Programación II

Bloque temático 1. Lenguajes de programación

Bloque temático 2. Metodología de programación

Tema 2. Programación dirigida por eventos

Tema 3. Verificación de programas

Bloque temático 3. Esquemas algorítmicos

Tema 2. Programación dirigida por eventos

2.1. Introducción
2.2. Patrón Modelo-Vista-Controlador
2.3. Ventanas
2.4. Dibujar gráficos en un panel
2.5. Paneles y Gestores de Distribución
2.6. Componentes
2.7. Gestión de eventos
2.8. Ventanas y diálogos
2.9. Eventos de bajo nivel
2.10. Bibliografía

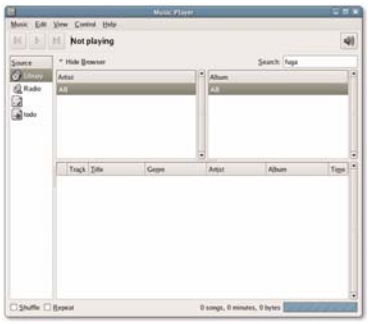
2.1 Introducción

En función del tipo de interacción con el usuario, los programas se clasifican en:

- ***Secuenciales*: reciben los datos de entrada, realizan un conjunto de operaciones o cálculos y muestran la salida**
  - **por ejemplo: comando "`ls`" de Linux**
- ***Interactivos*: exigen la intervención del usuario en tiempo de ejecución, pero el programa realiza acciones independientemente de las órdenes del usuario**
  - **por ejemplo: juego en tiempo real (deportes, estrategia, ...)**
- ***Dirigidos por eventos*: el programa "espera" la llegada de un evento (orden del usuario, ...), cuando el evento se produce realiza la acción correspondiente y vuelve a esperar**
  - **por ejemplo: procesador de textos**

Interfaces gráficas de usuario (GUIs)

Muchos de los programas basados en GUIs (*Graphical User Interface*) constituyen un ejemplo de programa dirigido por eventos

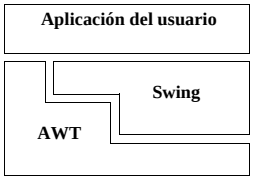


Las GUIs facilitan la interacción del usuario con el programa

Librerías Java para la creación de GUIs

En Java existen varias librerías que permiten crear interfaces gráficas de usuario:

- **AWT: Abstract Window Toolkit**
- ***Swing*: más moderna, basada en AWT**
- **SWT: Standard Widget Toolkit (Eclipse)**
- **Fundamentos**



Usaremos Swing+AWT:

- **Forman parte de las JFC (*Java Foundation Classes*)**
- **Permite cumplir el objetivo: practicar con la programación dirigida por eventos**
- **Paquetes `java.awt`, `javax.swing` (`http://java.sun.com/javase/6/docs/api/`)**

Elementos de una GUI

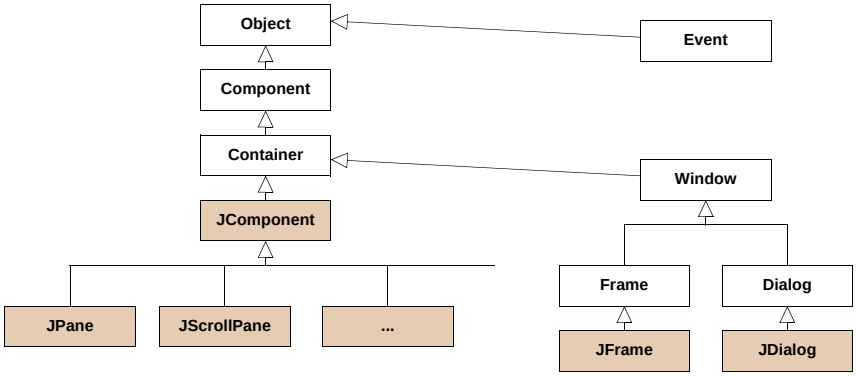
- **Componentes: un objeto que el usuario puede ver en la pantalla y con el que puede interactuar**
  - **botón, barra de desplazamiento, cuadro de texto, ...**
- **Contenedores: un componente que contiene otros componentes**
  - **ventana, panel, ...**
- **Eventos: una acción disparada por el usuario**
  - **pulsación de tecla, movimiento de ratón, pulsación del botón del ratón, ...**

El diseño de una GUI requiere:

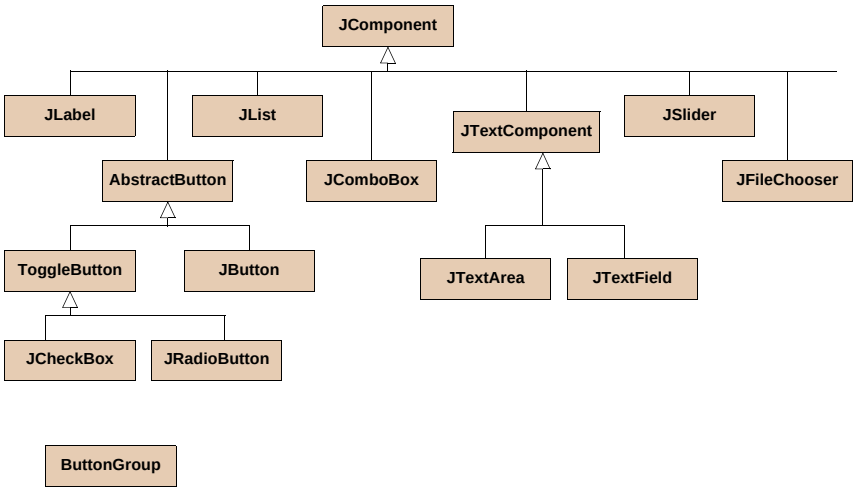
- **crear componentes,**
- **ponerles dentro de contenedores,**
- **y manejar los eventos generados por el usuario**

Jerarquía de clases Swing y AWT

Ventanas, contenedores y eventos:



Componentes Swing:



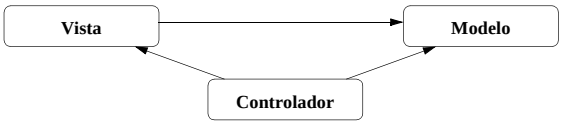
2.2 Patrón Modelo-Vista-Controlador

Modelo Vista Controlador **(MVC) es un patrón de arquitectura de software**

- **utilizado en aplicaciones con interfaz gráfica**
- **que divide la aplicación en tres partes: el Modelo, la Vista y el Controlador**

- ***Modelo*****: conjunto de clases que constituyen la solución del problema**
  - **puede existir independientemente de que exista la GUI**
- ***Vista*****: conjunto de elementos (ventanas) que constituyen la "vista" que el usuario tiene de la aplicación**
- ***Controlador*****: actúa cuando el usuario realiza una acción sobre un elemento activo de la vista (botón, cuadro de texto, ...)**

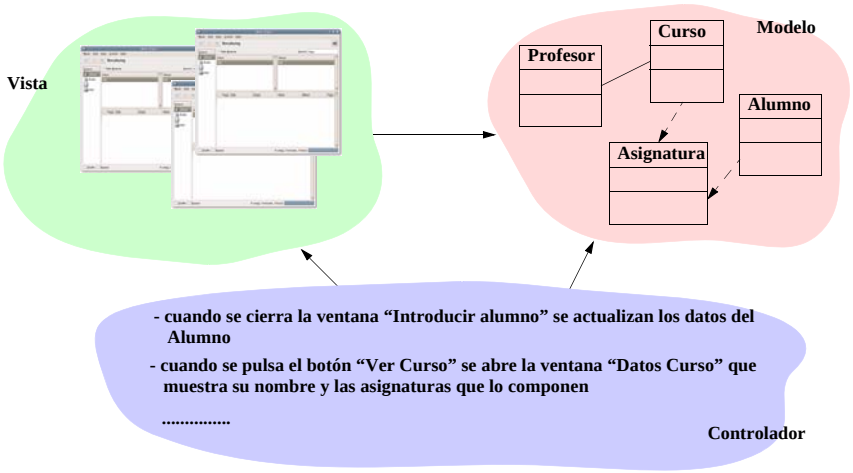
Relación entre Modelo, Vista y Controlador



- El Controlador accede a la Vista y al Modelo
  - necesita acceder a la Vista para conocer lo que el usuario ha introducido en los distintos elementos que la forman
  - con esa información accederá al Modelo para consultar o cambiar sus contenidos
- La Vista necesita acceder al Modelo:
  - por ejemplo para representar una gráfica con datos contenidos en el Modelo
- El modelo es independiente:
  - ni siquiera “sabe” que existen la Vista y el Controlador

Ejemplo de utilización del patrón MVC

Aplicación de gestión de un colegio



2.3 Ventanas

Clase `JFrame`: ventana con título y marco

- Algunos métodos heredados por la clase `JFrame`:

Método	Descripción
`void setLocation(int x, int y)`	Posiciona la ventana en la pantalla. (x, y) son las coordenadas de su vértice superior izquierdo
`void setSize(int ancho, int alto)`	Cambia el tamaño de la ventana
`void setVisible(boolean b)`	Hace la ventana visible o la oculta (cuando b es false)
`void pack()`	Reduce la ventana alrededor de sus componentes
`void setDefaultCloseOperation (int acción);`	Acción al cerrar la ventana (`JFrame.EXIT_ON_CLOSE`)

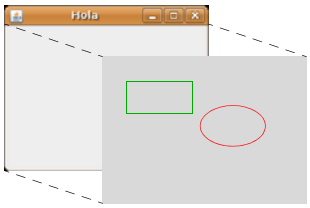
Ejemplo:

```
import javax.swing.*;
public class VentanaHola extends JFrame {
  public VentanaHola() {
    super("Hola");
    setLocation(50,100);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(250, 200);
    setVisible(true);
  }
  public static void main(String[] args) {
    new VentanaHola();
  }
}
```

Panel de contenidos (”*content pane*”)

Cada ventana tiene un panel asociado: “*content pane*”

- **sobre él se dibujan los componentes y los gráficos**



Para obtener o cambiar el *content pane* existen los métodos:

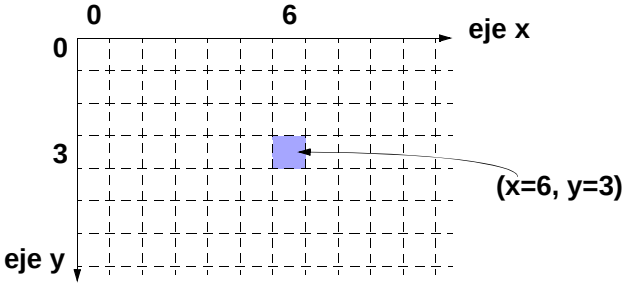
```
Container getContentPane()
void setContentPane(Container contentPane)
```


2.4 Dibujar gráficos en un panel

Los dibujos se realizan sobre un contexto gráfico

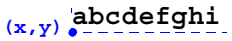
- **representado por un objeto de la clase `Graphics`**
- **los paneles tienen un contexto gráfico asociado**

Sistema de coordenadas:



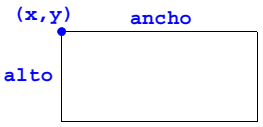
Algunos métodos de la clase `Graphics` (1/2)

```
void drawString(String str,
                int x, int y)
```

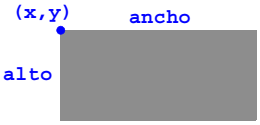



```
void drawLine(int x1, int y1,
              int x2, int y2)
```


```
void drawRect(int x, int y,
              int ancho,
              int alto)
```

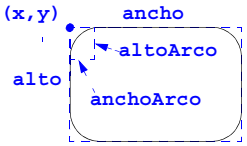



```
void fillRect(int x, int y,
              int ancho,
              int alto)
```

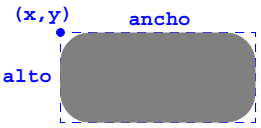


Algunos métodos de la clase `Graphics` (2/2)

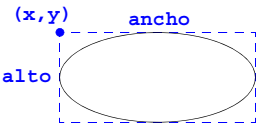
```
void drawRoundRect(int x, int y,
                   int ancho,
                   int alto,
                   int anchoArco,
                   int altoArco)
```




```
void fillRoundRect(int x, int y,
                   int ancho,
                   int alto,
                   int anchoArco,
                   int altoArco)
```




```
void drawOval(int x, int y,
              int ancho,
              int alto)
```



Control del color (1/2)

Los gráficos se dibujan en el color vigente

La clase `Graphics` proporciona dos métodos para cambiar y leer el color vigente:

```
public void setColor(Color c);
public Color getColor();
```

La clase `Color` proporciona constantes predefinidas para algunos colores:

```
public final static Color red
public final static Color green
public final static Color blue
public final static Color yellow
public final static Color white
public final static Color black
...
```

Control del color (2/2)

La clase `Color` también proporciona constructores para crear nuestros propios colores:

```
public Color(int r, int g, int b)
// r, g y b en el rango [0, 255]

public Color(float r, float g, float b)
// r, g y b en el rango [0.0, 1.0]
```

La forma de utilizar los constructores de la clase `Color` es:

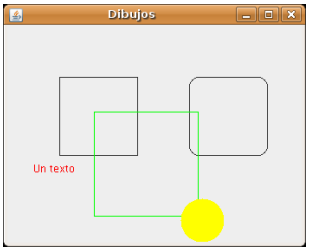
```
g.setColor(new Color(240, 35, 0));

// estas dos líneas son equivalentes
g.setColor(new Color(1.0, 1.0, 0.0));
g.setColor(Color.yellow);
```


Ejemplo de gráficos

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

class DibujaEnVentana extends JFrame{

  public static void main(String[] args) {
    new DibujaEnVentana("Dibujos");
  }

  public DibujaEnVentana(String título) {
    super(título);
    setContentPane(new PanelDibujos());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 300);
    setVisible(true);
  }
```



```
  public class PanelDibujos extends JPanel {

     public void paintComponent(Graphics g) {
      super.paintComponent(g);
      g.drawRect(60, 60, 90, 90);
      g.drawRoundRect(210, 60, 90, 90, 20, 20);
      g.setColor(Color.red);
      g.drawString("Un texto", 30, 170);
      g.setColor(new Color(0,255,0));
      g.drawRect(100, 100, 120, 120);
      g.setColor
        (new Color(255,255,0));
      g.fillOval(200,200,50,50);
    }
  }

} // fin clase DibujaEnVentana
```



Repintado de componentes

Un componente (objeto de una subclase de `JComponent`) se repinta desde el método:

```
void paint(Graphics g)
```

- **el parámetro `g` es el contexto gráfico asociado al componente**

`paint` es invocado automáticamente por el sistema cuando:

- **el componente (o una parte de él) se hace visible**
- **cambia el tamaño del componente**

El método `paint` borra el contexto gráfico del componente e invoca los siguientes tres métodos en el orden indicado:

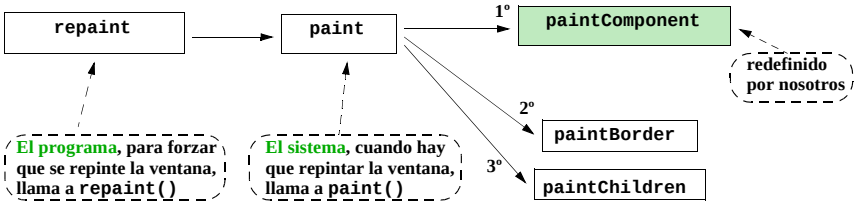
1. `void paintComponent(Graphics g)`: **le vamos a redefinir para dibujar en el componente**
2. `void paintBorder(Graphics g)`
3. `void paintChildren(Graphics g)`

Para realizar un dibujo crearemos una clase derivada de `JPanel` que redefina el método `paintComponent`

- **el método redefinido debe invocar a `super.paintComponent`**
- **y posteriormente realizar el dibujo deseado**

`paint` y `paintComponent` *¡nunca deben invocarse directamente!*

- **se invocan indirectamente mediante el método `repaint`:**
  `void repaint()`
- **`repaint` llama a `paint` que llama a `paintComponent`**



Ejemplo `repaint` y `paintComponent`

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

public class MuevePelota extends JFrame {

  public static void main(String[] args) {
    JFrame f = new MuevePelota();
    // Llama a repaint cada 50ms
    while (true) {
      try {
        Thread.sleep(50); // Espera 50ms
      } catch (InterruptedException e) { }
      f.repaint();
    }
  }
```

```
  public MuevePelota() {
    super("Pelota");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setContentPane(new PanelPelota());
    setSize(450, 500);
    setVisible(true);
  }

  public class PanelPelota extends JPanel {
    // coordenadas del centro de la pelota
    private int xCentro = 0;
    private int yCentro = 0;
    // componentes de la velocidad
    private int vx = 5; // (en pixels/periodo main)
    private int vy = 5; // (en pixels/periodo main)
    // radio de la pelota
    private final int radio = 50;
```



```
    @Override
    protected void paintComponent(Graphics g) {
      super.paintComponent(g);
      // calcula nueva posición del centro
      xCentro=xCentro + vx; yCentro=yCentro + vy;
      // comprueba si choca contra los bordes
      if ((vx>0 && xCentro + radio > getWidth()) ||
          (vx<0 && xCentro - radio < 0))
        vx = -vx;
      if ((vy>0 && yCentro + radio > getHeight()) ||
          (vy<0 && yCentro - radio < 0))
        vy = -vy;
      // pinta la pelota
      g.setColor(Color.red);
      g.fillOval(xCentro-radio, yCentro-radio,
          radio*2, radio*2);
    }
  } // fin clase PanelPelota
} // fin clase MuevePelota
```


2.5 Paneles y Gestores de Distribución

Swing define varios tipos de paneles contenedores:

- **`JPanel`, `JScrollPane`, `JSplitPane` y `JTabbedPane`**

Para añadir componentes a un panel se utilizan los métodos:

```
Component add(Component comp)
Component add(Component comp, int index)
```

Para eliminar componentes existe el método:

```
void remove(Component comp)
```

Por ejemplo, para añadir un botón a un `JPanel`, ponemos añadir estas líneas en su constructor:

```
Button b = new Button("Etiqueta botón");
add(b);
```

Algunos métodos de los paneles contenedores

```
void setBackground(Color c)
Color getBackground()
```

- Permite cambiar y obtener el color de fondo

```
int getWidth()
int getHeight()
```

- Retorna la altura y anchura del componente

```
void setLayout(LayoutManager mgr)
```

- Permite configurar la forma en que se distribuyen los componentes en el contenedor

```
void setPreferredSize(Dimension preferredSize)
```

- Determina el tamaño del componente, p.e.:

```
panel.setPreferredSize(new Dimension(200,100));
```


Añadir componentes a una ventana

Para añadir componentes a una ventana (`JFrame`) hay que añadirles a su *content pane*

- En el constructor de un `JFrame` podrían aparecer las siguientes líneas de código:

```
Button b = new Button("Etiqueta botón");
getContentPane().add(b);
```

Por comodidad los métodos `add`, `remove` y `setLayout` de un `JFrame` están redefinidos para que operen sobre su *content pane*

- por consiguiente, escribir en el constructor de un `JFrame`:

```
add(b);
```

- es lo mismo que escribir:

```
getContentPane().add(b);
```


Gestores de distribución (1/3)

Permiten determinar la forma en que se distribuyen los componentes dentro de un contenedor

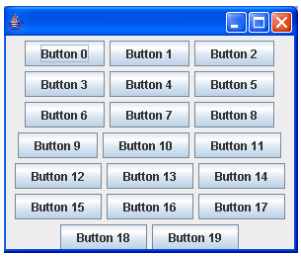
Veremos los tres más comunes:

- **`FlowLayout`, `GridLayout` y `BorderLayout` (gestor de distribución por omisión en todos los paneles contenedores)**

`FlowLayout`:

- **Los componentes se van colocando de izquierda a derecha**
- **Cuando una línea se completa se pasa a la siguiente**
- **Mantiene el tamaño de los componentes**
- **Constructor:**

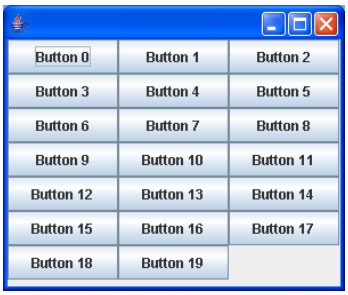
```
FlowLayout()
```



Gestores de distribución (2/3)

`GridLayout`:

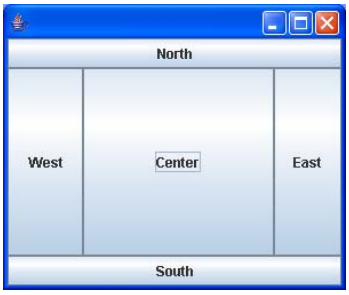
- Parrilla de celdas iguales que se rellena de izquierda a derecha, línea a línea
- Cambia el tamaño de los componentes
- El número de filas o columnas se indica en el constructor
  - el otro parámetro se deja a 0
  - es calculado automáticamente por el gestor de distribución
- Constructor:
  `GridLayout(int filas, int columnas)`



Gestores de distribución (2/3)

`BorderLayout`:

- Coloca los componentes en los 4 puntos cardinales y en el centro
- El del centro se expande para ocupar el mayor área posible
- El resto ocupan el menor espacio posible
- Constructor:
  `BorderLayout()`

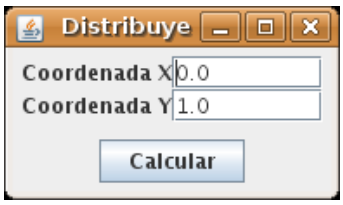


- El método `add` tiene un parámetro adicional para indicar la zona
  `Component add(Component comp, int index)`

```
BorderLayout.NORTH   BorderLayout.SOUTH
BorderLayout.WEST    BorderLayout.EAST
          BorderLayout.CENTER
```


Ejemplo: Distribución de componentes

Escribir el constructor de una ventana que logre la siguiente distribución de los componentes:



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

public class VentanaDistribuye extends JFrame {

  public static void main(String[] args) {
    new VentanaDistribuye("Distribuye");
  }

  // componentes
  private JTextField textX = new JTextField("0.0");
  private JTextField textY = new JTextField("1.0");
  private JButton bCalcular =
                   new JButton("Calcular");
```

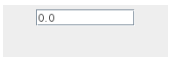


```
  public VentanaDistribuye(String title){
    super(title);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // panel 1 (grid): etiquetas y campos de texto
    JPanel p1 = new JPanel(new GridLayout(2,0));
    p1.add(new JLabel("Coordenada X"));
    p1.add(textX);
    p1.add(new JLabel("Coordenada Y"));
    p1.add(textY);

    // panel 2 (flow): evita que se expanda el botón
    JPanel p2 = new JPanel(new FlowLayout());
    p2.add(bCalcular);

    // panel 3 (flow): evita que se expanda el p1
    JPanel p3 = new JPanel(new FlowLayout());
    p3.add(p1);
```

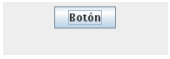


```
    // añade paneles al ”content pane” de la ventana
    add(p3, BorderLayout.CENTER);
    add(p2, BorderLayout.SOUTH);

    pack();
    setVisible(true);
  }
}
```

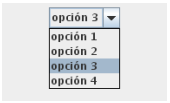
2.6 Componentes



JTextField

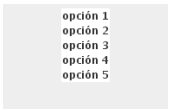


JButton



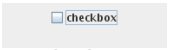
JComboBox

JLabel



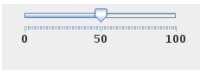
JList

JRadioButton

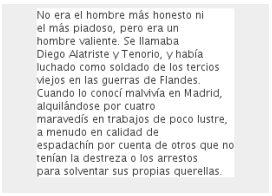


JCheckBox

ButtonGroup



JSlider



JTextArea

JLabel

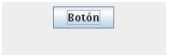
Texto de sólo lectura

- **Constructor**
  `JLabel(String text)`
- **Retorna y cambia el texto de la etiqueta**
  `String getText()`
  `void setText(String text)`

JLabel

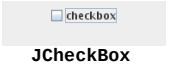
Botones

JButton: botón que puede ser pulsado
`JButton(String text)`



JButton

JCheckBox: casilla que puede ser marcada
`JCheckBox()`
`JCheckBox(String text)`
`JCheckBox(String text, boolean state)`



JCheckBox

JRadioButton: casilla que puede ser marcada
`JRadioButton(String text)`
`JRadioButton(String text, boolean selected)`

JRadioButton

Además todas heredan de `AbstractButton`:
`String getText()` `void setText(String text)`
`boolean isSelected()` `void setSelected(boolean b)`

Grupo de botones: `ButtonGroup`

Permite agrupar varias casillas de forma que no pueda estar marcada más de una simultáneamente

- **Constructor:**
  ```
  ButtonGroup()
  ```
- **Añade un botón al grupo:**
  ```
  void add(AbstractButton b)
  ```

`ButtonGroup`

`JTextField`

Línea de texto que puede ser editada por el usuario

- **Constructores:**
  ```
  JTextField()
  JTextField(int columns)
  JTextField(String text)
  JTextField(String text,
             int columns)
  ```

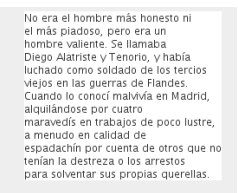
`JTextField`

- **Retorna y cambia el texto escrito en el campo de texto:**
  ```
  String getText()
  void setText(String t)
  ```


`JTextArea`

Area de texto de varias líneas que puede ser configurada para ser editada por el usuario o de sólo lectura

- **Constructores:**
  ```
  JTextArea()
  JTextArea(int rows, int columns)
  JTextArea(String text)
  JTextArea(String text, int rows,
            int columns)
  ```

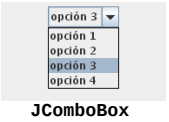


`JTextArea`

- **Configura como editable o no:**
  ```
  void setEditable(boolean b)
  ```
- **Retorna y cambia el texto escrito en el área de texto:**
  ```
  String getText()
  void setText(String t)
  ```

JComboBox

Menú desplegable con varias opciones. La opción elegida en cada momento aparece en el nombre del menú



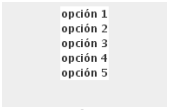
JComboBox

- **Constructor:**
```
JComboBox()
JComboBox(Object[] items)
```
- **Retorna la opción seleccionada (su nombre o su posición):**
```
Object getSelectedItem()
int getSelectedIndex()
```
- **Añade una opción al final de la lista:**
```
void addItem(Object anObject)
```

Ejemplo:
```
String[] opciones = {"opción 1", "opción 2",
                     "opción 3", "opción 4"};
JComboBox cb = new JComboBox(opciones);
```


JList

Lista de opciones. Puede configurarse para que permita elegir una o varias opciones



JList

- **Configura el tipo de selección**
```
void setSelectionMode(int selectionMode)
```

```
ListSelectionModel.SINGLE_SELECTION
ListSelectionModel.SINGLE_INTERVAL_SELECTION
ListSelectionModel.MULTIPLE_INTERVAL_SELECTION (valor por defecto)
```

- **Retorna la opción u opciones seleccionadas**
```
int getSelectedIndex()
int[] getSelectedIndices()
Object getSelectedValue()
Object[] getSelectedValues()
```
- **Cambia la opción u opciones seleccionadas**
```
void setSelectedIndex(int index)
void setSelectedIndices(int[] índices)
```

JList (cont.)

Cada `JList` tiene asociado un objeto de la clase `ListModel`

- **es el que permite añadir y eliminar opciones de la lista**
- **nosotros utilizaremos el `DefaultListModel` (el más sencillo)**

Se configura en el constructor:

```
JList lst = new JList(new DefaultListModel());
```

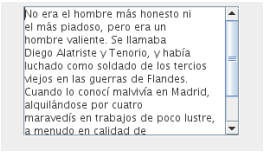
Se obtiene con el método:

```
ListModel getModel()
```

Tiene métodos para añadir, eliminar, ...

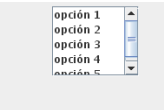
```
void add(int index, Object element)
void addElement(Object obj)
Object remove(int index)
Object set(int index, Object element)
void clear()
int size()
...
```


Uso de `JScrollPane`



Panel que incluye barras de desplazamiento

- **sirve para desplazarse por componentes demasiado grandes para ser mostrados en su totalidad**
- **típicamente usado con áreas de texto y listas**



- **Constructores**

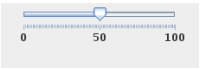
```
JScrollPane(Component view)
JScrollPane(Component view,
            int vsbPolicy,
            int hsbPolicy)
```

```
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED
ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER
ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS
```

```
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS
```


`JSlider`

Barra de desplazamiento



`JSlider`

- **Constructores:**

```
JSlider() // rango 0..100
JSlider(int min, int max, int value)
JSlider(int orientation, int min, int max,
        int value)
```

```
SwingConstants.VERTICAL
SwingConstants.HORIZONTL
```

- **Retorna el valor actual de la barra de desplazamiento:**

```
int getValue()
```

- **Espacio entre "*ticks*" mayores y menores**

```
void setMajorTickSpacing(int n)
void setMinorTickSpacing(int n)
```

- **Muestra "*ticks*" y etiquetas**

```
void setPaintLabels(boolean b)
void setPaintTicks(boolean b)
```

Ejemplo JComboBox y ButtonGroup


```
import java.awt.*;
import javax.swing.*;
class VentanaFormas extends JFrame{
  private final String[] colores =
          {"Rojo", "Azul", "Verde"};
  // Componentes
  private ButtonGroup bgForma = new ButtonGroup();
  private JRadioButton rbCírculo =
    new JRadioButton("Círculo", true);
  private JRadioButton rbCuadrado =
    new JRadioButton("Cuadrado", false);
  private JRadioButton rbRectángulo =
    new JRadioButton("Rectángulo", false);
  private JPanel panelForma = new Dibujo();
  private JComboBox comboColor =
                        new JComboBox(colores);
```

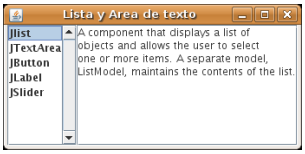


```
  // constructor de la ventana
  public VentanaFormas(String título) {
    super(título);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // añade botones al ButtonGroup
    bgForma.add(rbCírculo);
    bgForma.add(rbCuadrado);
    bgForma.add(rbRectángulo);
    // panel 1 (grid): CheckboxGroup forma
    JPanel p1 = new JPanel(new GridLayout(0,1));
    p1.add(rbCírculo);
    p1.add(rbCuadrado);
    p1.add(rbRectángulo);
    // panel 2 (flow): línea superior
    JPanel p2 = new JPanel(new FlowLayout());
    p2.add(p1);
    p2.add(comboColor);
```



```
    // posiciona paneles en la ventana
    add(p2, BorderLayout.NORTH);
    add(panelForma, BorderLayout.CENTER);
    // asocia manejadores de eventos
    ... // lo veremos en el apartado 2.7
    pack();
    setVisible(true);
  }

  ... // panelForma y manejadores de eventos

  // main
  public static void main(String[] args) {
    new VentanaFormas("Formas");
  }
}
```

Ejemplo JList, JTextArea y JScrollPane




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

public class ListaYArea extends JFrame {

  String[] componentes = {"Jlist", "JTextArea",
       "JButton", "JLabel", "JSlider"};

  String[] descripción =
  {"A component that displays a list of\n" +
   "objects and allows the user to select\n" +
   "one or more items. A separate model,\n" +
   "ListModel, maintains the contents of the list.",
   ...
  };
```



```
  // Componentes
  JList lstComponentes = new JList(componentes);
  JTextArea taDescripción =
                    new JTextArea(descripción[0]);

  // Constructor
  public ListaYArea() {
    super("Lista y Area de texto");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // lista de selección única
    lstComponentes.setSelectionMode(
           ListSelectionModel.SINGLE_SELECTION);

    // seleccionado el primer elemento
    lstComponentes.setSelectedIndex(0);
```



```
    // añade componentes a paneles de desplazamiento
    JScrollPane spLista = new JScrollPane(
      lstComponentes,
      ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
      ScrollPaneConstants.
                    HORIZONTAL_SCROLLBAR_AS_NEEDED);
    JScrollPane spTexto =
                   new JScrollPane(taDescripción);
    // añade los paneles a la ventana
    add(spLista, BorderLayout.WEST);
    add(spTexto, BorderLayout.CENTER);
    // asocia manejadores de eventos
    ... // lo veremos en el apartado 2.7
    pack();
    setVisible(true);
  }
  ... // manejadores y main
```

Ejemplo: `DefaultListModel`

Ejemplo muy sencillo que muestra cómo se obtiene el "ListModel" de una JList y se añaden opciones a la lista


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

public class VentLstMdl extends JFrame {
  private JList lst =
              new JList(new DefaultListModel());

  public static void main(String[] args) {
    LinkedList<String> lstOpciones =
      new LinkedList<String>(Arrays.asList("uno",
                "dos", "tres", "cuatro", "cinco"));
    new ListModel(lstOpciones);
  }
```



```
  // Constructor
  public VentLstMdl(LinkedList<String> lstOpciones){
    super("Muestra LinkedList en JList");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // añade la lista a la ventana
    add(lst, BorderLayout.CENTER);

    // añade dinámicamente las opciones a la lista
    // usando su ListModel
    DefaultListModel lstModel =
              (DefaultListModel) lst.getModel();
    for(String s: lstOpciones)
      lstModel.addElement(s);

    pack();
    setVisible(true);
  }
} // ListModel
```


2.7 Gestión de eventos

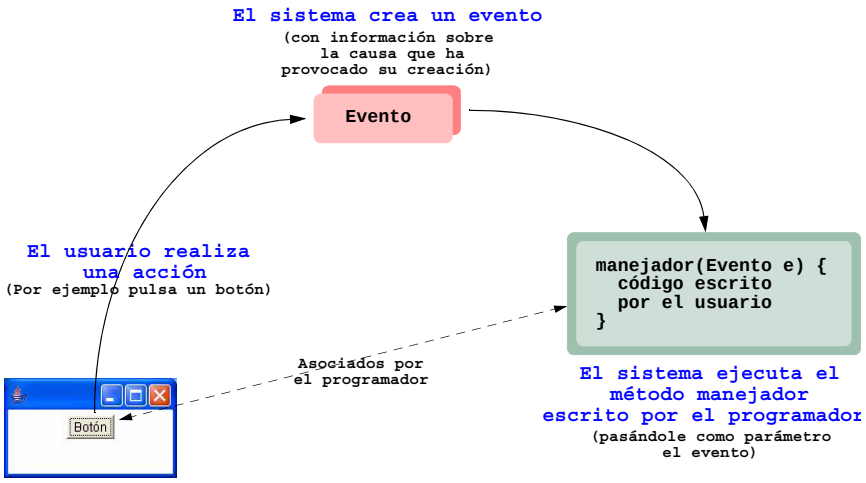
***Evento*: acción asíncrona, generalmente provocada por el usuario**

- **pulsación del botón del ratón, pulsación de tecla, modificación del estado de un componente, etc.**

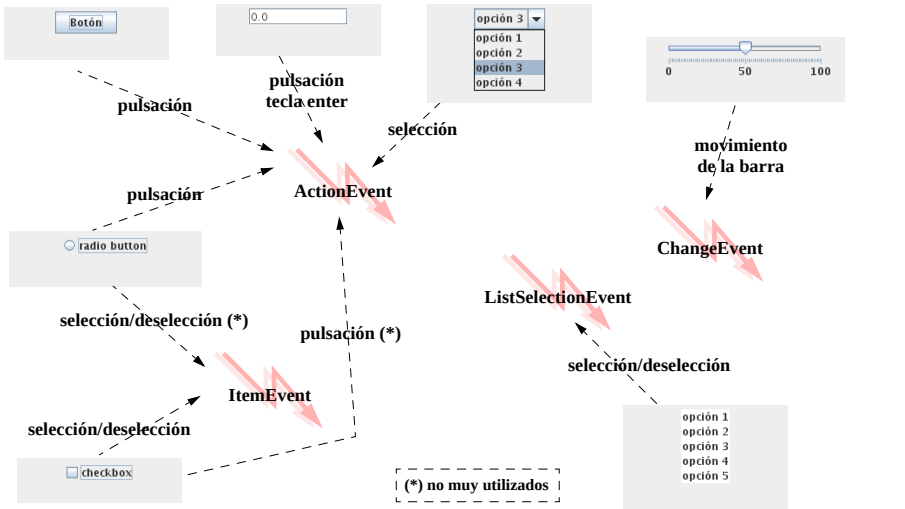
En un programa *dirigido por eventos*:

- **el programador escribe métodos para manejar los eventos**
- **la aplicación espera a que llegue un evento**
- **cuando el sistema detecta un evento llama a su manejador asociado**
- **el manejador debe terminar pronto**
  - **no debe quedarse esperando (p.e., pidiendo datos al usuario)**
- **el usuario dispara la ejecución de las acciones**
  - **distinto de las aplicaciones "tradicionales", donde las acciones las iniciaba la propia aplicación**

Generación de eventos



Tipos de eventos



Los eventos se manejan mediante una *clase manejadora* que implemente la interfaz apropiada

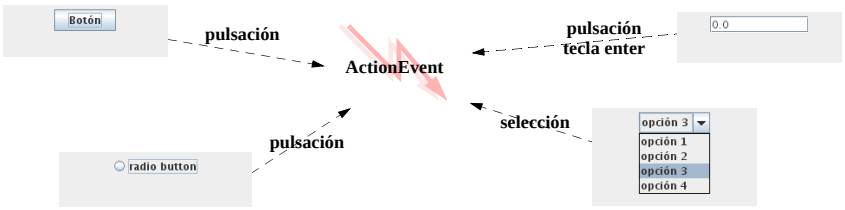
Evento	Interfaz manejadora	método manejador
`ActionEvent`	`ActionListener`	`void actionPerformed (ActionEvent e)`
`ChangeEvent`	`ChangeListener`	`void stateChanged (ChangeEvent e)`
`ItemEvent`	`ItemListener`	`void itemStateChanged (ItemEvent evt)`
`ListSelectionEvent`	`ListSelectionListener`	`void valueChanged (ListSelectionEvent evt)`

Los eventos heredan de `java.util.EventObject` el método:

```
Object getSource() // retorna el componente que
                   // ha producido el evento
```

Existen otros eventos que veremos en el apartado 2.9, "Eventos de bajo nivel"

Eventos de la clase `ActionEvent`



Se manejan utilizando una clase que implemente la interfaz:

```
public interface ActionListener
                              extends EventListener {
  public void actionPerformed(ActionEvent evt);
}
```

La asociación de la clase manejadora con el componente se realiza mediante el método:

```
  void addActionListener(ActionListener l)
```


Ejemplo: evento generado por un botón

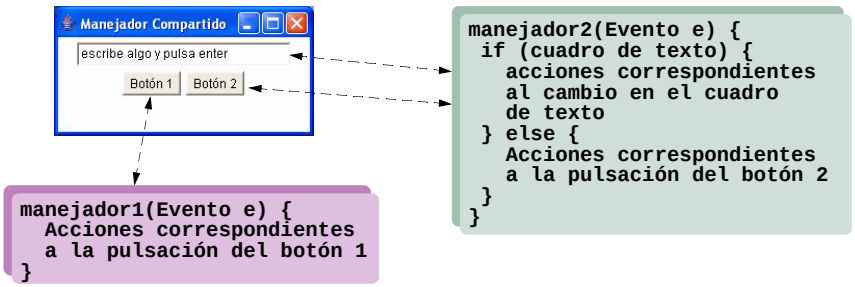
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class VentanaBotón extends JFrame {
  private JButton b = new JButton("Botón");
  // Constructor
  public VentanaBotón(String título) {
    super(título);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new FlowLayout());
    add(b);
    // asocia el manejador con el botón
    b.addActionListener(new ManejaEventoBoton());
    setSize(300, 250);
    setVisible(true);
  }
```



```
  // manejador de eventos del botón
  class ManejaEventoBoton implements ActionListener{
    // Manejador del evento "pulsación de botón"
    public void actionPerformed(ActionEvent e) {
      System.out.println("Botón pulsado");
    }
  }

  // Método main
  public static void main(String[] args) {
    // Crea la ventana
    new VentanaBotón("Test Botón");
  }

}
```

Manejadores compartidos

Podemos utilizar un manejador para cada componente o compartir el mismo manejador entre varios componentes



Dentro de un manejador compartido podemos conocer el componente que ha generado el evento con el método:

```
Object getSource() // de la clase EventObject
```


Ejemplo: manejador compartido


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

class VentanaManejadorCompartido extends JFrame{
  private JButton b1 = new JButton("Botón 1");
  private JButton b2 = new JButton("Botón 2");
  private JTextField tf =
      new JTextField("escribe algo y pulsa enter");
  // constructor
  public VentanaManejadorCompartido(String título) {
    super(título);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new FlowLayout());
    // añade componentes
    add(tf);
    add(b1);
```



```
    add(b2);

    // asocia manejadores
    b1.addActionListener(new ManejaEventosB1());
    ActionListener l = new ManejaEventosB2TF();
    b2.addActionListener(l);
    tf.addActionListener(l);

    pack();
    setVisible(true);
  }

  // manejador de eventos del botón 1
  class ManejaEventosB1 implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println("Manejador1:pulsado b1");
    }
  }
```

```
  // manejador de eventos compartido por b2 y tf
  class ManejaEventosB2TF implements ActionListener{
    public void actionPerformed(ActionEvent e) {
      System.out.print("Manejador2: ");
      if (e.getSource()==tf){ // Evento del tf
        System.out.println("Pulsado enter en tf");
      } else { // Evento del b2
        System.out.println("Pulsado b2");
      }
    }
  }
  // método main
  public static void main(String[] args) {
    new VentanaManejadorCompartido(
            "Manejador compartido");
  }
}
```

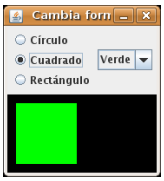

Ejemplo: `VentanaFormas` con eventos

Los componentes principales son:

- **`VentanaFormas` y `PanelForma`**

Las haremos en clases separadas e independientes

- **con sus atributos privados**
- **está será la estrategia a seguir en aplicaciones más complejas**



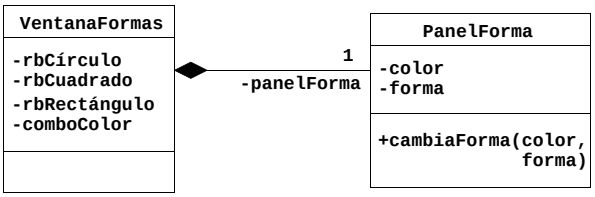


Diagrama de clases de la "Vista" de nuestra aplicación


```
/**
 * Clase VentanaFormas
 */
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class VentanaFormas extends JFrame{
  private final String[] colores = {"Rojo", "Azul", "Verde"};
  // Componentes
  private ButtonGroup bgForma = new ButtonGroup();
  private JRadioButton rbCírculo =
    new JRadioButton("Círculo", true);
  private JRadioButton rbCuadrado =
    new JRadioButton("Cuadrado", false);
  private JRadioButton rbRectángulo =
    new JRadioButton("Rectángulo", false);
  private PanelForma panelForma =
    new PanelForma(Color.red, PanelForma.Forma.círculo);
  private JComboBox comboColor = new JComboBox(colores);
```

```
// constructor de la ventana
public VentanaFormas(String título) {
  super(título);
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  // añade botones al ButtonGroup
  bgForma.add(rbCírculo);
  bgForma.add(rbCuadrado);
  bgForma.add(rbRectángulo);

  // panel 1 (grid): CheckboxGroup forma
  JPanel p1 = new JPanel(new GridLayout(0,1));
  p1.add(rbCírculo);
  p1.add(rbCuadrado);
  p1.add(rbRectángulo);

  // panel 2 (flow): línea superior
  JPanel p2 = new JPanel(new FlowLayout());
  p2.add(p1);
  p2.add(comboColor);
```



```
  // posiciona paneles en la ventana
  add(p2, BorderLayout.CENTER);
  add(panelForma, BorderLayout.SOUTH);

  // asocia manejador
  ActionListener manejador = new ManejadorFormas();
  rbCírculo.addActionListener(manejador);
  rbCuadrado.addActionListener(manejador);
  rbRectángulo.addActionListener(manejador);
  comboColor.addActionListener(manejador);

  // finaliza inicialización de la ventana
  pack();
  setResizable(false);
  setVisible(true);

} // fin constructor
```



```
// Manejador de eventos
class ManejadorFormas implements ActionListener {

  private final Color[] colores =
  {Color.red, Color.blue, Color.green};

  public void actionPerformed(ActionEvent e) {
    PanelForma.Forma forma=PanelForma.Forma.círculo;
    // mira la forma seleccionada
    if (rbCuadrado.isSelected())
      forma=PanelForma.Forma.cuadrado;
    if (rbRectángulo.isSelected())
      forma=PanelForma.Forma.rectángulo;

    // actualiza la forma a dibujar
    panelForma.cambiaForma(
        colores[comboColor.getSelectedIndex()], forma);
  }
}
```

```
    // main
    public static void main(String[] args) {
      new VentanaFormas("Cambia forma con eventos");
    }

  } // clase VentanaFormas
```



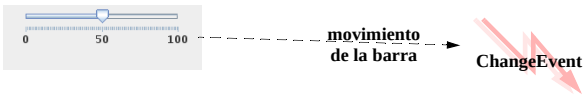
```
/**
 * Clase PanelForma
 */
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.JPanel;

public class PanelForma extends JPanel {

  public static enum Forma
            {círculo, cuadrado, rectángulo};

  // forma y color de la figura a dibujar
  private Forma forma = Forma.círculo;
  private Color color = Color.green;

  // posición y dimensiones de la figura
  private final int xFig = 10, yFig = 10;
  private final int largoFig = 70;
```



```
  @Override
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    setBackground(Color.black); // fondo negro

    g.setColor(color); // color actual
    switch (forma) {
    case círculo:
      g.fillOval(xFig, yFig, largoFig, largoFig);
      break;
    case cuadrado:
      g.fillRect(xFig, yFig, largoFig, largoFig);
      break;
    case rectángulo:
      g.fillRect(xFig, yFig, largoFig*2, largoFig);
      break;
    }
  }
```

```
  // constructor
  public PanelForma(Color color, Forma forma) {
    super();
    setPreferredSize(new Dimension(160, 90));
    this.forma=forma;
    this.color=color;
  }

  // cambia la forma y el color del dibujo
  public void cambiaForma(Color color, Forma forma){
    this.forma=forma;
    this.color=color;
    repaint();
  }

} // fin clase PanelForma
```


Eventos de la clase ChangeEvent



Se manejan utilizando una clase que implemente la interfaz:

```
public interface ChangeListener
                          extends EventListener {
  public void stateChanged(ChangeEvent e);
}
```

La asociación de la clase manejadora con el componente se realiza mediante el método:

```
  void addChangeListener(ChangeListener l)
```


Ejemplo: Cambia colores


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

class VentanaCambiaColores extends JFrame {
  // scrollbars para los colores (R,G y B)
  private JSlider sbR =
    new JSlider(SwingConstants.HORIZONTAL,0,255,255);
  private JSlider sbG =
    new JSlider(SwingConstants.HORIZONTAL,0,255,255);
  private JSlider sbB =
    new JSlider(SwingConstants.HORIZONTAL,0,255,255);
  // campos de texto para los colores
  private JTextField tfR = new JTextField("255");
  private JTextField tfG = new JTextField("255");
  private JTextField tfB = new JTextField("255");
  // panel que muestra el color seleccionado
  private JPanel panelColor = new JPanel();
```

```
// constructor
public VentanaCambiaColores(String título) {
  super(título);
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // distribuye componentes
  GridLayout gl = new GridLayout(0,3);
  gl.setVgap(10); // espacio entre filas
  JPanel pScrolls = new JPanel(gl);
  pScrolls.add(new JLabel("Rojo"));
  pScrolls.add(sbR);
  pScrolls.add(tfR);
  pScrolls.add(new JLabel("Verde"));
  pScrolls.add(sbG);
  pScrolls.add(tfG);
  pScrolls.add(new JLabel("Azul"));
  pScrolls.add(sbB);
  pScrolls.add(tfB);
  add(pScrolls,BorderLayout.NORTH);
  panelColor.setPreferredSize(new Dimension(200,100));
  panelColor.setBackground(Color.white);
  add(panelColor,BorderLayout.CENTER);
```



```
  // crea e instala manejadores
  ChangeListener sbListener =
                      new ManejaEventosScrollbars();
  sbR.addChangeListener(sbListener);
  sbG.addChangeListener(sbListener);
  sbB.addChangeListener(sbListener);
  ActionListener tfListener =
                      new ManejaEventosCamposTexto();
  tfR.addActionListener(tfListener);
  tfG.addActionListener(tfListener);
  tfB.addActionListener(tfListener);

  // finaliza la configuración de la ventana y la hace
  // visible
  pack();
  setResizable(false);
  setVisible(true);
} // fin constructor VentanaCambiaColores
```



```
// manejador eventos de barras de desplazamiento
class ManejaEventosScrollbars
                          implements ChangeListener {

  public void stateChanged(ChangeEvent e) {
    // cambia el color del panel
    panelColor.setBackground(new Color(sbR.getValue(),
         sbG.getValue(),
         sbB.getValue()));

    // actualiza los campos de texto
    tfR.setText(Integer.toString(sbR.getValue()));
    tfG.setText(Integer.toString(sbG.getValue()));
    tfB.setText(Integer.toString(sbB.getValue()));
  }

}
```

```
// manejador de eventos de los campos de texto
class ManejaEventosCamposTexto
                          implements ActionListener {

  public void actionPerformed(ActionEvent e) {
    // actualiza posición de las barras de despl.
    sbR.setValue(Integer.parseInt(tfR.getText()));
    sbG.setValue(Integer.parseInt(tfG.getText()));
    sbB.setValue(Integer.parseInt(tfB.getText()));

    // cambia el color del panel
    panelColor.setBackground(new Color(sbR.getValue(),
        sbG.getValue(),
        sbB.getValue()));
  }
}

public static void main(String[] args) {
  new VentanaCambiaColores("Cambia colores");
}
}// clase VentanaCambiaColores
```


Eventos de la clase `ItemEvent`

Se manejan utilizando una clase que implemente la interfaz:

```
public interface ItemListener
                          extends EventListener{
  public void itemStateChanged(ItemEvent evt);
}
```

La asociación de la clase manejadora con el componente se realiza mediante el método:

```
void addItemListener(ItemListener l)
```


Ejemplo: Dibuja coche

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

/**
 * Clase VentanaCoche
 */
public class VentanaCoche extends JFrame {

  // Componentes
  private JCheckBox cbRuedas =
            new JCheckBox("ruedas", false);
  private JCheckBox cbCarrocería =
            new JCheckBox("carrocería", false);
  private JCheckBox cbVentanas =
            new JCheckBox("ventanas", false);
  private PanelCoche panelCoche = new PanelCoche();
```

```
/** constructor */
public VentanaCoche() {
  super("Ventana Coche");
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  // panel 1 (grid): checkboxes
  JPanel p1 = new JPanel(new GridLayout(0,1));
  p1.add(cbRuedas);  p1.add(cbCarrocería);
  p1.add(cbVentanas);

  // añade componentes a la ventana
  add(p1, BorderLayout.WEST);
  add(panelCoche, BorderLayout.CENTER);

  // asocia manejador
  Manejador m = new Manejador();
  cbRuedas.addItemListener(m);
  cbCarrocería.addItemListener(m);
  cbVentanas.addItemListener(m);

  // termina de configurar la ventana
  pack();    setResizable(false);
  setVisible(true);
}
```



```
/** manejador de los checkboxes */
public class Manejador implements ItemListener {
  HashMap<JCheckBox,PanelCoche.Parte> partes =
    new HashMap<JCheckBox, PanelCoche.Parte>();

  public Manejador() {
    partes.put(cbRuedas,
         PanelCoche.Parte.ruedas);
    partes.put(cbCarrocería,
         PanelCoche.Parte.carrocería);
    partes.put(cbVentanas,
         PanelCoche.Parte.ventanas);
  }

  public void itemStateChanged(ItemEvent e) {
    panelCoche.dibujaParte(
         partes.get(e.getSource()),
         ((JCheckBox)(e.getSource())).isSelected());
  }
}
```



```
/** main */
public static void main(String[] args) {
  new VentanaCoche();
}

}
```

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.*;
/**
 * Clase PanelCoche
 */
public class PanelCoche extends JPanel {
  public enum Parte {ruedas, carrocería, ventanas};
  // atributos
  private boolean dibujaRuedas = false;
  private boolean dibujaCarrocería = false;
  private boolean dibujaVentanas = false;
  // posición de las partes
  private final int ancho=400, alto=200;
  private final int r1X=90, r1Y=140;
  private final int r2X=r1X+200, r2Y=r1Y;
  private final int rDiámetro=50;
  private final int c1X=30, c1Y=100, c1A=70, c1L=350;
  private final int c2A=c1A, c2L=(int)(c1L/1.7);
```

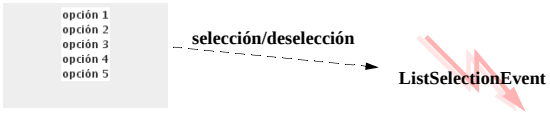


```
  private final int c2X=c1X+100, c2Y=c1Y-c2A+20;
  private final int v1X=c2X+5, v1Y=c2Y+5;
  private final int v1A=c2A-5-20, v1L=80;
  private final int v2X=v1X+v1L+30, v2Y=v1Y;
  private final int v2A=v1A, v2L=c2X+c2L-v2X-5;
  /** constructor */
  public PanelCoche() {
    setPreferredSize(new Dimension(ancho, alto));
  }
  /** dibuja/elimina parte */
  public void dibujaParte(Parte parte, boolean dibujar){
    switch (parte) {
    case ruedas: dibujaRuedas = dibujar;
      break;
    case carrocería: dibujaCarrocería = dibujar;
      break;
    case ventanas: dibujaVentanas = dibujar;
      break;
    }
    repaint();
  }
```



```
  /** paintComponent */
  @Override
  public void paintComponent(Graphics g){
    super.paintComponent(g);
    setBackground(Color.gray);
    if (dibujaRuedas) {
      g.setColor(Color.black);
      g.fillOval(r1X, r1Y, rDiámetro, rDiámetro);
      g.fillOval(r2X, r2Y, rDiámetro, rDiámetro);
    }
    if (dibujaCarrocería) {
      g.setColor(Color.red);
      g.fillRoundRect(c1X, c1Y, c1L, c1A, 20, 20);
      g.fillRoundRect(c2X, c2Y, c2L, c2A, 20, 20);
    }
    if (dibujaVentanas) {
      g.setColor(Color.cyan);
      g.fillRoundRect(v1X, v1Y, v1L, v1A, 20, 20);
      g.fillRoundRect(v2X, v2Y, v2L, v2A, 20, 20);
    }
  }
}
```

Eventos de la clase `ListSelectionEvent`



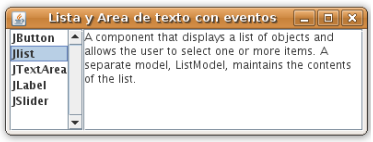
Se manejan utilizando una clase que implemente la interfaz:

```
public interface ListSelectionListener
                              extends EventListener {
  public void valueChanged(ListSelectionEvent evt);
}
```

La asociación de la clase manejadora con el componente se realiza mediante el método:

```
  void addListSelectionListener(
                  ListSelectionListener listener)
```


Ejemplo: Lista Descripción Componentes




```
import java.awt.BorderLayout;
import javax.swing.*;
import javax.swing.event.*;
/**
 * Clase ListaYArea
 */
public class ListaYArea extends JFrame {

  private String[] componentes = {"JButton", "Jlist",
      "JTextArea", "JLabel", "JSlider"};
  private String[] descripción = {
      "An implementation of a \"push\" button.",
      ...
      };

  // Componentes
  private JList lstComp = new JList(componentes);
  private JTextArea taDescrip =
                    new JTextArea(descripción[0]);
```



```
  /** Constructor */
  public ListaYArea() {
    super("Lista y Area de texto con eventos");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // salto de línea automático
    taDescrip.setLineWrap(true);
    taDescrip.setWrapStyleWord(true);
    // lista de selección única
    lstComp.setSelectionMode(
        ListSelectionModel.SINGLE_SELECTION);

    // seleccionado el primer elemento
    lstComp.setSelectedIndex(0);

    // añade componentes a los paneles de desplazamiento
    JScrollPane spLista = new JScrollPane(lstComp,
        ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
        ScrollPaneConstants.
                    HORIZONTAL_SCROLLBAR_AS_NEEDED);
    JScrollPane spTexto = new JScrollPane(taDescrip);
```

```
      // añade los paneles a la ventana
      add(spLista, BorderLayout.WEST);
      add(spTexto, BorderLayout.CENTER);

      // asocia manejador de eventos
      lstComp.addListSelectionListener(
          new ManejadorLista());

      // finaliza la configuración de la ventana
      pack();
      setVisible(true);
    }

    /** manejador de eventos de la lista */
    public class ManejadorLista implements
    ListSelectionListener {
      public void valueChanged(ListSelectionEvent evt) {
        taDescrip.setText(
            descripción[lstComp.getSelectedIndex()]);
      }
    }
```



```
    /** main */
    public static void main(String[] args) {
      new ListaYArea();
    }

  }
```


2.8 Ventanas y diálogos

Un diálogo (clase `JDialog`) es un tipo especial de ventana

- **con la misma apariencia que un `JFrame`**
- **normalmente utilizado para requerir información al usuario**

Puede crearse asociado a otra ventana (`JFrame` o `JDialog`)

- **que se denomina su "dueño" (*owner*)**
- **si el dueño se minimiza o maximiza, el diálogo también lo hace**

Existen dos tipos de diálogos:

- ***modales*: todas las demás ventanas de la aplicación se bloquean, de forma que el usuario sólo puede interaccionar con el diálogo**
- ***no modales*: las demás ventanas no se bloquean**

Algunos métodos de la clase `Dialog`

Constructores

`null` si no se quiere que tenga dueño

```
JDialog(JFrame owner, String título)
JDialog(JFrame owner, boolean ismodal)
JDialog(JFrame owner, String título, boolean ismodal)
JDialog(JDialog owner, String título)
JDialog(JDialog owner, boolean ismodal)
JDialog(JDialog owner, String título, boolean ismodal)
```

Métodos

- Muchos de los disponibles para la clase `Frame`

```
void setLocation(int x, int y)
void setSize(int ancho, int alto)
void setResizable(boolean res)
void setVisible(boolean b)
void pack()
void setLayout(LayoutManager mgr)
void add(Component c)
```

- y alguno específico

```
void setModal(boolean mod)
```


GUIs con varias ventanas y diálogos

Lo normal es que una GUI se componga de una ventana principal y otras ventanas (`JFrame` o `JDialog`) auxiliares

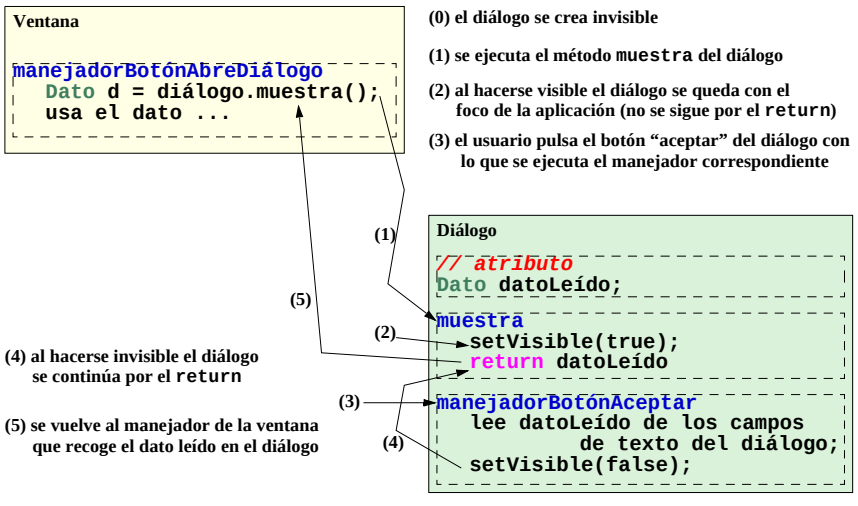
- **no todas las ventanas que componen la aplicación son visibles en un momento dado**
- **se van haciendo visibles o invisibles en respuesta a las acciones del usuario**

Gestión habitual de las ventanas y diálogos auxiliares:

- **las crearemos en el constructor de la ventana principal**
  - **pero no las haremos visibles**
- **los manejadores de eventos las hacen visibles o invisibles en respuesta a las acciones del usuario (usando `setVisible()`)**

Si hay muchas ventanas auxiliares y no se desea malgastar memoria pueden crearse y destruirse a medida que se van necesitando

Relación Ventana ⇔ Diálogo modal



(0) el diálogo se crea invisible

(1) se ejecuta el método `muestra` del diálogo

(2) al hacerse visible el diálogo se queda con el foco de la aplicación (no se sigue por el `return`)

(3) el usuario pulsa el botón "aceptar" del diálogo con lo que se ejecuta el manejador correspondiente

(4) al hacerse invisible el diálogo se continúa por el `return`

(5) se vuelve al manejador de la ventana que recoge el dato leído en el diálogo

Ejemplo: Ventana y Diálogo


```
/**
 * Clase VentanaDiálogo
 */
import javax.swing.*;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.*;

public class VentanaDiálogo extends JFrame {

  // componentes
  private JButton bNombre =
    new JButton("Cambia nombre");
  private JLabel lNombre = new JLabel("¡Hola Pepe!");
  private DiálogoNombre diálogoNombre =
    new DiálogoNombre(this);
```



```
  /** constructor de la ventana */
  public VentanaDiálogo(String título) {
    super(título);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // distribuye componentes
    setLayout(new FlowLayout());
    add(lNombre);
    add(bNombre);

    // asocia el manejador con el botón
    bNombre.addActionListener(new ManejadorBNombre());

    setSize(new Dimension(400,70));
    setVisible(true);
  }
```



```
  /** manejador del evento del botón */
  class ManejadorBNombre implements ActionListener {

    public void actionPerformed(ActionEvent e) {
      // muestra el diálogo de cambio de nombre y
      // recoge el valor retornado
      String nombre = diálogoNombre.muestra();

      // si no se ha cancelado el diálogo cambia
      // el nombre mostrado en la etiqueta
      if ( nombre != null)
        lNombre.setText("¡Hola " + nombre + "!");
    }
  }

  /** main */
  public static void main(String[] args) {
    new VentanaDiálogo("Ventana que crea un Diálogo");
  }

} // fin clase VentanaDiálogo
```

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.BorderLayout;
/**
 * Clase DiálogoNombre
 */
public class DiálogoNombre extends JDialog {
  // String retornado por el diálogo
  private String nombre;
  // componentes
  private JButton bAceptar = new JButton("Aceptar");
  private JTextField tfNombre = new JTextField("Lolo");

  public DiálogoNombre(JFrame owner) {
    // diálogo modal de título "Nombre"
    super(owner, "Nombre", true);

    // añade componentes
    add(tfNombre, BorderLayout.NORTH);
    add(bAceptar, BorderLayout.SOUTH);
```

¡Recordar que debe ser modal!


```
    pack();
    setResizable(false);

    // asocia manejadores a los componentes
    bAceptar.addActionListener(new ManejadorBAceptar());
    tfNombre.addActionListener(new ManejadorBAceptar());
  }


  /**
   * Hace visible el diálogo y espera a que se cierre
   * @return el nombre o null en caso de cancelación
   */
  public String muestra() {
    nombre = null; // valor por si se cancela

    setVisible(true); // hace visible el diálogo

    return nombre; // toma valor en el manejador del
                   // botón
  }
```



```
  /** Manejador del evento del botón */
  class ManejadorBAceptar implements ActionListener {

    public void actionPerformed(ActionEvent e) {
      // lee el nombre introducido
      nombre = tfNombre.getText();

      // oculta el diálogo
      setVisible(false);
    }
  }

} // fin clase DiálogoNombre
```

Diálogos predefinidos (clase `JOptionPane`)

Diálogo para mensajes cortos:

```
public static void showMessageDialog(
  Component parentComponent,
  Object message,
  String title,
  int messageType)
```

```
ERROR_MESSAGE
INFORMATION_MESSAGE
WARNING_MESSAGE
QUESTION_MESSAGE
PLAIN_MESSAGE
```

Ejemplo:

```
JOptionPane.showMessageDialog(ventana,
  "Datos incorrectos", "Error",
  JOptionPane.ERROR_MESSAGE);
```

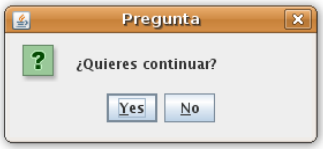
Pide confirmación del usuario:

```
public static int showConfirmDialog(
  Component parentComponent,
  Object message,
  String title,
  int optionType)
```

```
YES_NO_OPTION
YES_NO_CANCEL_OPTION
OK_CANCEL_OPTION
```

Ejemplo

```
int respuesta = JOptionPane.showConfirmDialog(ventana,
  "¿Quieres continuar?", "Pregunta",
  JOptionPane.YES_NO_OPTION);
```



Pide un dato al usuario:

```
public static String showInputDialog(
  Component parentComponent,
  Object message,
  String title,
  int messageType))
```

```
ERROR_MESSAGE
INFORMATION_MESSAGE
WARNING_MESSAGE
QUESTION_MESSAGE
PLAIN_MESSAGE
```

Ejemplo:

```
String ret = JOptionPane.showInputDialog(
  ventana, "Introduce tu nombre",
  "Información requerida",
  JOptionPane.QUESTION_MESSAGE);
```

Diálogo para seleccionar un fichero

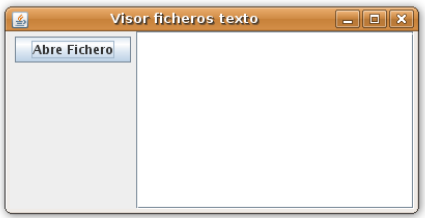
Swing proporciona la clase `JFileChooser`:

```
public JFileChooser() // constructor
int showOpenDialog(Component parent) // muestra
          // el diálogo y retorna CANCEL_OPTION,
          // APPROVE_OPTION o ERROR_OPTION
File getSelectedFile() // retorna el fichero
```

Modo de uso:

```
JFileChooser fc = new JFileChooser();
...
int ret = fc.showOpenDialog(ventana_padre);
if (ret == JFileChooser.APPROVE_OPTION) {
  File file = fc.getSelectedFile();
  ... // la aplicación usa el fichero
} else {
  ... // el usuario ha cancelado el diálogo
}
```


Ejemplo: Visor de ficheros de texto




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

public class VisorFicherosTexto extends JFrame{

  // componentes
  private JButton bAbre = new JButton("Abre Fichero");
  private JTextArea taFich = new JTextArea();


  /** main */
  public static void main(String[] args) {
    new VisorFicherosTexto();
  }
```



```
  /** constructor */
  public VisorFicherosTexto() {
    super("Visor ficheros texto");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // panel para el botón
    JPanel p1 = new JPanel(new FlowLayout());
    p1.add(bAbre);
    // panel para el área de texto
    JScrollPane spTexto = new JScrollPane(taFich);
    // coloca componentes
    add(p1, BorderLayout.WEST);
    add(spTexto, BorderLayout.CENTER);
    // asocia manejador
    bAbre.addActionListener(new manejador());
    setSize(400,200);
    setVisible(true);
  }
```

```
/** manejador del evento del botón */
public class manejador implements ActionListener {
  @Override
  public void actionPerformed(ActionEvent e) {
    JFileChooser fc = new JFileChooser();
    int ret = fc.showOpenDialog(
        VisorFicherosTexto.this);
    if (ret == JFileChooser.APPROVE_OPTION) {
      File file = fc.getSelectedFile();
      try {
        taFich.setText(leeFichero(file.getPath()));
      } catch (Exception excep) {
        JOptionPane.showMessageDialog(
            VisorFicherosTexto.this,
            "Error leyendo fichero", "Error",
            JOptionPane.ERROR_MESSAGE);
      }
    }
  }
}
```



```
/** lee los contenidos de un fichero de texto */
private String leeFichero(String nomFich)
                                  throws IOException {
  String str="", línea;
  BufferedReader ent = null;
  try {
    ent = new BufferedReader(new FileReader(nomFich));
    do {
      línea=ent.readLine(); // lee una línea
      // si hay más líneas seguimos procesando
      if (línea!=null)
        str=str+línea+"\n";
    } while (línea!=null);
  } finally {
    if (ent!=null)
      ent.close();
  } // try
  return str;
 }
}
```


2.9 Eventos de bajo nivel

En el apartado anterior vimos los eventos de alto nivel:

- **`ActionEvent`, `ChangeEvent`, `ItemEvent` y `ListSelectionEvent`**

Existen otros eventos (denominados de bajo nivel) que se producen cuando:

- **la ventana cambia de tamaño o de estado**
- **se pulsa una tecla sobre un componente**
- **el ratón entra o sale de un componente**
- **se produce un movimiento de ratón (`MouseEvent`)**
- **...**

Únicamente veremos los generados por el ratón

Eventos de la clase `MouseEvent` (1/3)

Existen dos tipos de eventos `MouseEvent`:

- ***De movimiento*. Se generan cuando:**
  - **el cursor del ratón se desplaza por la ventana**
- ***De estado del ratón*. Se generan cuando:**
  - **se presiona, se suelta o se hace clic en un botón del ratón**
  - **el cursor del ratón entra o sale de un componente**

Los eventos `MouseEvent` *de movimiento* se manejan utilizando una clase que implemente la interfaz `MouseMotionListener`:

```
public interface MouseMotionListener
                              extends EventListener {
  void mouseDragged(MouseEvent e);
  void mouseMoved(MouseEvent e);
}
```


Eventos de la clase `MouseEvent` (2/3)

Si se prefiere puede extenderse la clase `MouseMotionAdapter` que implementa los dos métodos con cuerpo nulo

La asociación de la clase manejadora con el componente se realiza mediante el método:

```
  void addMouseMotionListener(MouseMotionListener l)
```

Los eventos `MouseEvent` de *estado del ratón* se manejan utilizando una clase que implemente la interfaz `MouseListener`:

```
public interface MouseListener
                              extends EventListener{
  void mouseClicked(MouseEvent e);
  void mouseEntered(MouseEvent e);
  void mouseExited(MouseEvent e);
  void mousePressed(MouseEvent e);
  void mouseReleased(MouseEvent e);
}
```


Eventos de la clase `MouseEvent` (3/3)

Para que no sea necesario escribir una clase que implemente los cinco métodos de la interfaz `MouseListener` existe la clase:

```
public abstract class MouseAdapter
                            implements MouseListener{
    // implementa todos los métodos con cuerpo nulo
  public void mouseClicked(MouseEvent e) {}
  ...
}
```

El programador puede extender la clase `MouseAdapter` y sobrescribir únicamente los métodos en los que esté interesado

La asociación de la clase manejadora con el componente se realiza mediante el método:

```
  void addMouseListener(MouseListener l)
```

Eventos `MouseEvent` en listas

Se pueden usar para detectar si se ha seleccionado una opción de la lista o si se ha hecho un "doble clic"

- **Ejemplo: detección de "doble clic" sobre una `JList`**

```
private class ManejadorDobleClicEnLista
                                  extends MouseAdapter {

  public void mouseClicked(MouseEvent e) {
    if (e.getClickCount() == 2) {
      int index = list.locationToIndex(e.getPoint());
      // en index tenemos el índice de la opción
      // seleccionada
      ...
    }
  }

}
```


Ejemplo: Dibuja con ratón

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

public class VentanaDibujaConRatón extends JFrame {

  // main
  public static void main(String[] args) {
    new VentanaDibujaConRatón("Dibuja con ratón");
  }

  // componentes
  private PanelDibujo pDibujo =
    new PanelDibujo(Color.red);
  private final String[] colores = {"rojo",
                                    "verde", "azul"};
  private JComboBox cColores = new JComboBox(colores);
```



```
  /** Constructor de la ventana */
  public VentanaDibujaConRatón(String title) {
    super(title);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // panel west (flow)
    JPanel pWest = new JPanel(new FlowLayout());
    pWest.add(cColores);
    add(pWest,BorderLayout.WEST);
    add(pDibujo, BorderLayout.CENTER);
    // asocia manejadores de eventos
    pDibujo.addMouseMotionListener(
        new ManejadorMovimientosRaton());
    pDibujo.addMouseListener(
        new ManejadorBotonesRaton());
    cColores.addItemListener(
        new ManejadorEventosChColor());
    pack();
    setVisible(true);
  }
```

```
/** Manejador de eventos del choice de colores */
private class ManejadorEventosChColor
                              implements ItemListener {
  private final Color colores[] =
          {Color.red, Color.green, Color.blue};
  public void itemStateChanged(ItemEvent e){
    pDibujo.cambiaColor(
        colores[cColores.getSelectedIndex()]);
  }
}

/** Manejador de eventos de movimientos del ratón */
private class ManejadorMovimientosRaton
                        extends MouseMotionAdapter {

  public void mouseDragged(MouseEvent e){
    pDibujo.añadePunto(e.getPoint()); // pto actual
  }

}
```



```
/** Manejador de eventos de los botones del ratón */
private class ManejadorBotonesRaton
                                extends MouseAdapter {

  public void mousePressed(MouseEvent e) {
    pDibujo.añadePunto(e.getPoint()); // pto actual
  }

  public void mouseReleased(MouseEvent e) {
    pDibujo.añadePunto(new Point(-1, 0)); //pto fin
  }

}


} // clase VentanaDibujaConRaton
```



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

/** Cavas sobre el que se dibuja con el ratón */
public class PanelDibujo extends JPanel {
  // trayectoria del ratón (lista de puntos con color)
  private class Punto {
    Point p;
    Color c;
    Punto(Point p, Color c) {this.p=p; this.c=c;}
  }
  private LinkedList<Punto> listaPuntos =
    new LinkedList<Punto>(); // ptos en el dibujo
  private Color colorActual; // color seleccionado
  /** constructor */
  public PanelDibujo(Color c) {
    colorActual = c;
    setBackground(Color.black);
    setPreferredSize(new Dimension(300,300));
  }
```

```
/** dibuja los puntos de la trayectoria del ratón */
@Override
public void paintComponent(Graphics g) {
  super.paintComponent(g);
  if (listaPuntos.isEmpty()) return;
  Iterator<Punto> iter = listaPuntos.iterator();
  Punto p1, p0=iter.next();
  while (iter.hasNext()) {
    p1=iter.next();
    // fin de trazo?
    if (p1.p.x != -1) {
      // NO fin: dibuja línea entre dos puntos
      g.setColor(p1.c);
      g.drawLine(p0.p.x, p0.p.y, p1.p.x, p1.p.y);
    } else if (iter.hasNext()) {
      p1=iter.next(); // Fin de trazo: salta el punto
    }
    p0 = p1;
  } // while
}
```



```
/** añade un punto a la trayectoria */
public void añadePunto(Point p){
  listaPuntos.add(new Punto(p, colorActual));
  repaint();
}


/** cambia el color actualmente seleccionado */
public void cambiaColor(Color c){
  colorActual = c;
}


} // clase PanelDibujo
```


2.10 Bibliografía

- **The Java Tutorials: “Creating a GUI with JFC/Swing”. http://java.sun.com/docs/books/tutorial/uiswing/index.html**
- **Eitel, Harvey M. y Deitel, Paul J., “Cómo programar en Java”, quinta edición. Pearson Educación, México, 2004.**