto y ancho de la ventana, para que no se salgan, sería el siguiente:

```
public static void main(String[] args) {
	JFrame ventana = new JFrame("Ventana Layouts");
	JPanel panel = new JPanel();
	ventana.setContentPane(panel);
	ventana.setBounds(50, 50, 300, 300);
```

```
        JButton botones[] = new JButton[5];
        panel.setLayout(null);
        for(int i=0;i<botones.length;i++){
              botones[i]=new JButton("Botón"+i);
              botones[i].setBounds((int)(Math.random()*ventana.getWidth()),
                    (int)(Math.random()*ventana.getHeight()), 117, 29);
              panel.add(botones[i]);
        }
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
}
```

Y el resultado es:

- ***GridLayout***: Este layout nos permite situar los componentes como si se tratara de una cuadricula, de tal manera que podemos definir las filas (rows) y columnas (cols) que va a tener el layout. Si en las filas o columnas ponemos un cero, estamos indicando que queremos que los componentes ocupen todo el ancho o el alto respectivamente, por tanto si yo digo que las filas son 2 y las columnas 0 vamos a tener 2 filas de componentes distribuidos en múltiples columnas y se van a ir ajustando para ocupar entre todos los de la fila todo el ancho de la columna. Un ejemplo de uso con 2 filas y 0 columnas sería el siguiente:

```
public static void main(String[] args) {
        JFrame ventana = new JFrame("Ventana Layouts");
        JPanel panel = new JPanel();
        ventana.setContentPane(panel);
        ventana.setBounds(50, 50, 300, 300);
        JButton botones[] = new JButton[5];
        panel.setLayout(new GridLayout(2, 0));
        for (int i = 0; i < botones.length; i++) {
              botones[i] = new JButton("Botón" + i);
              panel.add(botones[i]);
        }
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
}
```

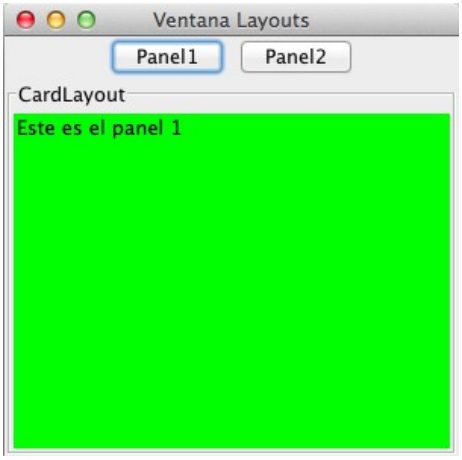
Y el resultado es:

- ***CardLayout***: Nos permite gestionar más de un layout al mismo tiempo. Sólo tenemos un layout visible en un momento determinado, pero nos permite gestionar varios y dejar visible uno de ellos, esto nos permite simular un efecto parecido al de las pestañas. Un ejemplo de uso sería el siguiente:

```
public static void main(String[] args) {
	JFrame ventana = new JFrame("Ventana Layouts");
	JPanel panel = new JPanel();
	ventana.setContentPane(panel);
	ventana.setBounds(50, 50, 300, 300);
	JButton botones[] = new JButton[2];
	panel.setLayout(new BoxLayout(ventana.getContentPane(),
			BoxLayout.Y_AXIS));
	JPanel pestanas = new JPanel();
	pestanas.setLayout(new BoxLayout(pestanas, BoxLayout.X_AXIS));
	panel.add(pestanas);
	final JPanel card = new JPanel();
	card.setBorder(BorderFactory.createTitledBorder("CardLayout"));
	card.setLayout(new CardLayout());
	panel.add(card);
	botones[0] = new JButton("Panel1");
	botones[0].addActionListener(new ActionListener() {
		@Override
		public void actionPerformed(ActionEvent arg0) {
			log.append("panel1");
			CardLayout c = (CardLayout) card.getLayout();
			c.show(card, "panel1");
		}
	});
	pestanas.add(botones[0]);
	botones[1] = new JButton("Panel2");
	botones[1].addActionListener(new ActionListener() {
		@Override
		public void actionPerformed(ActionEvent arg0) {
			log.append("panel2");
			CardLayout c = (CardLayout) card.getLayout();
			c.show(card, "panel2");
		}
	});
	pestanas.add(botones[1]);
	JPanel panel1 = new JPanel();
	panel1.setLayout(new BorderLayout());
	JLabel lb = new JLabel("Este es el panel 1");
```

```
        lb.setBackground(Color.GREEN);
        panel1.add(lb);
        card.add(panel1, "panel1");
        JPanel panel2 = new JPanel();
        panel2.setLayout(new BorderLayout());
        panel2.setBackground(Color.orange);
        lb = new JLabel("Este es el panel 2");
        lb.setBackground(Color.ORANGE);
        panel2.add(lb);
        card.add(panel2, "panel2");
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setVisible(true);
    }
```

Y el resultado es:



A todo lo visto con estos layouts cabe remarcar que dentro de un layout puedo agregar otros layouts, con lo cual las combinaciones para conseguir la interfaz deseada son inmensas. Por ejemplo puedo tener un BorderLayout y en la zona “WEST” situar un BoxLayout en el eje Y, de tal manera que podría tener en esa zona un conjunto de botones en vertical. De hecho en el ejemplo de “CardLayout” podemos observar este comportamiento.

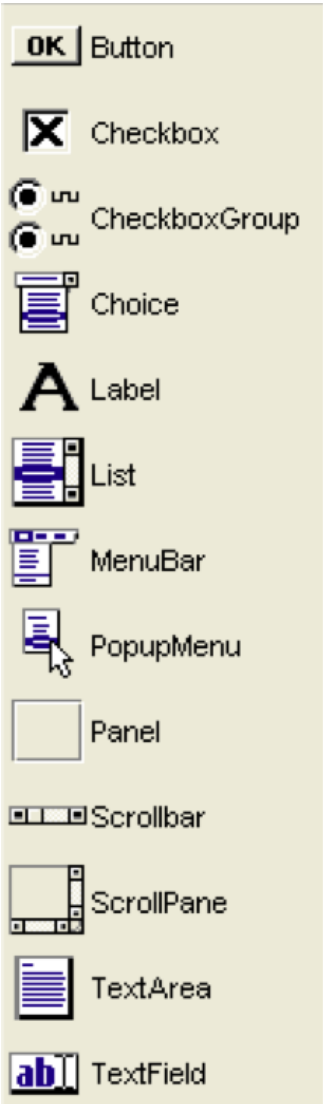
Otra cosa que nos va a ser de utilidad para decorar nuestras ventanas y componentes es la posibilidad de agregar un borde con un rótulo y colores, tal y como se ha hecho en el ejemplo del “CardLayout”. Para agregar un borde con rótulo usamos el método “setBorder” que corresponde a la línea:

```
card.setBorder(BorderFactory.createTitledBorder("CardLayout"));
```

Si lo que queremos es agregar color usaremos el método “setBackground” que en el caso del layout se lo hemos aplicado a la etiqueta con:

```
lb.setBackground(Color.GREEN);
```

9.4 Componentes



Los componentes son los botones, etiquetas, áreas de textos, etc... que van a llenar la ventana, y permitirnos de enseñar la información y interaccionar con el usuario. Todos los componentes tienen métodos comunes, heredan de la clase "Component", por tanto disponen de métodos para modificar el tamaño, el color, la fuente, etc... Pero cada componente tiene también sus propios métodos para sus acciones especificas. Aquí se van a exponer los métodos básicos con ejemplos sencillos, para más información sobre los métodos disponibles se debe recurrir a la documentación de Java. Los componentes se pueden incluir en la ventana principal, o en otro componente de tipo "Container", es decir un "JPanel" en el caso de los ejemplos que exponemos aquí. Hay que tener en cuenta que todos los elementos

9.4.1 Etiquetas

Las "etiquetas" o "Label" son aquellos elementos que nos sirven para situar texto en un determinado lugar del contenedor. La sintaxis para situar una etiqueta es la siguiente:

```
JLabel etiqueta = new JLabel("Esto es una etiqueta.");
```

9.4.2 Campos de texto

En este apartado vamos a distinguir tres componentes. El primero es un componente para recoger una cadena de valores, se trata del "TextField", el segundo es un "JPasswordField" que es muy parecido a "TextField" con la peculiaridad que los caracteres escritos no se muestran por pantalla, y el tercero es un componente para recoger una cadena de caracteres que admite múltiples líneas, se trata del "TextArea".

TextField

"TextField" dispone de un método "getText()" que nos devuelve el contenido del campo. Tiene diferentes constructores que nos permiten definir el texto que llevará por defecto o cuantos caracteres permite. Se va a ilustrar la creación de un "TextField":

```
JTextField tfNombre=new JTextField("Escribe aquí tu nombre");
```

Si queremos recoger su valor usaríamos el método "getText()":

```
System.out.println("Has escrito:"+tfNombre.getText());
```

JPasswordField

"JPasswordField" dispone de un método "getPassword()" que nos devuelve el contenido del campo en un array de char, por tanto hay que convertirlo a un String. Tiene diferentes constructores que nos permiten definir el texto que llevará por defecto o cuantos caracteres permite. Se va a ilustrar la creación de un "JPasswordField":

```
JPasswordField pfPass=new JPasswordField("Password");
```

Si queremos recoger su valor usaríamos el método "getText()":

```
System.out.println("Has escrito:"+new String(pfPass.getPassword()));
```

JTextArea

"TextArea" dispone de un método "getText()" que nos devuelve el contenido del campo. Tiene diferentes constructores que nos permiten definir el texto que llevará por defecto o cuantas filas y columnas de caracteres permite. Se va a ilustrar la creación de un "TextArea":

```
JTextArea taTexto=new JTextArea();
```

Si queremos recoger su valor usaríamos el método "getText()":

```
System.out.println("Has escrito:"+taTexto.getText());
```

Adicionalmente a este componente suele agregarse otro componente llamado "JScrollPane" que nos permite usar una barra de desplazamiento para poder visualizar todo el texto en el área de texto. Para decir a que componente afecta el "JscrollPane" indicaremos cuál es el componente pasando la instancia como parámetro del constructor, o bien podemos usar el método "setViewportView()" y pasándole igualmente la instancia del componente como parámetro. Una vez hecho esto, lo que agregaremos al contenedor no será el "TextArea" por ejemplo, sino el propio "JScrollPane". Su uso sería el siguiente:

```
JScrollPane sc=new JScrollPane(taTexto);
```

9.4.3 Selectores. Botones, ComboBox, CheckBox y RadioButton.

En este apartado veremos algunos de los componentes que nos sirven para realizar acciones del tipo selección.

Botones

Primero veremos como crear botones. El componente es "JButton", disponemos de varios constructores, pero generalmente usaremos el constructor al cual le pasamos como parámetro el nombre que va a tener el botón. Por tanto para crear un botón usaremos lo siguiente:

```
JButton ejemploBoton=new Jbutton("ejemplo");
```

ComboBox

Este componente consiste en una lista desplegable, cuando pinchamos en el se despliega una lista con diferentes opciones que son llamadas "Items". El componente es "JComboBox", disponemos de varios constructores, pero generalmente usaremos el constructor por defecto el cuál no recibe parámetros, y para agregar los "Items" usaremos el método "addItem(Object)" al cual le pasamos un objeto, que por ejemplo puede ser un String. Para recoger el valor seleccionado usaremos el método "getSelected()" que nos devuelve el objeto seleccionado. Por tanto para crear un botón usaremos lo siguiente:

```
JComboBox combo = new JComboBox();
combo.addItem("Opción 1");
combo.addItem("Opción 2");
combo.addItem("Opción 3");
```

CheckBox

Este componente se identifica con las casillas de marcado, por tanto tiene dos estados, marcada o no marcada, nos permiten seleccionar varias opciones de una vez. El componente es "JCheckBox", disponemos de varios constructores, pero generalmente usaremos el constructor al cual le pasamos como parámetro el nombre que va a tener el CheckBox y como segundo parámetro un valor boolean que indica si está marcado o

desmarcado. Cuando queramos recoger el valor para saber si está marcado o no usaremos el método “isSelected()” que devuelve true si está marcado o false si está desmarcado, y cuando queramos recoger el nombre del ChecBox usaremos el método “getName()”. Por tanto para crear un CheckBox usaremos lo siguiente:

```
JCheckBox cbCheck=new JCheckBox("marca 1",false);
```

RadioButton

Este componente es parecido a CheckBox con la peculiaridad que sólo podemos marcar una opción, de tal manera que todas las opciones son excluyentes, este componente se denomina “JRadioButton”. Entonces en este componente la cuestión radica en como controlamos que sólo se queda marcada una opción. Para conseguir el efecto deseado tenemos que usar un componente llamado “ButtonGroup” que nos permite agrupar botones, si agrupamos varios “RadioButton” con un “ButtonGroup” podemos conseguir que sólo quede marcado uno de manera automatica, debemos agregar los botones al “ButtonGroup” y después los mismos botones al contenedor. Los métodos para recoger el nombre y si está seleccionado son los mismos que para CheckBox. Veamos un ejemplo de uso:

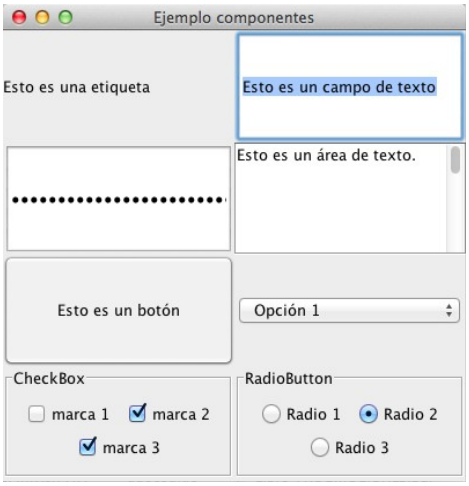
```
ButtonGroup bgGroup=new ButtonGroup();
JRadioButton rbRadio=new JRadioButton("Radio 1",false);
bgGroup.add(rbRadio);
panelRadio.add(rbRadio);
JRadioButton rbRadio2=new JRadioButton("Radio 2",false);
bgGroup.add(rbRadio2);
panelRadio.add(rbRadio2);
```

Veamos un ejemplo completo con todos estos componentes:

```
import java.awt.GridLayout;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class EjemploComponentes extends JFrame{
      public EjemploComponentes(){
            super("Ejemplo componentes");
            this.setBounds(50,50,400,400);
            JPanel panel=new JPanel();
            panel.setLayout(new GridLayout(0,2));
            JLabel etiq=new JLabel("Esto es una etiqueta");
            panel.add(etiq);
            JTextField campo=new JTextField("Esto es un campo de texto");
```

```
            panel.add(campo);
            JPasswordField pass=new JPasswordField("Esto es un campo de
password");
            panel.add(pass);
            JTextArea ta=new JTextArea();
            ta.setRows(20);
            ta.setText("Esto es un área de texto.");
            JScrollPane js=new JScrollPane(ta);
            panel.add(js);

            JButton boton=new JButton("Esto es un botón");
            panel.add(boton);
            JComboBox combo = new JComboBox();
            combo.addItem("Opción 1");
            combo.addItem("Opción 2");
            combo.addItem("Opción 3");
            panel.add(combo);
            JPanel panelCheck=new JPanel();
            panelCheck.setBorder(BorderFactory.createTitledBorder("CheckBox"));
            JCheckBox cbCheck=new JCheckBox("marca 1",false);
            panelCheck.add(cbCheck);
            cbCheck=new JCheckBox("marca 2",true);
            panelCheck.add(cbCheck);
            cbCheck=new JCheckBox("marca 3",true);
            panelCheck.add(cbCheck);
            panel.add(panelCheck);
            JPanel panelRadio=new JPanel();

      panelRadio.setBorder(BorderFactory.createTitledBorder("RadioButton"));
            ButtonGroup bgGroup=new ButtonGroup();
            JRadioButton rbRadio=new JRadioButton("Radio 1",false);
            bgGroup.add(rbRadio);
            panelRadio.add(rbRadio);
            rbRadio=new JRadioButton("Radio 2",true);
            bgGroup.add(rbRadio);
            panelRadio.add(rbRadio);
            rbRadio=new JRadioButton("Radio 3",false);
            bgGroup.add(rbRadio);
            panelRadio.add(rbRadio);
            panel.add(panelRadio);
            this.setContentPane(panel);
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.setVisible(true);
      }
      public static void main(String[] args) {
            EjemploComponentes miVentana=new EjemploComponentes();
      }
}
```

El resultado obtenido sería el siguiente:



9.4.4 Menús

Con los componentes que se van a exponer en este apartado seremos capaces de crear menús para nuestras aplicaciones. Disponemos de diferentes componentes que debemos saber utilizar y organizar para conseguir el efecto deseado.

- **JMenuBar** → Es la barra de menú principal. Una barra horizontal alargada en la que se colocarán las distintas opciones. Si miras en tu navegador, arriba, verás una barra de estas con opciones como "Archivo", "Editar", etc.
- **JMenu** → Es una de las cosas que se pueden añadir a un JMenuBar o a otro JMenu. Cuando añadimos uno de estos, tendremos un algo que al pinchar despliega un nuevo menú. Si en tu navegador, arriba donde pone "Archivo" pinchas con el ratón, verás que se despliega un menú con más opciones como "Abrir", "Guardar como", etc. Si añades un JMenu dentro de otro, tendrás un nuevo submenú que se despliega. Por ejemplo, si miras abajo en tu windows, en el botón de "Inicio" de abajo a la izquierda, lo pinchas y sale un menú. Ese "Inicio" sería un JMenu si estuviese hecho en java. En las opciones que despliega, ves que "Programas" tiene una flechita a la derecha. Poniendo ahí el ratón sale otro nuevo menú. Eso quiere decir que "Programas" también sería un JMenu que está dentro de JMenu "Inicio".
- **JMenuItem** → Es el elemento seleccionable del menú. Es el que cuando lo pinchas hace algo útil, como "guardar como", "abrir", etc.
- **JSeparator** → Este sólo sirve para poner una línea y separar varios JMenuItem. Por ejemplo, dentro de "Editar", las opciones "Copiar", "Cortar" y "Pegar" suelen estar separadas con líneas de otras opciones en el mismo menú.

Veamos un ejemplo de creación de un menú en una ventana:

```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JSeparator;
```

```
public class EjemploMenu {
	public static void main(String[] args) {
		JFrame ventana = new JFrame("Ventana Menu");
		JPanel panel = new JPanel();
		ventana.setContentPane(panel);
		ventana.setBounds(50, 50, 300, 300);
		JMenuBar menuBar = new JMenuBar();
		menuBar.setBounds(0, 0, panel.getWidth(), 22);
		panel.setLayout(new BorderLayout());
		panel.add(menuBar,BorderLayout.NORTH);

		JMenu mnuArchivo = new JMenu("Archivo");
		menuBar.add(mnuArchivo);

		JMenuItem miAbrir = new JMenuItem("Abrir");
		mnuArchivo.add(miAbrir);

		JMenuItem miImprimir = new JMenuItem("Imprimir");
		mnuArchivo.add(miImprimir);

		JMenu mnuGuardar = new JMenu("Guardar");
		mnuArchivo.add(mnuGuardar);

		JMenuItem miGuardar = new JMenuItem("Guardar");
		mnuGuardar.add(miGuardar);

		JMenuItem miGuardarComo = new JMenuItem("Guardar como");
		mnuGuardar.add(miGuardarComo);

		JSeparator separator = new JSeparator();
		mnuArchivo.add(separator);

		JMenuItem miSalir = new JMenuItem("Salir");
		mnuArchivo.add(miSalir);

		JMenu mnuAcercaDe = new JMenu("Acerca de");
		menuBar.add(mnuAcercaDe);
		ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		ventana.setVisible(true);
	}
}
```

9.5 Modelo/Vista/Controlador

Antes de explicar como funcionan los eventos debe entenderse también como funciona el modelo/vista/controlador (MVC) ya que es la base para entender las interacciones del usuario con estos componentes.

El patrón MVC fue una de las primeras ideas en el campo de las interfaces gráficas de usuario y uno de los primeros trabajos en describir e implementar aplicaciones software en términos de sus diferentes funciones.

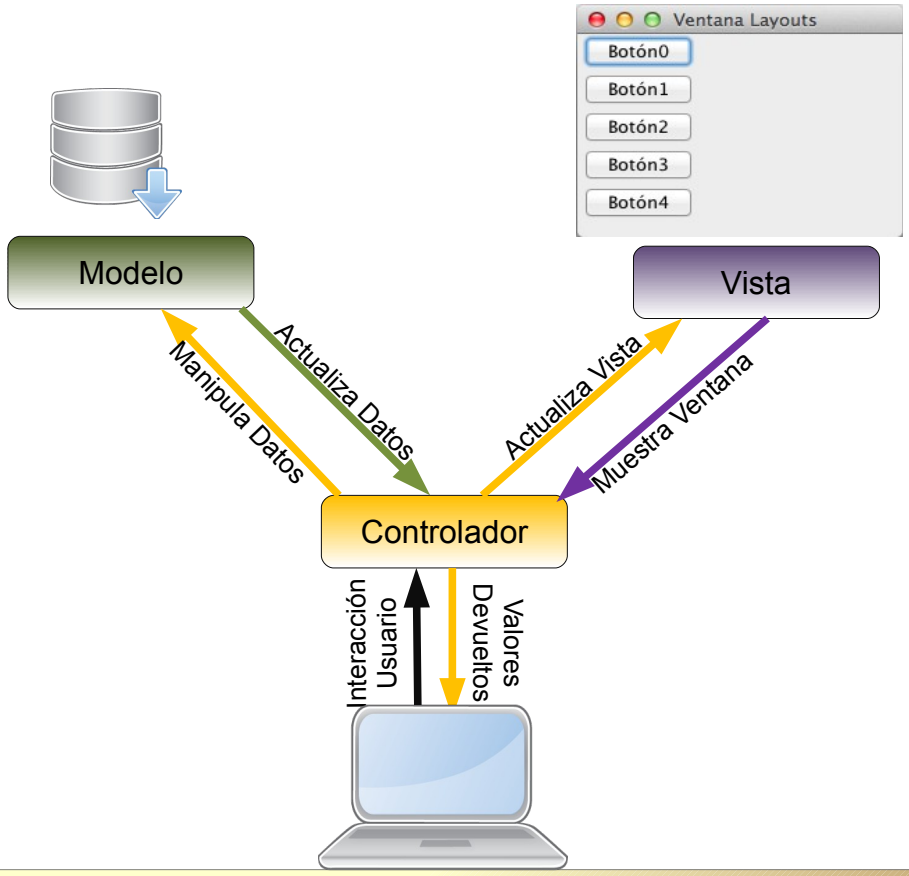
El modelo/vista/controlador (MVC) es un patrón de arquitectura de software que separa

los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

De manera genérica, los componentes de MVC se podrían definir como sigue:

- **El Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la "vista" aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al "modelo" a través del "controlador" .
- **El Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al "modelo" cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su "vista" asociada si se solicita un cambio en la forma en que se presenta de "modelo" (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el "controlador" hace de intermediario entre la "vista" y el "modelo".
- **La Vista**: Presenta el "modelo" (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho "modelo" la información que debe representar como salida.

Veamos de una manera más gráfica como interactúan estos 3 elementos:



9.6 Eventos

El modelo de eventos tiene una clara diferencia respecto a la manera que hemos visto de programar hasta ahora. Hasta este momento cuando programamos el flujo del programa lo vamos dirigiendo nosotros como programadores, aunque en ciertos momentos necesitemos la interacción del usuario para introducir algún dato, pero somos nosotros los que dictamos cuando entra en juego el usuario. Con el modelo de eventos esto no pasa, aunque como programadores dirigimos el flujo principal del programa, este puede variar dependiendo de las interacciones que haga el usuario en cualquier momento.

Por tanto esto hace que tengamos que cambiar un poco la forma de ver nuestra aplicación. Se podría decir que antes era el usuario el que debe estar pendiente cuando se necesita una acción por su parte, ahora es nuestro programa, o más concretamente nuestros componentes quienes deben estar pendientes de las acciones del usuario, ya que el usuario puede realizar una acción sobre cualquiera de ellos, presionar un botón, escribir en un campo de texto, pasar el ratón por encima de algún componente, pulsar una tecla, etc..., ¿y como hacemos que cada uno de nuestros componentes esté pendiente de las acciones del usuario?, pues para ello debemos hacer uso de lo que se llaman “Listeners” o “escuchadores”, que por decirlo de alguna manera es como si el componente estuviese a la escucha de las acciones del usuario. Existen multitud de “Listeners”, para detectar pulsaciones de teclas, pulsaciones y movimientos del ratón, pulsaciones o selecciones sobre un elemento concreto, etc... Cuando yo suscribo un componente debo hacerlo a aquel “Listener” que quiero que escuche. Puedo suscribir un mismo componente a multitud de “Listeners”.

En nuestro caso vamos a hacer uso del “ActionListener”, que es aquel “escuchador” que se ejecuta cuando ejecuto una acción de confirmación sobre un elemento, como por ejemplo pulsar un botón, seleccionar un elemento de una lista, escribir en un campo de texto y pulsar intro, etc... El “ActionListener” cuando se ejecuta lanza el método “actionPerformed” el cuál ejecutará aquello que contenga, aquí es donde entramos como programadores, cuando yo quiero que se ejecute una acción cuando pinche un botón debo implementar el método “actionPerformed”. El método “actionPerformed” recibe un objeto del tipo “ActionEvent” el cuál nos permite identificar el componente que lo ha llamado, este objeto podemos usarlo o no, dependiendo si varios componentes están llamando a un mismo “ActionListener”, en caso de no ser así no sería necesario identificar el componente, ya que sólo existe uno que lo ha llamado.

Veamos un ejemplo para capturar el evento de pulsación de un botón:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class Eventos1 extends JFrame{
	public Eventos1() {
		super("Eventos1");
		JPanel panel=new JPanel();
```

```
		this.setContentPane(panel);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JButton btPulsa=new JButton("Pulsame!!");
		btPulsa.addActionListener(new ActionListener(){
			@Override
			public void actionPerformed(ActionEvent arg0) {
				JOptionPane.showMessageDialog(getParent(),
						"Hola mundo!!", "Saludo",
						JOptionPane.INFORMATION_MESSAGE);
			}
		});
		panel.add(btPulsa);
		this.setBounds(100, 100, 300, 200);
		this.setVisible(true);
	}
	public static void main(String[] args) {
		Eventos1 ev1=new Eventos1();
	}
}
```

La parte que está marcada de color naranja es en la que podemos observar como agregamos un “ActionListener” a “btPulsa”. Y en el método “actionPerformed” realizamos la acción cuando pinchamos en el botón, que en este caso mostramos un mensaje informativo.

Se va a mostrar otra forma, más elegante con la que podemos también controlar los eventos del mismo botón, está forma consiste en poner a parte el evento para que no esté mezclado en el código, van a marcar en naranja las zonas relevantes:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class Eventos2 extends JFrame implements ActionListener{
	public Eventos2() {
		super("Eventos2");
		JPanel panel=new JPanel();
		this.setContentPane(panel);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JButton btPulsa=new JButton("Pulsame!!");
		btPulsa.addActionListener(this);
		panel.add(btPulsa);
		this.setBounds(100, 100, 300, 200);
		this.setVisible(true);
	}
	@Override
	public void actionPerformed(ActionEvent e) {
		JOptionPane.showMessageDialog(getParent(), "Hola mundo!!",
				"Saludo", JOptionPane.INFORMATION_MESSAGE);
	}
	public static void main(String[] args) {
```

```
            Eventos2 ev1=new Eventos2();
      }
}
```

El inconveniente de la forma expuesta arriba es que todas las acciones van al mismo "actionPerformed" por tanto si tenemos más de dos componentes, en nuestro caso dos botones, tendríamos que diferenciarlos, para diferenciarlos asociamos una cadena a cada botón con el método "setActionCommand(String)", así después en el método "actionPerformed" podemos identificarlos. Para hacerlo con el ejemplo anterior tendría que ser de la siguiente manera:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class Eventos2 extends JFrame implements ActionListener{
      public Eventos2() {
            super("Eventos2");
            JPanel panel=new JPanel();
            this.setContentPane(panel);
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            JButton btPulsa=new JButton("Primer botón");
            btPulsa.setActionCommand("boton1");
            btPulsa.addActionListener(this);
            panel.add(btPulsa);
            JButton btPulsa2=new JButton("Segundo botón");
            btPulsa2.setActionCommand("boton2");
            btPulsa2.addActionListener(this);
            panel.add(btPulsa2);
            this.setBounds(100, 100, 300, 200);
            this.setVisible(true);
      }
      @Override
      public void actionPerformed(ActionEvent e) {
            if (e.getActionCommand().equals("boton1")){
                  JOptionPane.showMessageDialog(getParent(),
                              "Primer botón","boton 1",
                              JOptionPane.INFORMATION_MESSAGE);
            }
            if (e.getActionCommand().equals("boton2")){
                  JOptionPane.showMessageDialog(getParent(),
                              "Segundo botón","boton 2",
                              JOptionPane.INFORMATION_MESSAGE);
            }
      }
      public static void main(String[] args) {
            Eventos2 ev1=new Eventos2();
      }
}
```

Y ahora por último se va a explicar otra manera de realizar acciones asociadas a un botón que también podría considerarse una manera organizada y limpia. Consiste en crear el "ActionListener" en una zona aparte, veamos un ejemplo:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class Eventos3 extends JFrame{
	public Eventos3() {
		super("Eventos3");
		JPanel panel=new JPanel();
		this.setContentPane(panel);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JButton btPulsa=new JButton("Pulsame!!");
		btPulsa.addActionListener(accionPulsa);
		panel.add(btPulsa);
		this.setBounds(100, 100, 300, 200);
		this.setVisible(true);
	}
	ActionListener accionPulsa = new ActionListener(){
		@Override
		public void actionPerformed(ActionEvent arg0) {
			JOptionPane.showMessageDialog(getParent(), "Hola mundo!!",
					"Saludo", JOptionPane.INFORMATION_MESSAGE);
		}
	};
	public static void main(String[] args) {
		Eventos3 ev1=new Eventos3();
	}
}
```

9.7 Ejemplo completo

En este apartado se va a dejar el código para que se analice un ejemplo completo con todos los componentes y layouts que se han visto en este tema.

```
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
```

```
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JTextArea;

public class EjemploCompleto extends JFrame{
	private JPanel panelPrincipal,panelComponentes,card,panelRadio,panelCheck;
	JTextArea log;
	JComboBox combo;
	public EjemploCompleto(){
		super("EjemploCompleto");
		setBounds(0,0,1000,800);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.getContentPane().setLayout(new BorderLayout());
		panelPrincipal = new JPanel();
		this.getContentPane().add(panelPrincipal,BorderLayout.CENTER);
		panelPrincipal.setLayout(new BoxLayout(panelPrincipal,
BoxLayout.Y_AXIS));
		crearMenu();
		panelComponentes = new JPanel();


panelComponentes.setBorder(BorderFactory.createTitledBorder("Componentes"));
		panelPrincipal.add(panelComponentes);
		panelComponentes.setLayout(new GridLayout(0, 2, 0, 0));

		log = new JTextArea();
		log.setBorder(BorderFactory.createTitledBorder("Ventana de Log"));
		JScrollPane sc=new JScrollPane(log);
		panelPrincipal.add(sc);
		JButton btnMessageDialog=new JButton("MessageDialog");
		btnMessageDialog.addActionListener(new ActionListener(){
			@Override
			public void actionPerformed(ActionEvent arg0) {
				JOptionPane.showMessageDialog(getParent(), "Mensaje de
aviso",
					"Mi Mensaje",
JOptionPane.INFORMATION_MESSAGE);
			}
```

```
		});
		panelComponentes.add(btnMessageDialog);

		JButton btnInputDialog=new JButton("InputDialog");
		btnInputDialog.addActionListener(new ActionListener(){
			@Override
			public void actionPerformed(ActionEvent arg0) {
				String num = JOptionPane.showInputDialog("Dame un
número: ");
				if (num!=null){
					JOptionPane.showMessageDialog(null, "El número es:
"+num);
				}
			}
		});
		panelComponentes.add(btnInputDialog);

		JButton btnInputDialog2=new JButton("InputDialog2");
		btnInputDialog2.addActionListener(new ActionListener(){
			@Override
			public void actionPerformed(ActionEvent arg0) {
				Object seleccion = JOptionPane.showInputDialog(
							getParent(),
							"Seleccione opcion",
							"Selector de opciones",
							JOptionPane.QUESTION_MESSAGE,
							null,  // null para icono defecto
							new Object[] { "opcion 1", "opcion 2",
"opcion 3" },
							"opcion 1");
				if (seleccion!=null){
					JOptionPane.showMessageDialog(null, "Has
seleccionado: "+seleccion);
				}
			}
		});
		panelComponentes.add(btnInputDialog2);

		JButton btnConfirmDialog=new JButton("ConfirmDialog");
		btnConfirmDialog.addActionListener(new ActionListener(){
			@Override
			public void actionPerformed(ActionEvent arg0) {
				int op = JOptionPane.showConfirmDialog(getParent(),"Eres
programador de Java?");
				if (op == JOptionPane.YES_OPTION){
					JOptionPane.showMessageDialog(null, "Has
seleccionado SI.");
				}else{
					if (op == JOptionPane.NO_OPTION){
						JOptionPane.showMessageDialog(null, "Has
seleccionado NO.");
					}
```

```
                        }
                  }
            });
            panelComponentes.add(btnConfirmDialog);

            JButton btnFileChooser=new JButton("JFileChooser");
            btnFileChooser.addActionListener(new ActionListener() {
                  @Override
                  public void actionPerformed(ActionEvent arg0) {
                        JFileChooser file = new JFileChooser();
                        file.showOpenDialog(getParent());
                        File fichero = file.getSelectedFile();
                        log.append("\nEl fichero abierto
es:"+fichero.getName());
                  }
            });
            panelComponentes.add(btnFileChooser);

            // Creacion del JComboBox y añadir los items.
            combo = new JComboBox();
            combo.addItem("Layouts");
            combo.addItem("FlowLayout");
            combo.addItem("BorderLayout");
            combo.addItem("BoxLayout.X_AXIS");
            combo.addItem("BoxLayout.Y_AXIS");
            combo.addItem("AbsoluteLayout");
            combo.addItem("GridLayout");
            combo.addItem("CardLayout");

            // Accion a realizar cuando el JComboBox cambia de item
seleccionado.
            combo.addActionListener(new ActionListener() {
                  @Override
                  public void actionPerformed(ActionEvent e) {
                        layouts(combo.getSelectedItem().toString());
                        log.append("\n"+combo.getSelectedItem().toString());
                  }
            });
            panelComponentes.add(combo);

            final TextField tfNombre=new TextField("Escribe aquí tu nombre");
            tfNombre.addActionListener(new ActionListener() {
                  @Override
                  public void actionPerformed(ActionEvent arg0) {
                        log.append("\nHas escrito:"+tfNombre.getText());
                  }
            });
            panelComponentes.add(tfNombre);

            final JPasswordField pfPass=new JPasswordField("Password");
            pfPass.addActionListener(new ActionListener() {
                  @Override
```

```
                public void actionPerformed(ActionEvent arg0) {
                    log.append("\nHas escrito:"+new
String(pfPass.getPassword()));
                }
            });
            panelComponentes.add(pfPass);

            panelCheck=new JPanel();
            panelCheck.setBorder(BorderFactory.createTitledBorder("CheckBox"));
            JCheckBox cbCheck=new JCheckBox("marca 1",false);
            cbCheck.addActionListener(accionCheck);
            panelCheck.add(cbCheck);
            cbCheck=new JCheckBox("marca 2",true);
            cbCheck.addActionListener(accionCheck);
            panelCheck.add(cbCheck);
            cbCheck=new JCheckBox("marca 3",true);
            cbCheck.addActionListener(accionCheck);
            panelCheck.add(cbCheck);
            panelComponentes.add(panelCheck);

            panelRadio=new JPanel();

        panelRadio.setBorder(BorderFactory.createTitledBorder("RadioButton"));
            ButtonGroup bgGroup=new ButtonGroup();
            JRadioButton rbRadio=new JRadioButton("Radio 1",false);
            rbRadio.addActionListener(accionRadio);
            bgGroup.add(rbRadio);
            panelRadio.add(rbRadio);
            rbRadio=new JRadioButton("Radio 2",true);
            rbRadio.addActionListener(accionRadio);
            bgGroup.add(rbRadio);
            panelRadio.add(rbRadio);
            rbRadio=new JRadioButton("Radio 3",false);
            rbRadio.addActionListener(accionRadio);
            bgGroup.add(rbRadio);
            panelRadio.add(rbRadio);
            panelComponentes.add(panelRadio);

            this.setVisible(true);
        }
        public void crearMenu(){
            JMenuBar menuBar = new JMenuBar();
            menuBar.setBounds(0, 0, this.getContentPane().getWidth(), 22);
            this.getContentPane().add(menuBar,BorderLayout.NORTH);

            JMenu mnuArchivo = new JMenu("Archivo");
            menuBar.add(mnuArchivo);

            JMenuItem miAbrir = new JMenuItem("Abrir");
            mnuArchivo.add(miAbrir);

            JMenuItem miImprimir = new JMenuItem("Imprimir");
```

```
        mnuArchivo.add(miImprimir);

        JMenu mnuGuardar = new JMenu("Guardar");
        mnuArchivo.add(mnuGuardar);

        JMenuItem miGuardar = new JMenuItem("Guardar");
        mnuGuardar.add(miGuardar);

        JMenuItem miGuardarComo = new JMenuItem("Guardar como");
        mnuGuardar.add(miGuardarComo);

        JSeparator separator = new JSeparator();
        mnuArchivo.add(separator);

        JMenuItem miSalir = new JMenuItem("Salir");
        mnuArchivo.add(miSalir);

        JMenu mnuAcercaDe = new JMenu("Acerca de");
        menuBar.add(mnuAcercaDe);
    }
    ActionListener accionCheck= new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent arg0) {
            String marcados="Has marcado los CheckBox:\n";
            for(Component c:panelCheck.getComponents()){
                JCheckBox cb=(JCheckBox)c;
                if (cb.isSelected()){
                    marcados=marcados+cb.getText()+"\n";
                }
            }
            log.append(marcados);
        }
    };
    ActionListener accionRadio= new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent arg0) {
            String marcados="Has marcado:\n";
            for(Component c:panelRadio.getComponents()){
                JRadioButton rb=(JRadioButton)c;
                if (rb.isSelected()){
                    marcados=marcados+rb.getText()+"\n";
                }
            }
            log.append(marcados);
        }
    };
    public void layouts(String tipo){
        JFrame ventana=new JFrame("Ventana Layouts");
        JPanel panel=new JPanel();
        ventana.setContentPane(panel);
        ventana.setBounds(50,50,300,300);
        JButton botones[]=new JButton[5];
```

```
		if (tipo.equals("FlowLayout")){
			panel.setLayout(new FlowLayout());
			for(int i=0;i<botones.length;i++){
				botones[i]=new JButton("Botón"+i);
				panel.add(botones[i]);
			}
		}
		if (tipo.equals("BorderLayout")){
			panel.setLayout(new BorderLayout());
			botones[0]=new JButton("NOTRH");
			panel.add(botones[0],BorderLayout.NORTH);
			botones[1]=new JButton("EAST");
			panel.add(botones[1],BorderLayout.EAST);
			botones[2]=new JButton("CENTER");
			panel.add(botones[2],BorderLayout.CENTER);
			botones[3]=new JButton("WEST");
			panel.add(botones[3],BorderLayout.WEST);
			botones[4]=new JButton("SOUTH");
			panel.add(botones[4],BorderLayout.SOUTH);
		}
		if (tipo.equals("BoxLayout.X_AXIS")){
			panel.setLayout(new
BoxLayout(ventana.getContentPane(),BoxLayout.X_AXIS));
			for(int i=0;i<botones.length;i++){
				botones[i]=new JButton("Botón"+i);
				panel.add(botones[i]);
			}
		}
		if (tipo.equals("BoxLayout.Y_AXIS")){
			panel.setLayout(new
BoxLayout(ventana.getContentPane(),BoxLayout.Y_AXIS));
			for(int i=0;i<botones.length;i++){
				botones[i]=new JButton("Botón"+i);
				panel.add(botones[i]);
			}
		}
		if (tipo.equals("AbsoluteLayout")){
			panel.setLayout(null);
			for(int i=0;i<botones.length;i++){
				botones[i]=new JButton("Botón"+i);
				botones[i].setBounds((int)
(Math.random()*ventana.getWidth()),
					(int)(Math.random()*ventana.getHeight()),
117, 29);
				panel.add(botones[i]);
			}
		}
		if (tipo.equals("GridLayout")){
			panel.setLayout(new GridLayout(2, 0));
			for(int i=0;i<botones.length;i++){
				botones[i]=new JButton("Botón"+i);
```

```
                        panel.add(botones[i]);
                  }
            }
            if (tipo.equals("CardLayout")){
                  panel.setLayout(new
BoxLayout(ventana.getContentPane(),BoxLayout.Y_AXIS));
                  JPanel pestanas=new JPanel();
                  pestanas.setLayout(new GridLayout(1,2));
                  panel.add(pestanas);
                  card=new JPanel();
                 card.setBorder(BorderFactory.createTitledBorder("CardLayout"));
                  card.setLayout(new CardLayout());
                  panel.add(card);
                  botones[0]=new JButton("Panel1");
                  botones[0].addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                              log.append("panel1");
                              CardLayout c=(CardLayout)card.getLayout();
                              c.show(card, "panel1");
                        }});
                  pestanas.add(botones[0]);
                  botones[1]=new JButton("Panel2");
                  botones[1].addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                              log.append("panel2");
                              CardLayout c=(CardLayout)card.getLayout();
                              c.show(card, "panel2");
                        }
                  });
                  pestanas.add(botones[1]);
                  JPanel panel1=new JPanel();
                  panel1.setLayout(new BorderLayout());
                  JLabel lb=new JLabel("Este es el panel 1");
                  lb.setBackground(Color.GREEN);
                  panel1.add(lb);
                  card.add(panel1,"panel1");
                  JPanel panel2=new JPanel();
                  panel2.setLayout(new BorderLayout());
                  panel2.setBackground(Color.orange);
                  lb=new JLabel("Este es el panel 2");
                  lb.setBackground(Color.ORANGE);
                  panel2.add(lb);
                  card.add(panel2,"panel2");
            }
            ventana.setVisible(true);
      }
      public static void main(String[] args) {
            EjemploCompleto ec=new EjemploCompleto();
      }
}
```

9.8 Ejercicios Propuestos

1. Realizar un Editor de textos que nos muestre la información de un archivo de texto y nos permita editarlo. El editor va a constar de un menú superior que consta del menú “Archivo” que tendrá la siguientes opciones “Nuevo”, “Abrir”, “Guardar”, “Guardar como” y “Salir”, y el menú “Acerca de” que nos mostrará una ventana informativa con la información acerca del autor del editor de texto. También tendremos en un panel inferior en el que mostraremos mediante una etiqueta la ruta y el archivo en edición. La opción “Nuevo” nos permitirá escribir un fichero nuevo que podrá ser guardado, “Abrir” debe mostrarnos una ventana para seleccionar el fichero de texto, y la opción “Guardar” actualizará la información del fichero en disco, además la opción “Guardar como” nos mostrara una ventana emergente en la cuál escribiremos el nombre del nuevo fichero.

2. Realizar un gestor de coches que me permite visualizar los campos de un coche en una ventana. Los Coches van a constar de una Marca, una Matricula, un Modelo, unos Caballos, una Posición y un estado Encendido. El gestor va a constar de un menú superior que consta del menú “Archivo” que tendrá la siguientes opciones “Nuevo”, “Abrir”, “Guardar”, “Guardar como” y “Salir”, y el menú “Acerca de” que nos mostrará una ventana informativa con la información acerca del autor del editor de texto. También tendremos en un panel inferior en el que mostraremos mediante una etiqueta la ruta y el archivo en edición. La opción “Nuevo” nos permitirá escribir un fichero nuevo que podrá ser guardado, “Abrir” debe mostrarnos una ventana para seleccionar el fichero de datos, y la opción “Guardar” actualizará la información del fichero en disco, además la opción “Guardar como” nos mostrara una ventana emergente en la cuál escribiremos el nombre del nuevo fichero. En la interfaz dispondremos de los botones “Siguiente” y “Anterior” para movernos por los diferentes registros, también se va a disponer de un botón para “Insertar” nuevos coches, y otro para “Editar” el actual, además de un botón para “Eliminar” el coche actual.