

Interfaces de usuario con AWT y Swing

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es



Objetivos

- Escribir programas para dibujar elementos gráficos
- Escribir programas con interfaces de usuario elaboradas
- Conocer la tecnología Applet
- Conocer las API para mostrar imágenes y reproducir sonidos



Índice

- Abstract Windowing Toolkit (AWT) vs. Swing
- Componentes GUI AWT
- Gráficos
- Gestores de Layout
- Componentes Swing
- Applets
- Modelo de delegación de eventos
- Frame Windows
- Uso de clases Inner para Listeners
- Construcción de aplicaciones con GUIs
- Multimedia con Java



Interfaces de usuario (GUI) con Java

- Java proporciona las clases necesarias para el desarrollo de aplicaciones con interfaces de usuario interactivas
- Las clases proporcionan los componentes GUI necesarios para crear aplicaciones y applets Java
- Las clases derivan de la Java Foundation Classes (JFCs) que es una parte importante del Java SDK que es una colección de cinco APIs
 - AWT, Swing, Java2D, Accessibility, Drag and Drop



AWT y Swing

- AWT (Abstract Windows Toolkit) y Swing son librerías de clases para el desarrollo de GUIs
- Algunos componentes de AWT usan código nativo y por ello es dependiente de la plataforma
- Swing está escrito completamente en Java por lo que es independiente de la plataforma
 - Las aplicaciones distribuidas entre varias plataformas tiene la misma apariencia
 - Se puede considerar como el reemplazo de AWT



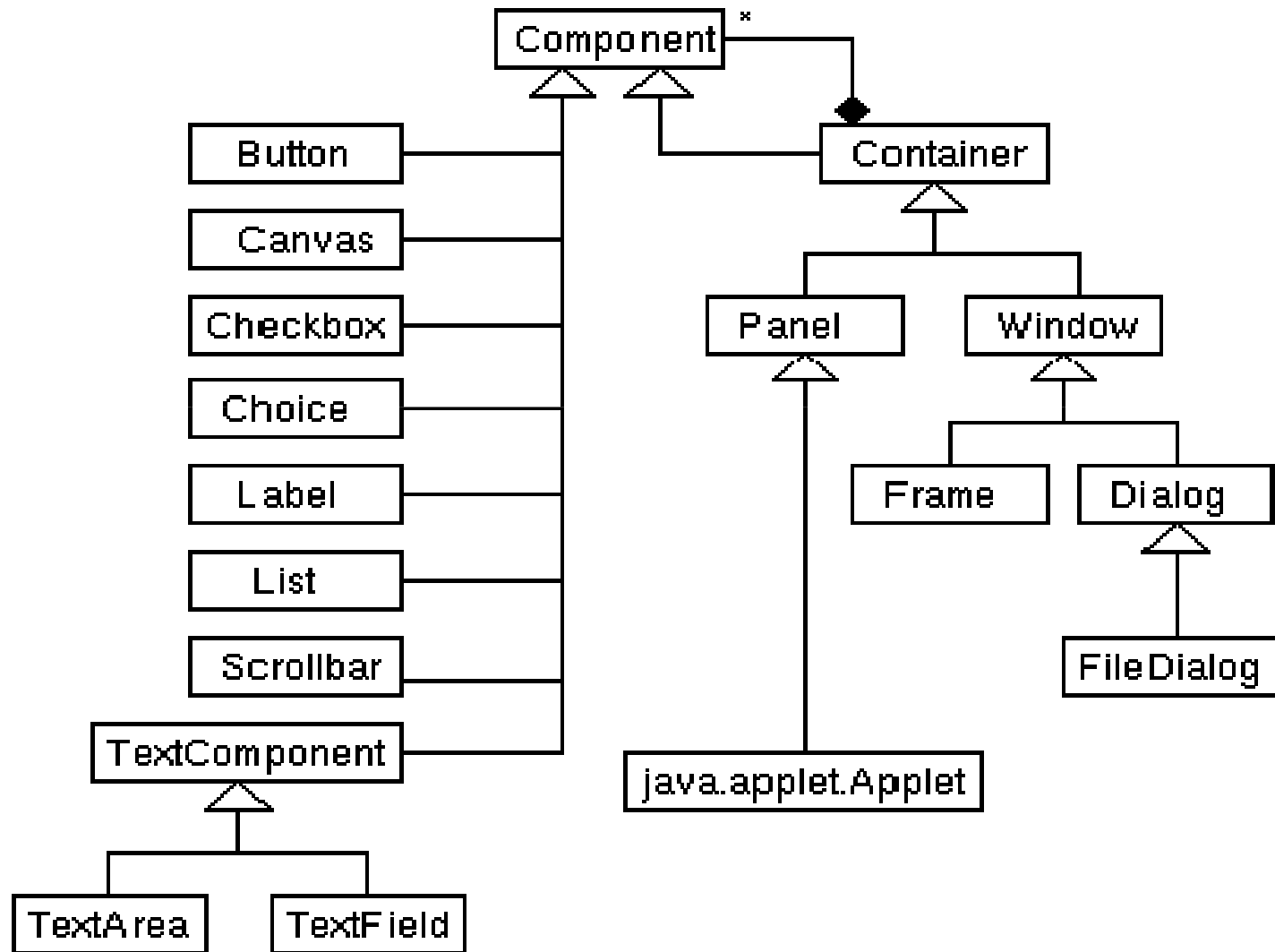
Componentes AWT

Abstract Windows Toolkit (AWT): java.awt

- GUI elements:
 - Container (Panel, Frame, Dialog, etc.)
 - Primitive (Button, Label, Checkbox, Scrollbar, etc.)
- Layout managers: FlowLayout, BorderLayout, etc.
- Supporting classes:
 - Event handling
 - java.awt.event package
 - Graphics
 - Color, Font, Graphics, etc.
 - Geometry
 - Point, Rectangle, Dimension, etc.
 - Imaging
 - Image class and java.awt.image package



Jerarquía de Componentes AWT





Componentes GUI AWT: Clases Window fundamentales

- Los componentes GUI se colocan en contenedores

Clase AWT	Descripción
Component	An abstract class for objects that can be displayed on the console and interact with the user. The root of all other AWT classes
Container	An abstract subclass of the <i>Component</i> class. A component that can contain other AWT components.
Panel	Extends the <i>Container</i> class. A frame or window without the titlebar, the menubar nor the border. Superclass of the Applet class
Window	Extends the <i>Container</i> class. A Window object is a top-level window with no borders and no menubar. (default BorderLayout)
Frame	Entends the <i>Window</i> class. A window with a title, menubar, border, and resizing corners.



Componentes AWT: Métodos de la clase Window

- Asignación del tamaño de ventana:
`void setSize(int width, int height)`
`void setSize(Dimension d)`
donde *Dimension d* tiene *width* y *height* como campos
- Un window por defecto no es visible. Configuración de su visibilidad:
`void setVisible(boolean b)`
Si *b* es true, window es visible



Componentes AWT: Clases fundamentales Window

- Los objetos *Frame* son muy usados en el diseño de GUI

```
import java.awt.*;
/* Frame vacio. Para terminar Ctrl-C */
public class SampleFrame extends Frame {
    public static void main(String args[]) {
        SampleFrame sf = new SampleFrame();
        sf.setSize(100, 100);
        sf.setVisible(true);
    }
}
```



Componentes AWT: Clase Graphics

- Métodos de la clase Graphics (abstract)

<code>drawString()</code>	<code>drawLine()</code>
<code>drawArc()</code>	<code>fillArc()</code>
<code>drawOval()</code>	<code>fillOval()</code>
<code>drawPolygon()</code>	<code>fillPolygon()</code>
<code>drawRect()</code>	<code>fillRect()</code>
<code>drawRoundRect()</code>	<code>fillRoundRect()</code>

- Constructores de la clase Color

<code>Color(int r, int g, int b)</code>	Valor entero entre 0 y 255
<code>Color(float r, float g, float b)</code>	Valor real entre 0 y 1
<code>Color(int rgbValue)</code>	Rango de valores: 0 – $2^{24}-1$ (blanco-negro) Rojo: bits 16-23 Verde: bits 8-15 Azul: bits 0-7



Métodos de la Clase java.awt.Graphics

`drawString (String s, int x, int y)`

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

`void drawRect (int x, int y, int w, int h)`

`void fillRect (int x, int y, int w, int h)`

`void drawOval (int x, int y, int w, int h)`

`void fillOval (int x, int y, int w, int h)`

`void drawRoundRect (int x, int y, int w, int h, int rw, int rh)`

`void fillRoundRect (int x, int y, int w, int h, int rw, int rh)`

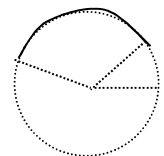
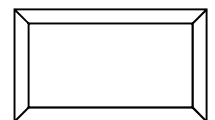
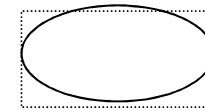
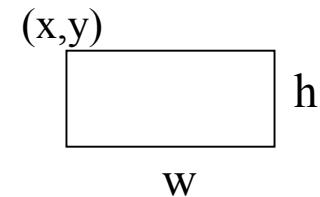
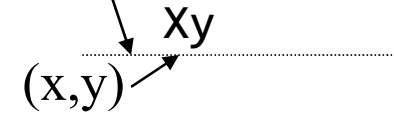
`void draw3DRect (int x, int y, int w, int h, boolean raised)`

`void fill3DRect (int x, int y, int w, int h, boolean raised)`

`void drawArc (int x, int y, int w, int h, int rs, int ra)`

`void fillArc (int x, int y, int w, int h, int rs, int ra)`

Baseline





Métodos de la Clase `java.awt.Graphics`

<code>setColor(color)</code>	asigna color en curso
<code>setFont(font)</code>	asigna font en curso
<code>setPaintMode()</code>	asigna modo de pintado
<code>setXORMode(color)</code>	asigna el modo XOR
<code>getColor()</code>	lee el color en curso
<code>getFont()</code>	lee el font en curso
<code>getFontMetrics()</code>	lee las dimensiones del font en curso
<code>getFontMetrics(font)</code>	lee las dimensiones del font especific.

La clase `Graphics` requiere un `Graphics Context` que es una abstracción de varias superficies de dibujo:

- `screen`
- `printer`
- imagen off-screen (imagen almacenada en memoria)



Ejemplo de Componentes GUI: Graphics

```
import java.awt.*;
public class GraphicPanelDemo extends Panel {
    GraphicPanelDemo() {
        setBackground(Color.black);
    }
    public void paint(Graphics g) {
        g.setColor(new Color(0,255,0));        //green
        g.setFont(new Font("Helvetica",Font.PLAIN,16));
        g.drawString("Hola Mundo GUI!", 30, 100);
        g.setColor(new Color(1.0f,0,0));        //red
        g.fillRect(30, 100, 150, 10);
    }
    public static void main(String args[]) {
        Frame f = new Frame("Prueba de Graphics Panel");
        GraphicPanelDemo gp = new GraphicPanelDemo();
        f.add(gp);
        f.setSize(600, 300);
        f.setVisible(true);
    }
}
```



Más componentes AWT

- Controles AWT
 - Componentes que permiten al usuario interactuar con la aplicación GUI
 - Subclases de la clase Component

Label



Label

Button



Button

TextField



Text Field

TextArea



Text Area

Checkbox



Checkbox

Choice



Choice

List



List

Scrollbar



Scrollbar

ScrollPane



Scroll Pane

Panel



Panel

Canvas



Canvas

MenuBar



Menu Bar

PopupMenu



Popup Menu



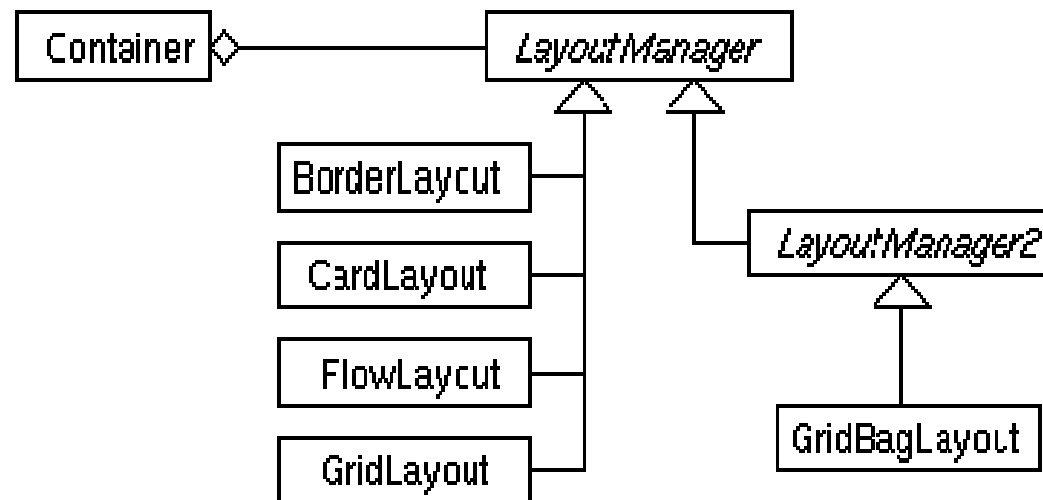
Ejemplo de Componentes GUI

```
import java.awt.*;
class FrameWControlsDemo extends Frame {
    public static void main(String args[]) {
        FrameWControlsDemo fwc = new FrameWControlsDemo();
        fwc.setLayout(new FlowLayout()); fwc.setSize(600, 200);
        fwc.add(new Button("Pruebame!"));
        fwc.add(new Label("Label")); fwc.add(new TextField());
        CheckboxGroup cbg = new CheckboxGroup();
        fwc.add(new Checkbox("chk1", cbg, true));
        fwc.add(new Checkbox("chk2", cbg, false));
        fwc.add(new Checkbox("chk3", cbg, false));
        List list = new List(3, false);
        list.add("MTV"); list.add("V"); fwc.add(list);
        Choice chooser = new Choice();
        chooser.add("Lady Gaga"); chooser.add("Monica");
        chooser.add("Britney");
        fwc.add(chooser); fwc.add(new Scrollbar());
        fwc.setVisible(true);
    }
}
```




Gestores del Layout

- Los gestores de Layout determinan la disposición y tamaño de los componentes dentro de un contenedor
 - Las posiciones y tamaños de los componentes se ajustan automáticamente cuando la ventana cambia de tamaño.
- Los clases de los gestores de layout en Java son:
 - BorderLayout
 - GridLayout
 - GridBagLayout
 - CardLayout





Gestores del Layout: métodos

- Ajuste del gestor de layout

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

Si se pasa null, no se usa un gestor de layout

- Si no se usa un gestor es necesario posicionar los elementos manualmente

```
public void setBounds(int x, int y,  
                    int width, int height)
```

Método de la clase Component

Tedioso cuando se tiene varios objetos puesto que se tiene que usar para cada objeto



Gestor FlowLayout

- Es el gestor por defecto de la clase y subclases *Panel*
 - La clase *Applet* es una subclase de *Panel*
- Coloca los componentes de izquierda a derecha y de arriba abajo, empezando en la esquina superior izquierda

Resumen de Constructores

FlowLayout()	Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.
FlowLayout(int align)	Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.
FlowLayout(int align, int hgap, int vgap)	Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.



Gestor FlowLayout

- Gap
 - Espaciado entre componentes
 - Medido en pixels
- Valores de alineamiento posibles
 - FlowLayout.LEFT
 - FlowLayout.CENTER
 - FlowLayout.RIGHT



Ejemplo de gestor FlowLayout

```
import java.awt.*;  
class FlowLayoutDemo extends Frame {  
    public static void main(String args[]) {  
        FlowLayoutDemo fld = new FlowLayoutDemo();  
        fld.setLayout(new FlowLayout(FlowLayout.RIGHT,  
                                     10, 10));  
        fld.add(new Button("UNO"));  
        fld.add(new Button("DOS"));  
        fld.add(new Button("TRES"));  
        fld.setSize(100, 100);  
        fld.setVisible(true);  
    }  
}
```

Resultado:





Gestor BorderLayout

- Es el gestor por defecto de la clase y subclases *Window*
 - Incluye los tipos *Frame* y *Dialog*
- Divide el objeto *Container* en cinco partes donde se añaden objetos *Component* (North, South, East, West, Center)

Resumen de Constructores

BorderLayout()	Constructs a new border layout with no gaps between components.
BorderLayout(int hgap, int vgap)	Constructs a border layout with the specified gaps between components.

- Los parámetros *hgap* y *vgap* se refieren al espaciado entre los componentes dentro del contenedor



Uso del Gestor BorderLayout

- Para añadir un componente a una región específica:
 - Usar el método `add` y pasar dos argumentos:
 - Componente a añadir
 - Región donde se debe colocar el componente
- Regiones válidas:
 - `BorderLayout.NORTH`
 - `BorderLayout.SOUTH`
 - `BorderLayout.EAST`
 - `BorderLayout.WEST`
 - `BorderLayout.CENTER`



Ejemplo de gestor BorderLayout

```
import java.awt.*;  
class BorderLayoutDemo extends Frame {  
    public static void main(String args[]) {  
        BorderLayoutDemo bld = new BorderLayoutDemo();  
        bld.setLayout(new BorderLayout(10, 10));  
        bld.add(new Button("NORTE"), BorderLayout.NORTH);  
        bld.add(new Button("SUR"), BorderLayout.SOUTH);  
        bld.add(new Button("ESTE"), BorderLayout.EAST);  
        bld.add(new Button("OESTE"), BorderLayout.WEST);  
        bld.add(new Button("CENTRO"), BorderLayout.CENTER);  
        bld.setSize(200, 200);  
        bld.setVisible(true);  
    }  
}
```



Resultado



Gestor GridLayout

- Parecido a FlowLayout
 - La posición de los componentes va de izquierda a derecha y de arriba abajo
 - Empieza a añadir componentes en la esquina superior izquierda
- Divide el contenedor en un número de filas y columnas
 - Las regiones son de igual tamaño
 - Ignora el tamaño del componente principal



Gestor GridLayout

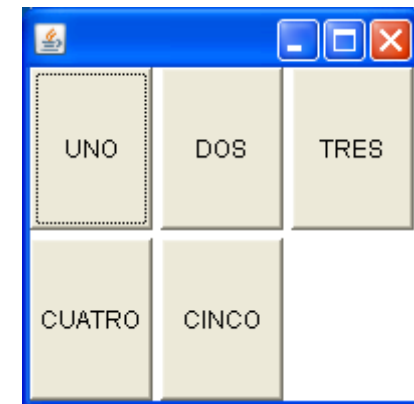
Resumen de Constructores

GridLayout()	Creates a grid layout with a default of one column per component, in a single row.
GridLayout(int rows, int cols)	Creates a grid layout with the specified number of rows and columns.
GridLayout(int rows, int cols, int hgap, int vgap)	Creates a grid layout with the specified number of rows and columns.



Ejemplo de gestor GridLayout

```
import java.awt.*;  
class GridLayoutDemo extends Frame {  
    public static void main(String args[]) {  
        GridLayoutDemo gld = new GridLayoutDemo();  
        gld.setLayout(new GridLayout(2, 3, 4, 4));  
        gld.add(new Button("UNO"));  
        gld.add(new Button("DOS"));  
        gld.add(new Button("TRES"));  
        gld.add(new Button("CUATRO"));  
        gld.add(new Button("CINCO"));  
        gld.setSize(200, 200);  
        gld.setVisible(true);  
    }  
}
```



Resultado



Panels y Layout complejos

- Para layouts complejos:
 - Se puede combinar los diferentes gestores de layouts
 - Uso de panels al mismo tiempo
- Recordar que:
 - Un *Panel* es un *Container* y un *Component*
 - Se puede insertar componentes en un *Panel*
 - Se puede añadir *Panel* a un *Container*



Ejemplo de Panels y Layout complejos

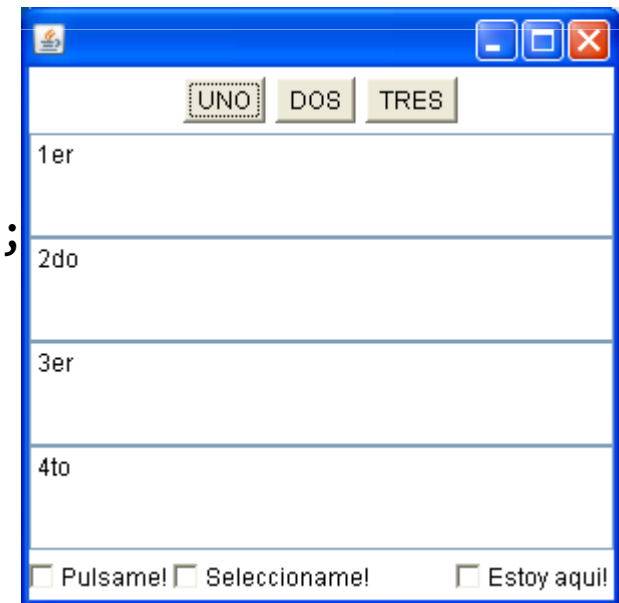
```
import java.awt.*;
class ComplexLayout extends Frame {
    public static void main(String args[]) {
        ComplexLayout cl = new ComplexLayout();
        Panel panelNorth = new Panel();
        Panel panelCenter = new Panel();
        Panel panelSouth = new Panel();
        /* North Panel */
        //Panels usan FlowLayout por defecto
        panelNorth.add(new Button("UNO"));
        panelNorth.add(new Button("DOS"));
        panelNorth.add(new Button("TRES"));
        /* Center Panel */
        panelCenter.setLayout(new GridLayout(4,1));
        panelCenter.add(new TextField("1er"));
        panelCenter.add(new TextField("2do"));
        panelCenter.add(new TextField("3er"));
        panelCenter.add(new TextField("4to"));
    }
}
```



Ejemplo de Panels y Layout complejos

```
/* South Panel */
panelSouth.setLayout(new BorderLayout());
panelSouth.add(new Checkbox("Seleccioname!"),
                BorderLayout.CENTER);
panelSouth.add(new Checkbox("Estoy aqui!"),
                BorderLayout.EAST);
panelSouth.add(new Checkbox("Pulsame!"),
                BorderLayout.WEST);

/* Adding the Panels to the Frame */
//Frames use BorderLayout by default
cl.add(panelNorth, BorderLayout.NORTH);
cl.add(panelCenter, BorderLayout.CENTER);
cl.add(panelSouth, BorderLayout.SOUTH);
cl.setSize(300,300);
cl.setVisible(true);
}
}
```



Resultado

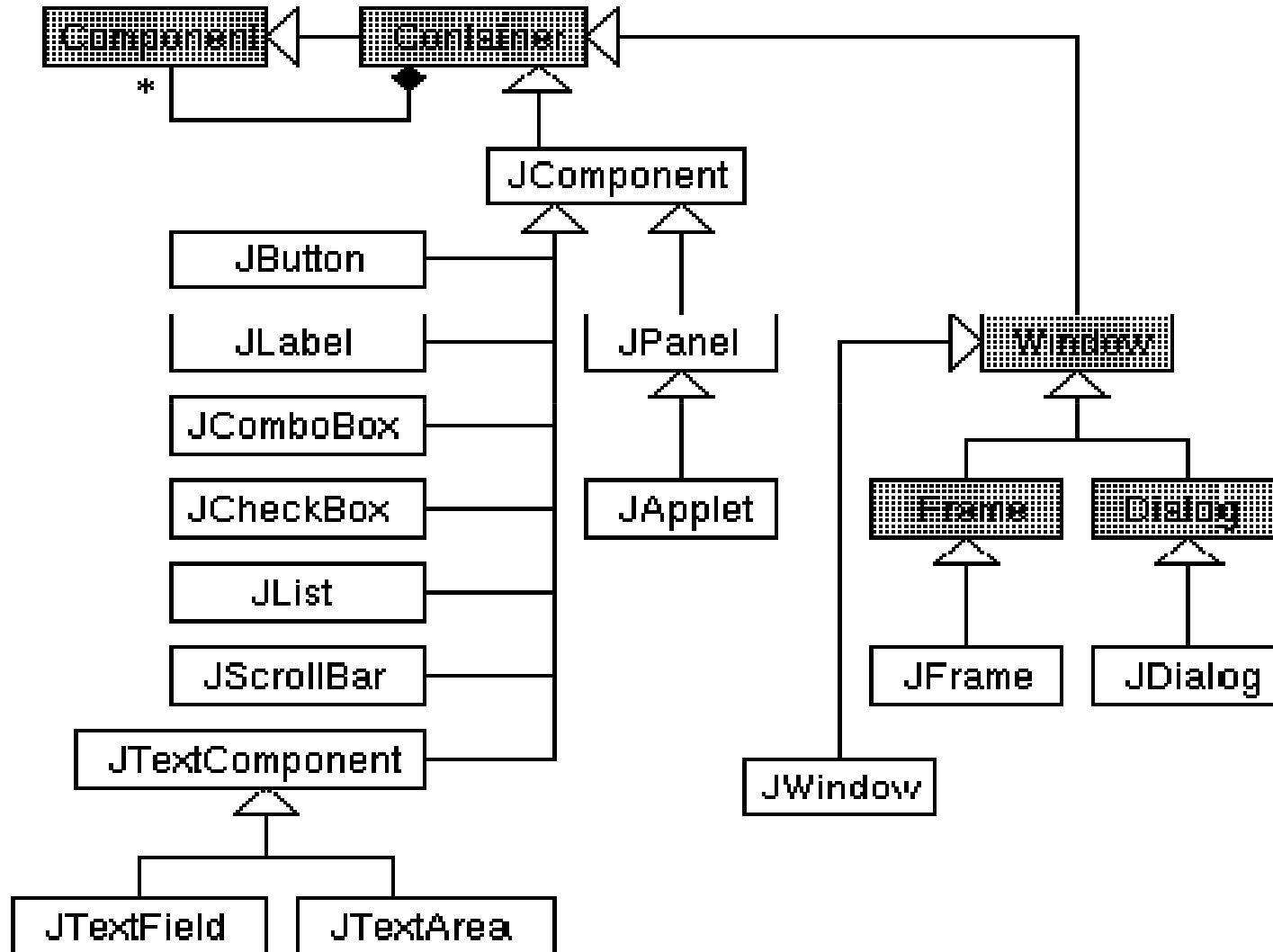


Componentes GUI Swing

- El paquete se encuentra en *javax.swing* y consiste de varios cientos de clases y numerosos subpaquetes
- Escritos completamente en Java y por ello tienen la misma apariencia si se ejecutan en diferentes plataformas
- Nombres de los componentes similares a los de AWT: empiezan con la letra **J** seguido del nombre del componente AWT. Ejemplo: JButton
- Proporciona componentes más interesantes: Selección de color, Panel de opciones, etc.



Jerarquía de Componentes Swing





Componentes GUI Swing básicos

Componente	Descripción
JButton	An implementation of a "push" button.
JCheckBox	An implementation of a check box -- an item that can be selected or deselected, and which displays its state to the user.
JComboBox	A component that combines a button or editable field and a drop-down list.
JComponent	The base class for all Swing components except top-level containers.
JDialog	The main class for creating a dialog window.
JFileChooser	JFileChooser provides a simple mechanism for the user to choose a file.
JFrame	An extended version of <code>java.awt.Frame</code> that adds support for the JFC/Swing component architecture.
JLabel	A display area for a short text string or an image, or both.



Componentes GUI Swing básicos

Componente	Descripción
JList	A component that displays a list of objects and allows the user to select one or more items.
JMenu	An implementation of a menu -- a popup window containing JMenuItem objects that is displayed when the user selects an item on the JMenuBar.
JOptionPane	JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something.
JPanel	JPanel is a generic lightweight container.
JRadioButton	An implementation of a radio button -- an item that can be selected or deselected, and which displays its state to the user.
JScrollBar	An implementation of a scrollbar.
JSlider	A component that lets the user graphically select a value by sliding a knob within a bounded interval.



Componentes GUI Swing básicos

Componente	Descripción
JTextArea	A JTextArea is a multi-line area that displays plain text.
TextField	TextField is a lightweight component that allows the editing of a single line of text.
JTree	A control that displays a set of hierarchical data as an outline.
JWindow	A JWindow is a container that can be displayed anywhere on the user's desktop.
JApplet	An extended version of java.applet.Applet that adds support for the JFC/Swing component architecture.



Swing: ajuste de contenedores de alto nivel

- Los contenedores de alto nivel de Swing son ligeramente incompatibles con los de AWT
 - En términos de adición de componentes al contenedor
- Para añadir un componente al contenedor:
 - Obtener el panel del contenido del contenedor mediante el método *getContentPane*
 - Añadir componentes al panel del contenedor mediante el método *add*



Clase JFrame

- Para mostrar un frame se siguen los pasos:

- Construir un objeto de la clase JFrame

```
JFrame frame = new JFrame();
```

- Asignar el tamaño del frame

```
frame.setSize(300,400);
```

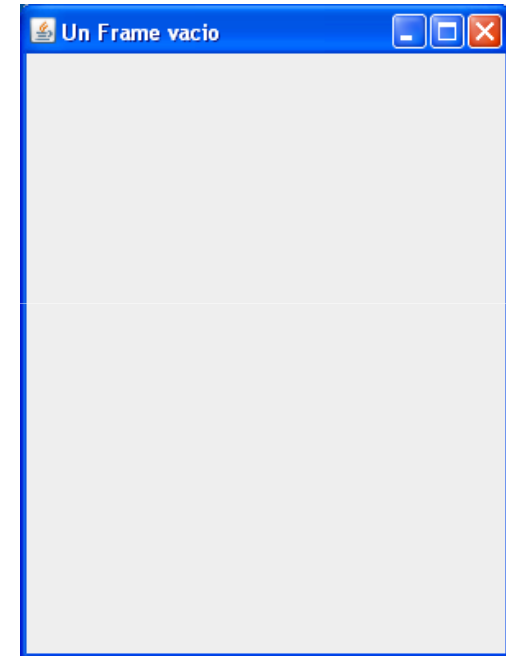
- Asignar el título del frame

```
frame.setTitle("Un Frame vacio");
```

- Asignar el 'default close operation'

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

- Hacerlo visible `frame.setVisible (true);`





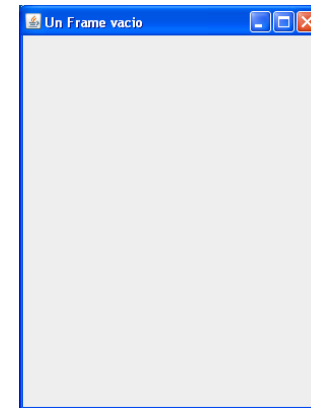
Ejemplo: JFrame vacío

```
import javax.swing.JFrame;

public class EmptyFrameViewer
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();

        frame.setSize(300, 400);
        frame.setTitle("Un Frame vacío");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setVisible(true);
    }
}
```



Resultado



Ejemplo: JFrame con componentes

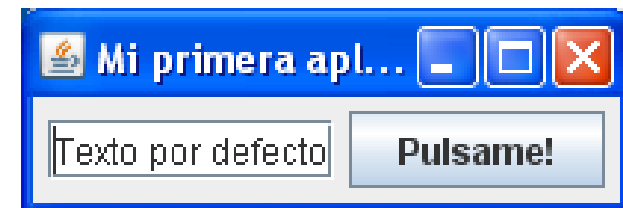
```
import javax.swing.*;
import java.awt.*;

class SwingDemo {
    JFrame frame;
    JPanel panel;
    JTextField textField;
    JButton button;
    Container contentPane;
    public static void main(String args[]) {
        SwingDemo sd = new SwingDemo();
        sd.launchFrame();
    }
}
```



Ejemplo: JFrame con componentes

```
void launchFrame() {  
    /* initialization */  
    frame = new JFrame("Mi primera aplicacion Swing");  
    panel = new JPanel();  
    textField = new JTextField("Texto por defecto");  
    button = new JButton("Pulsame!");  
    contentPane = frame.getContentPane();  
    //add components to panel-FlowLayout by default  
    panel.add(textField);  
    panel.add(button);  
    /* add components to contentPane- BorderLayout */  
    contentPane.add(panel, BorderLayout.CENTER);  
    frame.pack(); //Size of frame based on components  
    frame.setVisible(true);  
}  
}
```



Resultado



Dibujando sobre un componente

- No se puede dibujar directamente en un objeto JFrame
- Para ello, se crea un objeto y se añade al frame
- En la creación del objeto se sobrescribe el método `paintComponent` con las instrucciones de dibujo
- El método `paintComponent` se invoca de manera automática cuando:
 - El componente se muestra la primera vez
 - Cada vez que la ventana se redimensiona o después de ser ocultada



Parámetro Graphics

- El método `paintComponent` recibe un objeto del tipo `Graphics`
 - El objeto `Graphics` almacena el estado gráfico
 - El color, fuente, etc. en curso que se usa para las operaciones de dibujo
 - La clase `Graphics2D` extiende la clase `Graphics`
 - Proporciona métodos más potentes para dibujar objetos 2D
 - Cuando se usa Swing, el parámetro `Graphics` es en realidad de tipo `Graphics2D`, por lo que se requiere realizar un cast a `Graphics2D` para usarlo



Parámetro Graphics

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

public class RectangleComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g;

        // Las instrucciones de dibujo van aquí
    }
}
```

- Las instrucciones de dibujo se colocan dentro del método `paintComponent`, que se invoca siempre que el componente necesita ser repintado.
- Una vez realizado el cast se puede dibujar



Ejemplo: Parámetro Graphics

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import javax.swing.JComponent;

/* Un componente que dibuja dos rectangulos */
public class RectangleComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g; // Recupera Graphics2D
        // Crea un rectangulo, lo dibuja, lo mueve, lo dibuja
        Rectangle box = new Rectangle(5, 10, 20, 30);
        g2.draw(box);
        box.translate(15, 25);
        g2.draw(box);
    }
}
```

Se define un componente para dibujar dos rectangulos y ser utilizado



Ejemplo: Parámetro Graphics

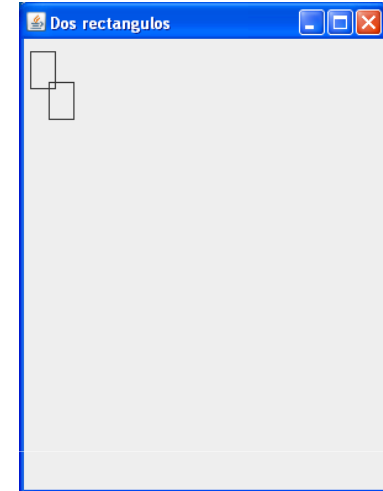
```
import javax.swing.JFrame;

public class RectangleViewer
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();

        frame.setSize(300, 400);
        frame.setTitle("Dos rectangulos");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        RectangleComponent component = new RectangleComponent();
        frame.add(component);

        frame.setVisible(true);
    }
}
```



Resultado

Se crea el nuevo componente y se añade al frame

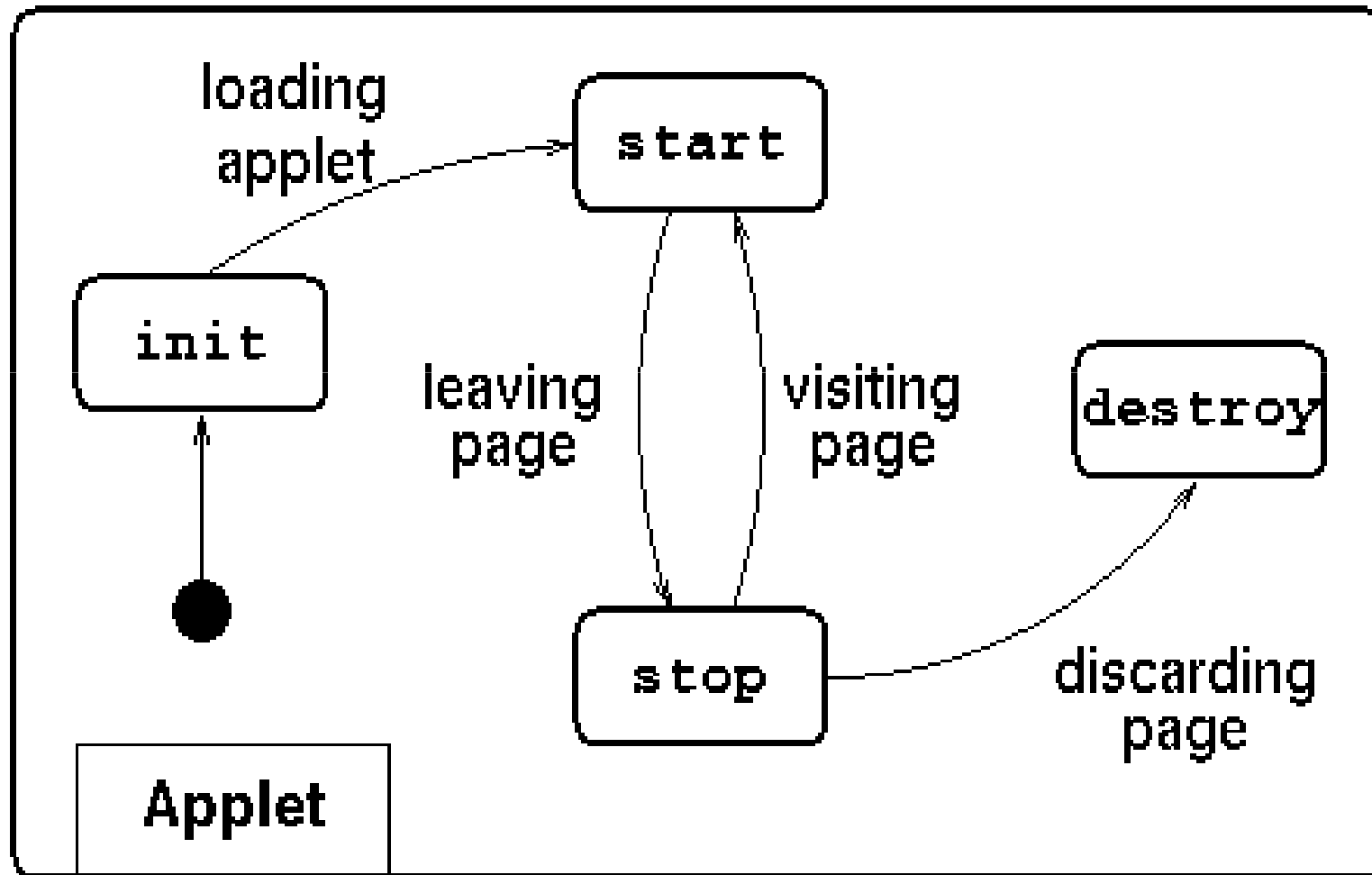


Applets

- Programas que se ejecutan en navegadores Web
- Se incrustan dentro de ficheros HTML usando la etiqueta `<applet>`
- No tienen método `main()`
- Debe extender la clase `Applet` (AWT) del paquete `java.applet.Applet` o `JApplet` (Swing) del paquete `javax.swing.JApplet`
- Hay varios métodos especiales que sirven para propósitos especiales



Ciclo de vida de un Applet





Métodos especiales para Applets relativos al ciclo de vida

- `init()`
 - Para inicializar el applet cada vez que se carga
- `start()`
 - Para iniciar la ejecución del applet, una vez cargado el applet o cuando el usuario vuelve a visitar la página que contiene el applet
- `stop()`
 - Para parar la ejecución del applet, cuando el usuario abandona la página o sale del navegador
- `destroy()`
 - Realiza una limpieza final para preparar la descarga



Métodos para dibujar en Applets

- Para dibujar en el interior de un Applet se sobrescribe el método paint

`public void paint(Graphics g)`

- Método ejecutado automáticamente para dibujar el contenido de applets. No se invoca directamente.
- Un objeto gráfico define el contexto gráfico en el que se dibuja formas y texto
- Sobre la clase Graphics se puede aplicar los métodos de dibujo de primitivas
- Ejemplo: `g.drawLine(10, 20, 150, 45);`
`g.drawRect(50, 20, 100, 40);`



Métodos para dibujar en Applets

- Para dibujar en el interior de un Applet se sobrescribe el método paint

```
public void update(Graphics g)
```

- Es llamado por el sistema para actualizar las ventanas
- Por defecto pinta el área completa de dibujo con el color de fondo
- Asigna el color del fondo
- Invoca el método paint()

```
public void repaint()
```

- Redibuja el contenido del componente



Parámetro Graphics en Applets

- Un Applet es un programa que puede ejecutarse dentro de un navegador de Internet
- Extienden la clase `JApplet`. No requieren de main, ni la separación de clases componentes y visores
- La plantilla de un Applet es:

```
public class MyApplet extends JApplet
{
    public void paint(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g;
        // Instrucciones de dibujo
    }
}
```

Extiende `JApplet`
en lugar de `JFrame`

El código de dibujo
va dentro del método
`paint` en lugar de
`paintComponent`



Ejemplo: Parámetro Graphics en Applets

```
import java.awt.Graphics;  
import java.awt.Graphics2D;  
import java.awt.Rectangle;  
import javax.swing.JApplet;
```

```
/* Applet que dibuja dos rectangulos */  
public class RectangleApplet extends JApplet  
{  
    public void paint(Graphics g)  
    {  
        Graphics2D g2 = (Graphics2D) g;  
  
        Rectangle box = new Rectangle(5, 10, 20, 30);  
        g2.draw(box);  
        box.translate(15, 25);  
  
        g2.draw(box);  
    }  
}
```

No requiere método 'main'.
Invocado en la página html

Etiqueta HTML:

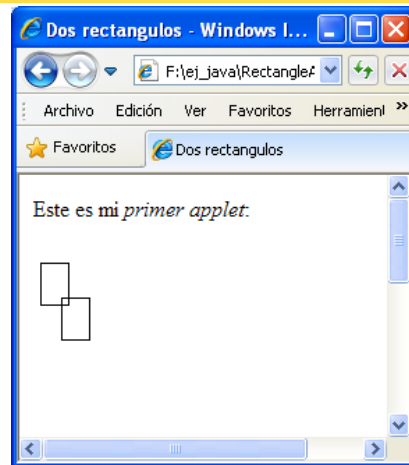
```
<applet code= "RectangleApplet.class"  
width= "300" height= "400" >  
</applet>
```



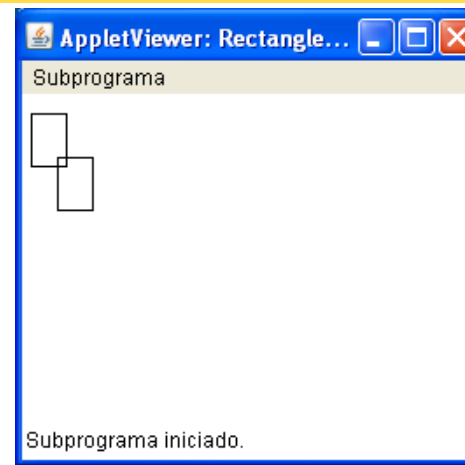
Ejemplo: Visualización de Applets

- Un Applet se puede visualizar mediante dos métodos:
 - Utilizar un navegador de Internet
 - Utilizar la utilidad **appletviewer** que viene con el Java SDK
- En ambos casos se requiere el fichero .html que contiene la etiqueta applet

Visualización con Navegador



Visualización con appletviewer





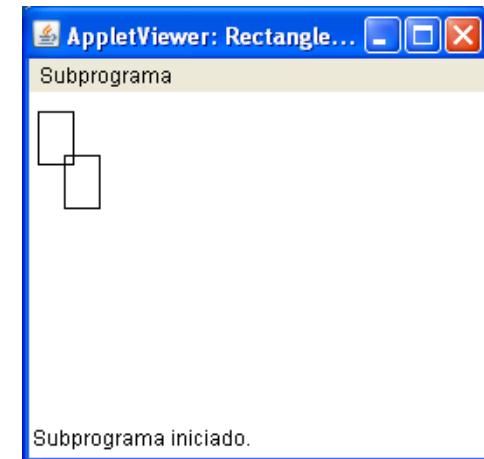
Ejemplo: página HTML mínima para visualizar en Applet

```
<html>
  <head>
    <title>Dos rectangulos</title>
  </head>
  <body>
    <p>Este es mi <i>primer applet</i>:</p>
    <applet code="RectangleApplet.class" width="300"
            height="400">
    </applet>
  </body>
</html>
```

Comando appletviewer

C:\ Símbolo del sistema

```
F:\ej_java>appletviewer RectangleApplet.html
```

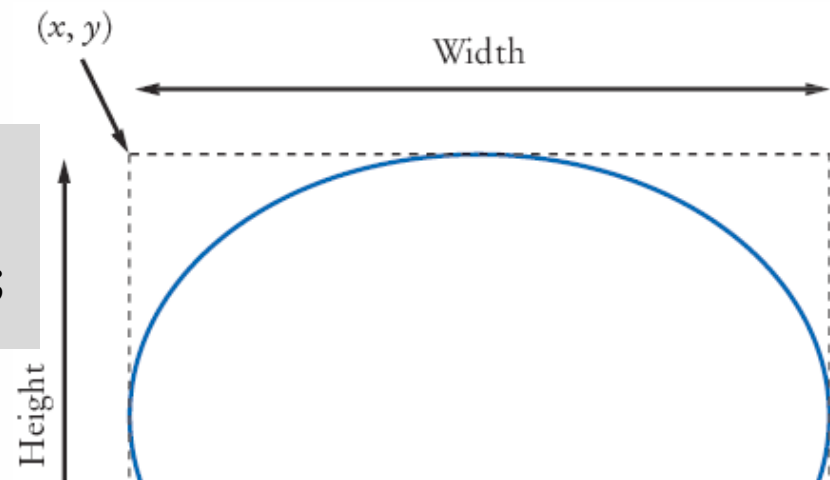




Dibujo de Elipses

- Las elipses se dibujan dentro de una 'caja' de la misma manera que se especifica un rectángulo
 - Proporcionar las coordenadas x e y de la esquina superior izquierda
 - Proporcionar el ancho y alto de la caja
 - Usar la clase `Ellipse2D.Double` para crear la elipse. que es parte de AWT

```
import java.awt.geom.Ellipse2D;  
Ellipse2D.Double ellipse = new  
Ellipse2D.Double(x, y, width, height);  
g2.draw(ellipse);
```





Dibujo de Líneas

- Las líneas se dibujan entre dos puntos
 - Se especifican cuatro valores (x_1, y_1, x_2, y_2)

```
import java.awt.geom.Line2D;  
Line2D.Double segment = new Line2D.Double(x1, y1, x2, y2);
```

- o se especifica dos puntos (p_1, p_2)

```
import java.awt.geom.*;  
Point2D.Double from = new Point2D.Double(x1, y1);  
Point2D.Double to = new Point2D.Double(x2, y2);  
Line2D.Double segment = new Line2D.Double(from, to);
```

- La segunda opción es más orientada a objetos y es más útil si los objetos de los puntos se reusan



Dibujo de Texto

- Usar el método `drawString` de la clase `Graphics2D` para dibujar una cadena en algún lugar de una ventana
 - Especificar el `String`
 - Especificar el punto base (coordenadas `x` e `y`)
 - La línea base es la coordenada `y` del punto base

```
g2.drawString("Message", 50, 100);
```





La clase `java.awt.Font`

- Los Fonts (tipos de letras) son especificados con tres atributos:
 - *font name*: Serif Sans-serif Monospaced Dialog DialogInput TimesRoman Helvetica Courier Dialog
 - *font style*: PLAIN BOLD ITALIC
 - Los estilos se pueden combinar: `Font.BOLD|Font.ITALIC`
 - *font size*: entero positivo
- Un font se puede crear como sigue:
`new Font(name, style, size)`



La clase `java.awt.Color`

- Las instancias de la clase `Color` representan colores.

`new Color(r, g, b)`

donde `r`, `g`, `b` son los valores de los componentes rojo, verde y azul respectivamente. Están en el rango de 0 a 255.
















La clase java.awt.Color

- Todas las formas y strings son dibujadas con un trazo negro y relleno blanco por defecto
- Para cambiar de color, se realiza a través del objeto de tipo Color
 - Java usa el modelo RGB
 - Se puede usar colores predefinidos o crear el propio

```
Color.PINK  
Color.RED
```

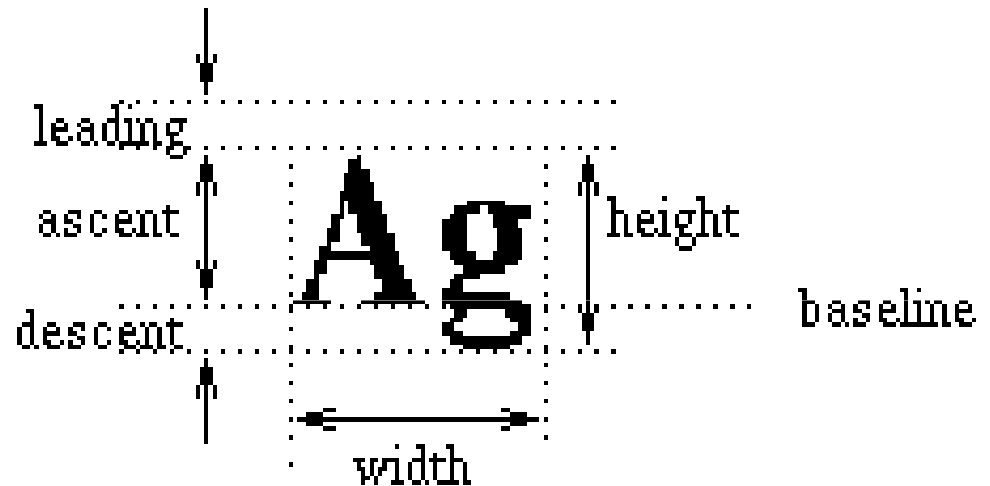
```
Color magenta = new Color(255, 0, 255);
```

Color		RGB Value
Color.BLACK		0, 0, 0
Color.BLUE		0, 0, 255
Color.CYAN		0, 255, 255
Color.GRAY		128, 128, 128
Color.DARKGRAY		64, 64, 64
Color.LIGHTGRAY		192, 192, 192
Color.GREEN		0, 255, 0
Color.MAGENTA		255, 0, 255
Color.ORANGE		255, 200, 0
Color.PINK		255, 175, 175
Color.RED		255, 0, 0
Color.WHITE		255, 255, 255
Color.YELLOW		255, 255, 0



La clase java.awt.Color

- Métodos:
 - getAscent()
 - getDescent()
 - getHeight()
 - getLeading()
 - stringWidth(s)





Uso de Color

- Para dibujar el trazo de una forma en un color diferente:
 - Asignar un color al objeto Graphics2D
 - Invocar el método `draw`

```
g2.setColor(Color.RED);  
g2.draw(circle);    // Dibuja la forma en rojo
```

- Para dibujar una forma rellena con un color diferente:
 - Asignar un color al objeto Graphics2D
 - Invocar el método `fill`

```
g2.setColor(Color.RED);  
g2.fill(circle);    // Rellena la forma de rojo
```



Pasos para dibujar formas

- Determinar las formas que se necesitan dibujar
 - Cuadrados y rectángulos
 - Círculos y elipses
 - Líneas y textos
- Fijar las coordenadas de cada forma
 - Para rectángulos y elipses se requiere la esquina superior izquierda, ancho y alto de la caja encerrante
 - Para líneas se requiere las posiciones x e y de los puntos de inicio y final
 - Para texto se requiere la posición x e y del punto base



Pasos para dibujar formas

- Escribir las instrucciones Java para dibujar las formas

```
Rectangle leftRectangle = new Rectangle(100, 100, 30, 60);
Rectangle rightRectangle = new Rectangle(160, 100, 30, 60);
Line2D.Double topLine = new Line2D.Double(130, 100, 160, 100);
Line2D.Double bottomLine = new Line2D.Double(130, 160, 160, 160);
```

- Incluir las instrucciones de dibujo en un componente

```
public class MyComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g;
        // Instrucciones de dibujo
    }
}
```

- Escribir la clase de visualización

```
public class MyViewer
{
    public static void main(String[] args)
    {
        // Crear el JFrame
        MyComponent component = new MyComponent();
        frame.add(component);
        frame.setVisible(true);
    }
}
```




Ejemplo: Dibujo de formas

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import javax.swing.JComponent;

/* A component that draws an alien face */
public class FaceComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g; // Recover Graphics2D
        Ellipse2D.Double head = new Ellipse2D.Double(5, 10, 100, 150); // Draw the head
        g2.draw(head);
        g2.setColor(Color.GREEN); // Draw the eyes
        Rectangle eye = new Rectangle(25, 70, 15, 15);
        g2.fill(eye);
        eye.translate(50, 0);
        g2.fill(eye);
    }
}
```



Ejemplo: Dibujo de formas

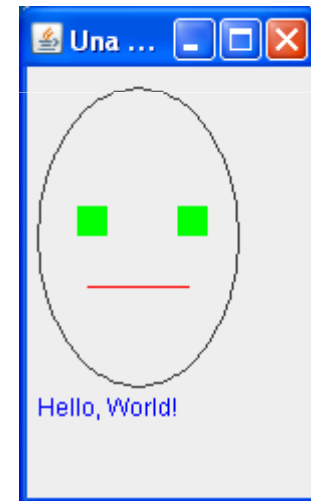
```
Line2D.Double mouth = new Line2D.Double(30, 110, 80, 110); // Draw the mouth
g2.setColor(Color.RED);
g2.draw(mouth);
g2.setColor(Color.BLUE); // Draw the greeting
g2.drawString("Hello, World!", 5, 175);
}
}
```

```
import javax.swing.JFrame;

public class FaceViewer
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        frame.setSize(150, 250);
        frame.setTitle("Una cara de alien");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        FaceComponent component = new FaceComponent();
        frame.add(component);

        frame.setVisible(true);
    }
}
```



Resultado



Ejemplo: Applet BannerDeslizante

```
import java.awt.*;
import javax.swing.JApplet;
public class BannerDeslizante
    extends JApplet implements Runnable {
    protected Thread bannerThread;
    protected String text;
    protected Font font =
        new java.awt.Font("Sans-serif", Font.BOLD, 24);
    protected int x, y;
    protected int delay = 100;
    protected int offset = 1;
    protected Dimension d;
    public void init() {
        // lee parametros "delay" y "text"
        String att = getParameter("delay");
        if (att != null) {
            delay = Integer.parseInt(att);
        }
        att = getParameter("text");
        if (att != null) {
            text = att;
        } else {
            text = "Banner deslizante.";
        }
        d = getSize(); // asigna pos. inicial del texto
        x = d.width;
        y = font.getSize();
    }

    public void paint(Graphics g2) {
        Graphics2D g = (Graphics2D) g2;
        g.setFont(font);
        FontMetrics fm = g.getFontMetrics();
        int length = fm.stringWidth(text);
        x -= offset;
        if (x < -length) x = d.width;
        g.setColor(Color.black);
        g.fillRect(0,0,d.width,d.height);
        g.setColor(Color.green);
        g.drawString(text, x, y);
    }

    public void start() {
        bannerThread = new Thread(this);
        bannerThread.start();
    }

    public void stop() {
        bannerThread = null;
    }

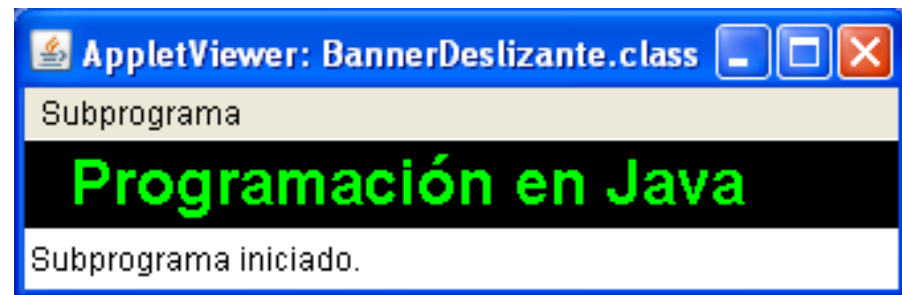
    public void run() {
        while (Thread.currentThread() == bannerThread) {
            try {
                Thread.currentThread().sleep(delay);
            }
            catch (InterruptedException e) {}
            repaint();
        }
    }
}
```



Ejemplo: Applet BannerDeslizante

```
<HTML>
  <HEAD>
    <TITLE> Applet Banner Deslizante</TITLE>
  </HEAD>
<BODY BGCOLOR=black TEXT=white>
  <CENTER>
    <H1> Applet Banner Deslizante</H1>
    <P>
      <APPLET CODE=BannerDeslizante.class
        WIDTH=330 HEIGHT=33>
      <PARAM NAME=text VALUE="Programación en Java">
      <PARAM NAME=delay VALUE=50>
    </APPLET>
  </CENTER>
</BODY>
</HTML>
```

Problema: parpadeo





Técnica para evitar el parpadeo

- El parpadeo es causado por `repaint()`:
 - `repaint()` llama al método `update()`
 - El método `update()` por defecto hace lo siguiente:
 - pinta el área completa con el color del fondo,
 - asigna el color del fondo,
 - llama al método `paint()`.
 - El método `update()` también es llamado por el sistema para actualizar las ventanas
- Solución:
 - priorizar el método `update()`
 - usar una imagen fuera de pantalla



Ejemplo: Applet BannerDeslizante mejorado

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

public class BannerDeslizante2
    extends BannerDeslizante {
    protected Image image;          // imagen fuera de pantalla
    protected Graphics offscreen;  // grafico fuera de pantalla
    public void update(Graphics g) {
        // crea la imagen fuera de pantalla si es la primera vez
        if (image == null) {
            image = createImage(d.width, d.height);
            offscreen = image.getGraphics();
        }
        // dibuja el frame en curso en la imagen fuera de pantalla
        // usando el metodo paint de la superclase
        super.paint(offscreen);
        // copia la imagen fuera de pantalla a la pantalla
        g.drawImage(image, 0, 0, this);
    }
    public void paint(Graphics g) {
        update(g);
    }
}
```



Applet de animación

- Categoría
 - Implementación de comportamiento (behavioral).
- Propósito
 - Actualización de la apariencia de un applet sin intervención del usuario.
- Conocido también como
 - Applet activo.
- Aplicación
 - Animación de procesos dinámicos.



Patrón de programación: Applet de Animación

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

public class AppletAnimacion extends JApplet implements Runnable {
    Thread mainThread = null;
    int delay;
    public void start() {
        if (mainThread == null) {
            mainThread = new Thread(this);
            mainThread.start();
        }
    }
    public void stop() {
        mainThread = null;
    }
    public void run() {
        while (Thread.currentThread() == mainThread) {
            repaint();
            try{
                Thread.currentThread().sleep(delay);
            } catch(InterruptedException){}
        }
    }
    public void paint(java.awt.Graphics g) {
        <pinta el frame en curso>
    }
    <otros metodos y campos>
}
```




Ejemplo: Applet BolaRebotante

```
import java.awt.*;
import javax.swing.JApplet;
public class BolaRebotante
    extends JApplet implements Runnable {
    protected Color color = Color.green;
    protected int radius = 20;
    protected int x, y;
    protected int dx = -2, dy = -4;
    protected Image image;
    protected Graphics offscreen;
    protected Dimension d;
    public void init() {
        String att = getParameter("delay");
        if (att != null) {
            delay = Integer.parseInt(att);
        }
        d = getSize();
        x = d.width * 1 / 3;
        y = d.height - radius;
    }

    public void update(Graphics g) {
        // crea la imagen fuera de pantalla
        // si es la primera vez
        if (image == null) {
            image = createImage(d.width, d.height);
            offscreen = image.getGraphics();
        }

        offscreen.setColor(Color.white);
        offscreen.fillRect(0,0,d.width,d.height);
        if (x < radius || x > d.width - radius) {
            dx = -dx; }
        if (y < radius || y > d.height - radius) {
            dy = -dy; }
        x += dx; y += dy;
        offscreen.setColor(color); // dibuja la bola
        offscreen.fillOval(x - radius, y - radius,
                           radius * 2, radius * 2);
        g.drawImage(image, 0, 0, this);
    }
    public void paint(Graphics g) { update(g); }
    protected Thread bouncingThread;
    protected int delay = 100;
    public void start() {
        bouncingThread = new Thread(this);
        bouncingThread.start();
    }
    public void stop() { bouncingThread = null; }
    public void run() {
        while (Thread.currentThread() == bouncingThread)
        { try {
            Thread.currentThread().sleep(delay);
        } catch (InterruptedException e) {}
        repaint();
        }
    }
}
```



Diseño de componentes genéricos

- Un componente genérico se refiere a componentes de programas, clases o paquetes, que se pueden adaptar y usar en varios contextos sin tener que modificar sus códigos fuentes. También se les conoce como *componentes reusables*.
- Los mecanismos usados son la herencia y la delegación. Las clases abstractas y las interfaces juegan un papel importante.



Factorización

- Una forma de identificar componentes genéricos es identificar segmentos de código recurrentes que son idénticos o casi idénticos entre varias aplicaciones.
- La factorización consiste en:
 - Identificar segmentos de código en un programa que realiza la misma lógica en diferentes sitios.
 - Capturar la lógica en un componente genérico.
 - Reorganizar el programa de tal forma que el segmento de código se reemplace por el componente genérico.



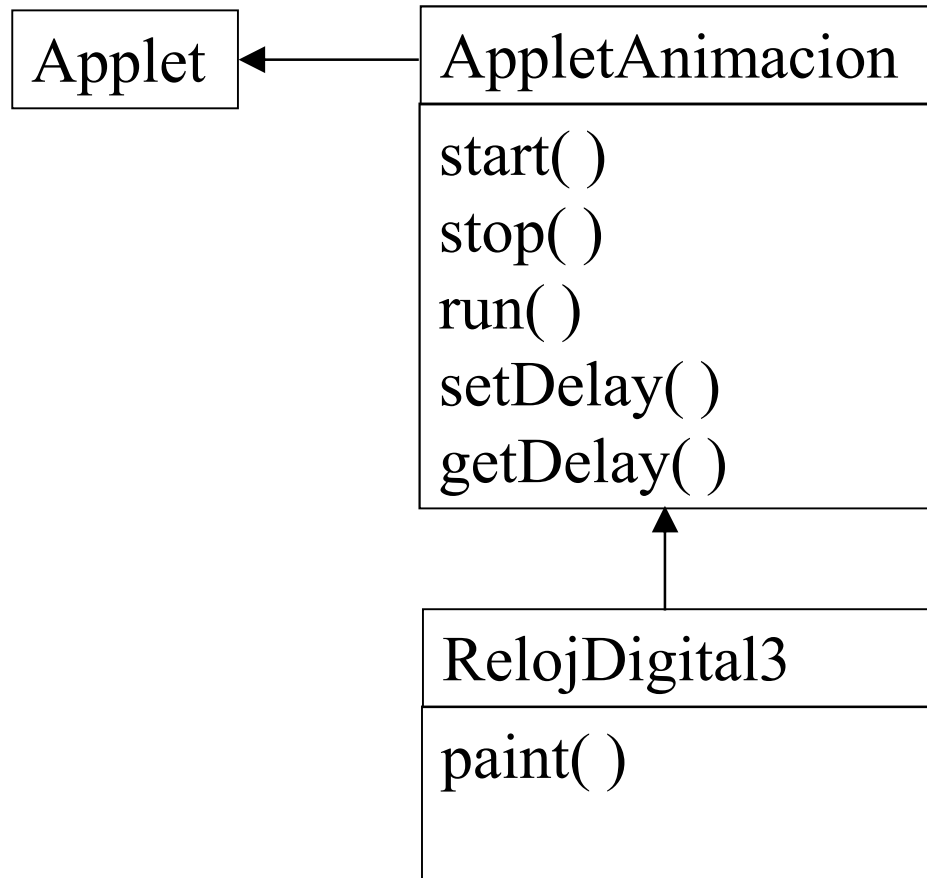
Applet de Animación genérico

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

public class AppletAnimacion extends JApplet implements Runnable {
    public void start() {
        animationThread = new Thread(this);
        animationThread.start();
    }
    public void stop() {
        animationThread = null;
    }
    public void run() {
        while (Thread.currentThread() == animationThread) {
            try {
                Thread.currentThread().sleep(delay);
            } catch (InterruptedException e) {}
            repaint();
        }
    }
    final public void setDelay(int delay) {
        this.delay = delay;
    }
    final public int getDelay() {
        return delay;
    }
    protected Thread animationThread;
    protected int delay = 100;
}
```



Applet genérico de animación con doble buffer





Ejemplo: Applet de animación

```
import java.awt.*;
import java.util.Calendar;
public class RelojDigital3 extends AppletAnimacion {
    public RelojDigital3() {
        setDelay(1000);
    }
    public void init() {
        String param = getParameter("color");
        if ("rojo".equals(param)) { color = Color.red;
        } else if ("azul".equals(param)) { color = Color.blue;
        } else { color = Color.green; }
    }
    public void paint(Graphics g) {
        Calendar calendar = Calendar.getInstance();
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        int minute = calendar.get(Calendar.MINUTE);
        int second = calendar.get(Calendar.SECOND);
        g.setFont(font); g.setColor(color);
        g.drawString(hour + ":" + minute / 10 + minute % 10 +
                    ":" + second / 10 + second % 10, 10, 60);
    }
    protected Font font = new Font("Monospaced", Font.BOLD, 48);
    protected Color color = Color.green;
}
```

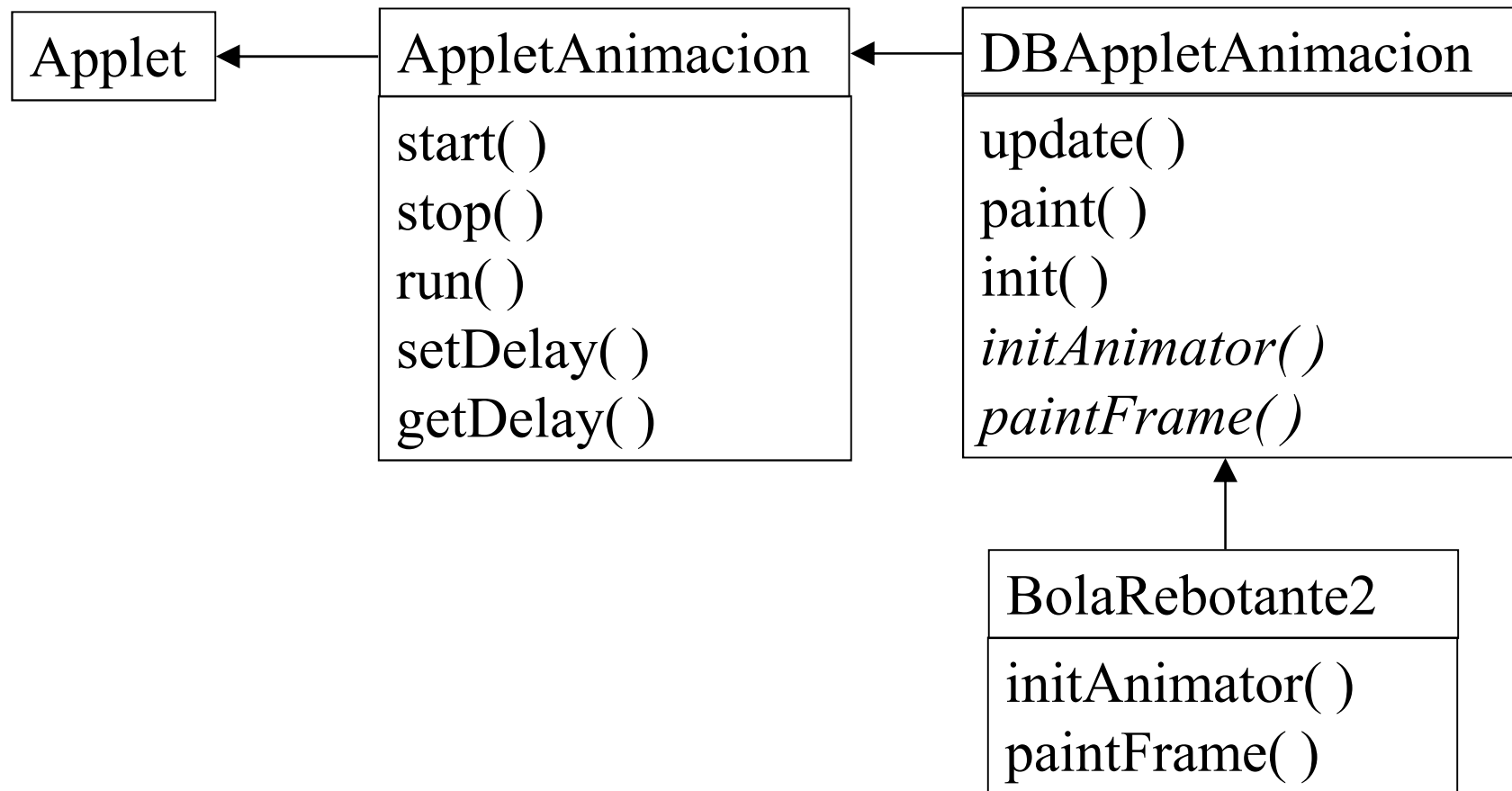


Applet genérico de animación con doble buffer

- Permite varios tamaños del area de visión
- Permite a las subclases decidir el uso de doble buffer
- Factoriza segmentos de código común que contiene partes variables



Applet genérico de animación con doble buffer





Applet genérico de animación con doble buffer

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

public abstract class DBAppletAnimacion extends AppletAnimacion {
    public void update(Graphics g) {
        paintFrame(offscreen);
        g.drawImage(im, 0, 0, this);
    }
    final public void paint(Graphics g) {
        paintFrame(g);
        update(g);
    }
    final public void init() {
        d = getSize();
        im = createImage(d.width, d.height);
        offscreen = im.getGraphics();
        initAnimator();
    }
    protected void initAnimator() {}

    abstract protected void paintFrame(Graphics g);

    protected Dimension d;
    protected Image im;
    protected Graphics offscreen;
}
```



Applet genérico de animación con doble buffer

```
import java.awt.*;

public class BolaRebotante2 extends DBAppletAnimacion {
    protected void initAnimator() {
        String att = getParameter("delay");
        if (att != null)
            setDelay(Integer.parseInt(att));
        x = d.width * 2 / 3 ;
        y = d.height - radius;
    }
    protected void paintFrame(Graphics g) {
        g.setColor(Color.white);
        g.fillRect(0,0,d.width,d.height);
        if (x < radius || x > d.width - radius) {
            dx = -dx;
        }
        if (y < radius || y > d.height - radius) {
            dy = -dy;
        }
        x += dx; y += dy;
        g.setColor(color);
        g.fillOval(x - radius, y - radius, radius * 2, radius * 2);
    }
    protected int x, y;
    protected int dx = -2, dy = -4;
    protected int radius = 20;
    protected Color color = Color.green;
}
```

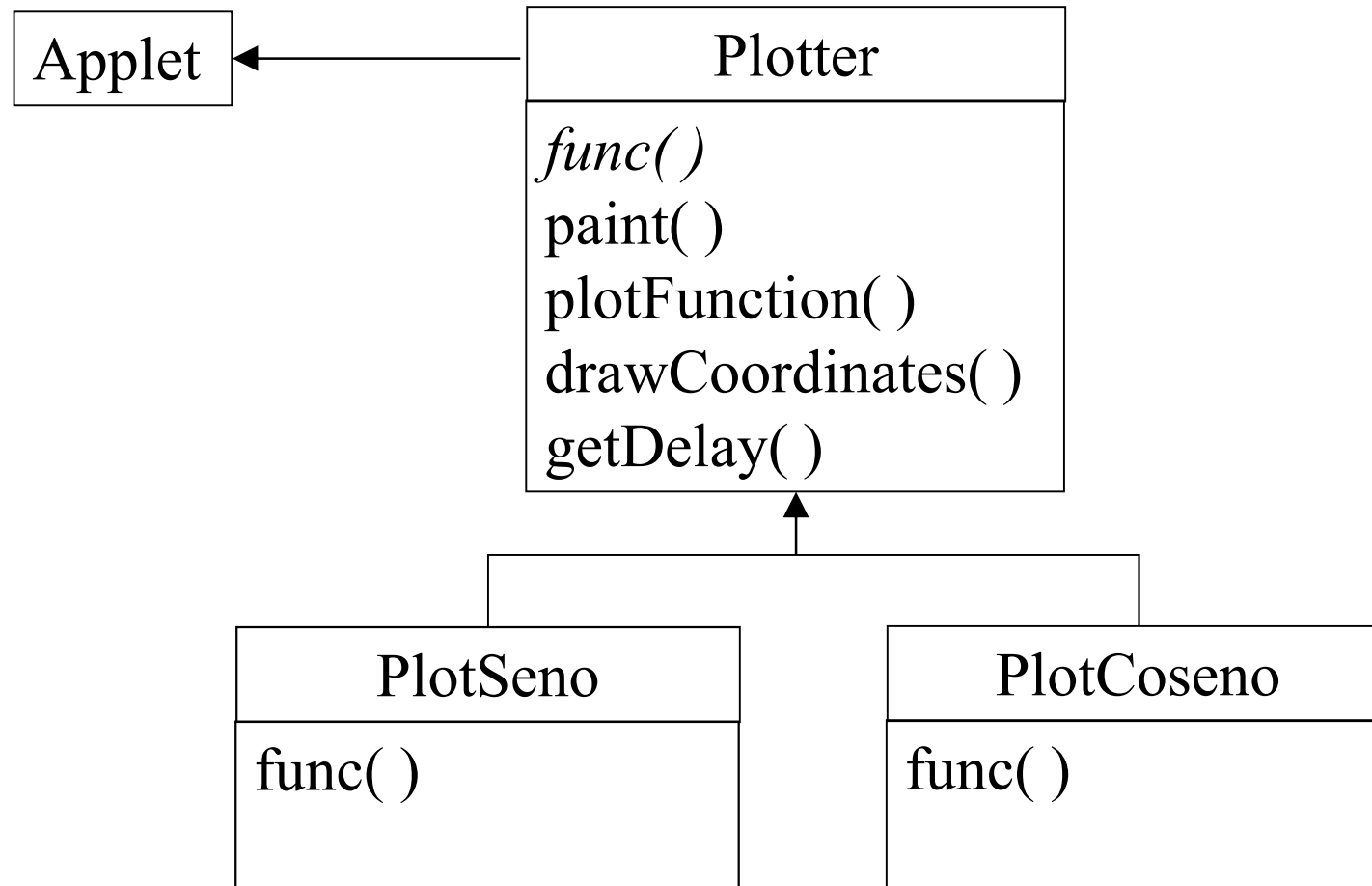


Patrón de diseño: método plantilla

- Categoría
 - Patrón de diseño de comportamiento.
- Propósito
 - Definir el esqueleto de un algoritmo en un método, dejando algunos pasos a las subclasses, con lo que se permite a las subclasses redefinir ciertos pasos del algoritmo.
- Aplicación
 - implementar la parte invariante del algoritmo una vez y dejar a las subclasses el comportamiento que puede variar.
 - factorizar y localizar el comportamiento común entre subclasses para evitar duplicación de código.



Applet plotter genérico: diseño





Applet plotter genérico

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

public abstract class Plotter extends JApplet {
    public abstract double func(double x);
    public void init() {
        d = getSize();
        String att = getParameter("xratio");
        if (att != null)
            xratio = Integer.parseInt(att);
        att = getParameter("yratio");
        if (att != null)
            yratio = Integer.parseInt(att);
        att = getParameter("xorigin");
        if (att != null)
            xorigin = Integer.parseInt(att);
        else
            xorigin = d.width / 2;
        att = getParameter("yorigin");
        if (att != null)
            yorigin = Integer.parseInt(att);
        else
            yorigin = d.height / 2;
    }
    public void paint(Graphics g) {
        drawCoordinates(g);
        plotFunction(g);
    }
}

/** dimension del area de vision */
protected Dimension d;

/** color usado para dibujo */
protected Color color = Color.black;

/** posicion del origen del sistema de
    coordenadas */
protected int xorigin, yorigin;

/** numero de pixels entre 0 y 1 en la
    direccion x e y */
protected int xratio = 100, yratio = 100;

protected void plotFunction(Graphics g) {
    for (int px = 0; px < d.width; px++) {
        try {
            double x = (double)(px - xorigin) /
                (double)xratio;
            double y = func(x);
            int py = yorigin - (int) (y * yratio);
            g.fillOval(px - 1, py - 1, 3, 3);
        } catch (Exception e) {}
    }
}
```

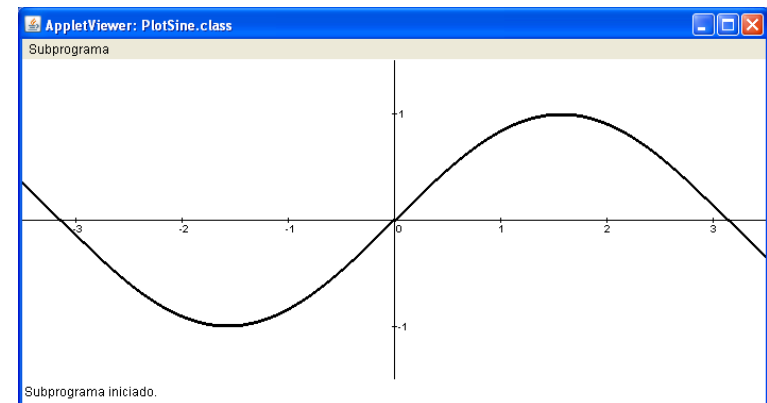


Applet plotter genérico

```
protected void drawCoordinates(Graphics g) {
    g.setColor(Color.white);
    g.fillRect(0, 0, d.width, d.height); g.setColor(color);
    g.drawLine(0, yorigin, d.width, yorigin);
    g.drawLine(xorigin, 0, xorigin, d.height);
    g.setFont(new Font("TimeRoman", Font.PLAIN, 10));
    int px, py; int i = 1; py = yorigin + 12;
    g.drawString("0", xorigin + 2, py);
    for (px = xorigin + xratio; px < d.width; px += xratio) {
        g.drawString(Integer.toString(i++), px - 2, py);
        g.drawLine(px, yorigin - 2, px, yorigin + 2);
    }
    i = -1;
    for (px = xorigin - xratio; px >= 0; px -= xratio) {
        g.drawString(Integer.toString(i--), px - 2, py);
        g.drawLine(px, yorigin - 2, px, yorigin + 2);
    }
    i = 1; px = xorigin + 4;
    for (py = yorigin - yratio; py >= 0; py -= yratio) {
        g.drawString(Integer.toString(i++), px, py + 4);
        g.drawLine(xorigin - 2, py, xorigin + 2, py);
    }
    i = -1;
    for (py = yorigin + yratio; py < d.height; py += yratio) {
        g.drawString(Integer.toString(i--), px, py + 4);
        g.drawLine(xorigin - 2, py, xorigin + 2, py);
    }
}
```

```
public class PlotSine extends Plotter {
    public double func(double x) {
        return Math.sin(x);
    }
}
```

```
<HTML>
<TITLE> Java Demo: Function Plotter
</TITLE>
<BODY>
<APPLET CODE="PlotSine.class" WIDTH=700
HEIGHT=300>
</APPLET>
</BODY>
</HTML>
```





Applet multiplotter

```
import java.awt.*;

public abstract class MultiPlotter extends
    Plotter {
    abstract public void initMultiPlotter();
    public void init() {
        super.init();
        initMultiPlotter();
    }
    final public void addFunction(Function f, Color
        c) {
        if (numOfFunctions < MAX_FUNCTIONS && f !=
            null) {
            functions[numOfFunctions] = f;
            colors[numOfFunctions++] = c;
        }
    }
    protected void plotFunction(Graphics g) {
        for (int i = 0; i < numOfFunctions; i++) {
            if (functions[i] != null) {
                Color c = colors[i];
                if (c != null)
                    g.setColor(c);
                else
                    g.setColor(Color.black);
                for (int px = 0; px < d.width; px++) {
                    try {
                        double x = (double) (px - xorigin) /
                            (double) xratio;
                        double y = functions[i].apply(x);
                        int py = yorigin - (int) (y * yratio);
                        g.fillOval(px - 1, py - 1, 3, 3);
                    } catch (Exception e) {}
                }
            }
        }
    }
    public double func(double x) {
        return 0.0;
    }
    protected static int MAX_FUNCTIONS = 5;
    protected int numOfFunctions = 0;
    protected Function functions[] = new
        Function[MAX_FUNCTIONS];
    protected Color colors[] = new
        Color[MAX_FUNCTIONS];
}
```



Applet multiplotter

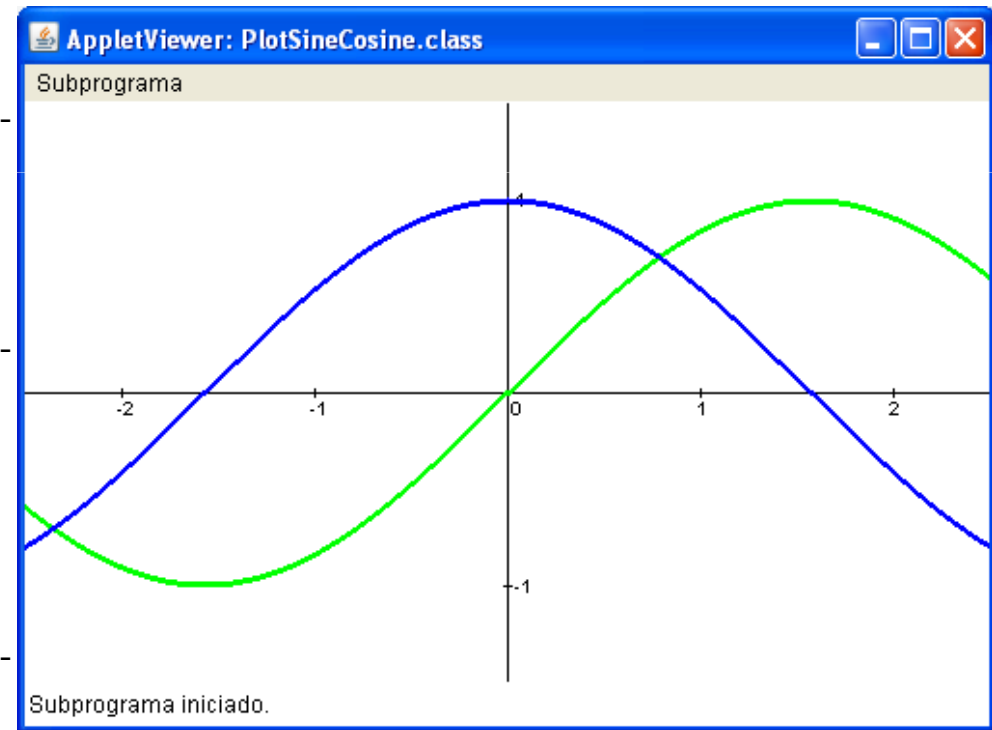
```
import java.awt.Color;

public class PlotSineCosine extends MultiPlotter
{
    public void initMultiPlotter() {
        addFunction(new Sine(), Color.green);
        addFunction(new Cosine(), Color.blue);
    }
}
```

```
public class Cosine implements Function {
    public double apply(double x) {
        return Math.cos(x);
    }
}
```

```
interface Function {
    double apply(double x);
}
```

```
public class Sine implements Function {
    public double apply(double x) {
        return Math.sin(x);
    }
}
```





Modelo de delegación de eventos

- Modelo usado por Java para gestionar la interacción del usuario con los componentes GUI
- Describe cómo responde un programa a la interacción del usuario
- Intervienen tres componentes:
 - La fuente del evento
 - El Listener/Handler del evento
 - El objeto evento



Fuente de eventos, Listener/Handler de eventos

- Fuente de eventos
 - Componente GUI que genera el evento
 - Ejemplo: button push o mouse move
- Listener/Handler de evento
 - Recibe y gestiona eventos: un listener puede escuchar a varias fuentes y diferentes tipos de eventos. Listeners puede ser una clase independientes o una clase interna
 - Contiene la lógica del negocio: la fuente puede se otro listener



El Objeto evento

- Creado cuando ocurre un evento (el usuario interactúa con el componente GUI)
- Contiene toda la información necesaria sobre el evento que ha ocurrido
 - Tipo de evento que ha ocurrido
 - Fuente del evento
- Representado por una clase Event



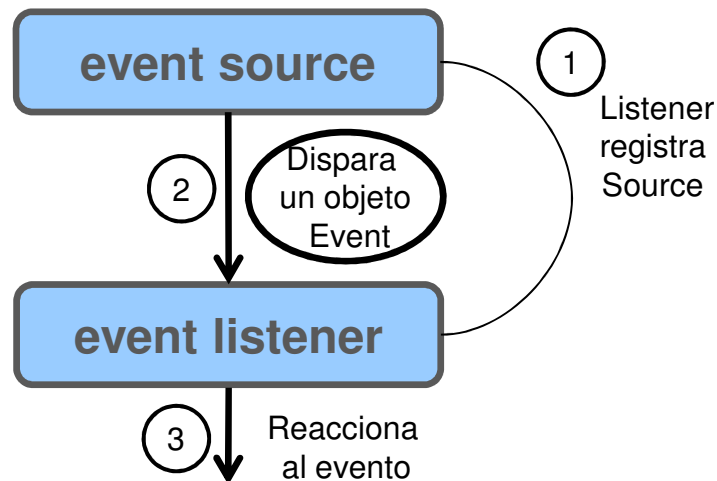
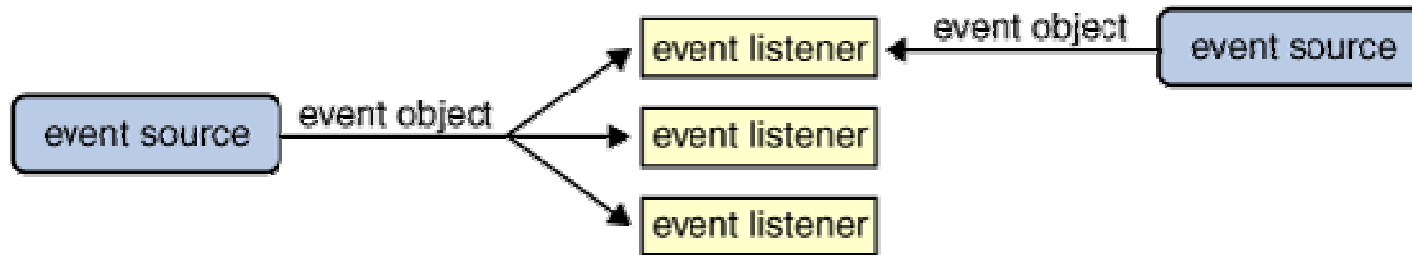
Registro de Listener de eventos a fuente de eventos

- Se debe registrar un listener con una fuente
- Después de registrado, el listener espera hasta que ocurra un evento
- Cuando ocurre un evento
 - Se crea un objeto event por la fuente de eventos
 - El objeto event es disparado por la fuente de eventos a los listeners registrados (método del listener de evento se invoca con un objeto event como parámetro)
- Cuando el listener recibe un objeto event de la fuente
 - Descifra el evento
 - Procesa el evento que ha ocurrido



Control del flujo del modelo de delegación de evento

- Se pueden registrar varios listeners para ser notificados por eventos de un tipo particular de una fuente particular. También, el mismo listener puede escuchar las notificaciones de diferentes objetos





Métodos de fuente de eventos usados para el registro de listeners

- Fuente de eventos registrando un listener

```
void add<Type>Listener(<Type>Listener listenerObj)
```

donde:

<Type> depende del tipo de la fuente de eventos

- Puede ser Key, Mouse, Focus, Component, Action y otros
 - Una fuente de eventos puede registrar varios listeners
- Para desregistrar un listener registrado
- ```
void add<Type>Listener(<Type>Listener listenerObj)
```



## Clases Event

---

- La clase *EventObject* se encuentra en el package *java.util*
- La clase *AWTEvent* es una subclase de *EventObject* definida en el paquete *java.awt*
  - Raíz de todos los eventos basados en AWT
  - Subclases siguen la siguiente convención de nombres:  
<Type>Event



# Clases Event

| Event Class           | Descripción                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------------------|
| <b>ActionEvent</b>    | extends AWTEvent. A semantic event which indicates that a component-defined action occurred.                     |
| <b>ComponentEvent</b> | extends AWTEvent. A low-level event which indicates that a component moved, changed size, or changed visibility. |
| <b>KeyEvent</b>       | An event which indicates that a keystroke occurred in a component.                                               |
| <b>MouseEvent</b>     | An event which indicates that a mouse action occurred in a component.                                            |
| <b>InputEvent</b>     | extends ComponentEvent. The root event class for all component-level input events.                               |
| <b>ItemEvent</b>      | extends AWTEvent. A semantic event which indicates that an item was selected or deselected.                      |
| <b>TextEvent</b>      | extends AWTEvent. A semantic event which indicates that an object's text changed.                                |
| <b>WindowEvent</b>    | extends ComponentEvent. A low-level event that indicates that a window has changed its status.                   |





# Clases Event Listeners

- Clases que implementan la interface <Tipo>Listener

| Event Listener             | Descripción                                                                                       |
|----------------------------|---------------------------------------------------------------------------------------------------|
| <b>ActionListener</b>      | for receiving action events.                                                                      |
| <b>MouseListener</b>       | for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component. |
| <b>MouseMotionListener</b> | for receiving mouse motion events on a component.                                                 |
| <b>WindowListener</b>      | for receiving window events.                                                                      |



# Métodos

| Event Listener        | Descripción                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ActionListener</b> | <code>void <b>actionPerformed</b>(ActionEvent e)</code><br>Invoked when an action occurs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>MouseListener</b>  | <code>void <b>mouseClicked</b>(MouseEvent e)</code><br>Invoked when the mouse button has been clicked (pressed and released) on a component.<br><br><code>void <b>mouseEntered</b>(MouseEvent e)</code><br>Invoked when the mouse enters a component.<br><br><code>void <b>mouseExited</b>(MouseEvent e)</code><br>Invoked when the mouse exits a component.<br><br><code>void <b>mousePressed</b>(MouseEvent e)</code><br>Invoked when a mouse button has been pressed on a component.<br><br><code>void <b>mouseReleased</b>(MouseEvent e)</code><br>Invoked when a mouse button has been released on a component. |



# Métodos

| Event Listener             | Descripción                                                                                                                                                                                                                                                                               |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MouseMotionListener</b> | <code>void <b>mouseDragged</b>(MouseEvent e)</code><br>Invoked when a mouse button is pressed on a component and then dragged.<br><br><code>void <b>mouseMoved</b>(MouseEvent e)</code><br>Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed. |



# Métodos

| Event Listener        | Descripción                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>WindowListener</b> | <p>void <b>windowActivated</b>(WindowEvent e)<br/>Invoked when the Window is set to be the active Window.</p> <p>void <b>windowClosed</b>(WindowEvent e)<br/>Invoked when a window has been closed as the result of calling dispose on the window.</p> <p>void <b>windowClosing</b>(WindowEvent e)<br/>Invoked when the user attempts to close the window from the window's system menu.</p> <p>void <b>windowDeactivated</b>(WindowEvent e)<br/>Invoked when a Window is no longer the active Window.</p> <p>void <b>windowDeiconified</b>(WindowEvent e)<br/>Invoked when a window is changed from a minimized to a normal state.</p> <p>void <b>windowIconified</b>(WindowEvent e)<br/>Invoked when a window is changed from a normal to a minimized state.</p> <p>void <b>windowOpened</b>(WindowEvent e)<br/>Invoked the first time a window is made visible.</p> |



# Pasos para crear aplicaciones GUI con manejo de eventos

---

- Crear una clase GUI
  - Describe y muestra la apariencia de la aplicación
- Crear una clase Event Listener (una clase que implementa la interface listener apropiada)
  - Sobrecribir todos los métodos de la interface listener
  - Describir en cada método se quiere gestionar el evento
  - Se puede dejar vacío la implementación de métodos que no se necesitan
- Registrar el objeto listener con la fuente de evento
  - El objeto es una instancia de la clase listener en paso ant.
  - Usar el método *add<Tipo>Listener* de la fuente de evento



# Ejemplo MouseEvents&CloseWindow

---

```
import java.awt.*;
import java.awt.event.*;
public class MouseEventsDemo extends Frame implements
 MouseListener, MouseMotionListener, WindowListener {
 TextField tf;
 // Method implem. of MouseListener and MouseMotionListener interf.
 public MouseEventsDemo(String title){
 super(title);
 tf = new TextField(60);
 addMouseListener(this);
 addWindowListener(this);
 }
 public void launchFrame() {
 add(tf, BorderLayout.SOUTH);
 setSize(300,300); setVisible(true);
 }
 public void mouseClicked(MouseEvent me) {
 String msg = "Mouse clicked.";
 tf.setText(msg);
 }
}
```

---



# Ejemplo MouseEvents&CloseWindow

---

```
public void mouseEntered(MouseEvent me) {
 String msg = "Mouse entered component."; tf.setText(msg);
}
public void mouseExited(MouseEvent me) {
 String msg = "Mouse exited component."; tf.setText(msg);
}
public void mousePressed(MouseEvent me) {
 String msg = "Mouse pressed."; tf.setText(msg);
}
public void mouseReleased(MouseEvent me) {
 String msg = "Mouse released."; tf.setText(msg);
}
public void mouseDragged(MouseEvent me) {
 String msg = "Mouse dragged at " + me.getX() + "," + me.getY();
 tf.setText(msg);
}
public void mouseMoved(MouseEvent me) {
 String msg = "Mouse moved at " + me.getX() + "," + me.getY();
 tf.setText(msg);
}
```

---



# Ejemplo MouseEvents&CloseWindow

---

```
// Methods implementations of WindowListener Interface
public void windowActivated(WindowEvent e) { }
public void windowClosed(WindowEvent e) { }
public void windowClosing(WindowEvent e) {
 setVisible(false);
 System.exit(0);
}
public void windowDeactivated(WindowEvent e) { }
public void windowDeiconified(WindowEvent e) { }
public void windowIconified(WindowEvent e) {
}
public void windowOpened(WindowEvent e) {
}
// Main method
public static void main(String args[]) {
 MouseEventsDemo med =
 new MouseEventsDemo("Mouse Events Demo");
 med.launchFrame();
}
}
```

---





# Clases Adapter

---

- Porqué usar clases Adapter?
  - Implementar todos los métodos de una interface toma tiempo
  - Interés en implementar solo algunos métodos de la interface
- Clases Adapter
  - Empotradas en Java
  - Implementa todos los métodos de cada interface listener con más de un método
  - La implementación de los métodos están vacías



## Ejemplo clase Adapter

---

```
import java.awt.*;
import java.awt.event.*;
class CloseFrameDemo extends Frame {
 Label label;
 CFlisterener w = new CFlisterener(this);
 CloseFrameDemo(String title) {
 super(title);
 label = new Label("Close the frame.");
 this.addWindowListener(w);
 }
 void launchFrame() {
 setSize(300,300);
 setVisible(true);
 }
 public static void main(String args[]) {
 CloseFrameDemo cf =
 new CloseFrameDemo("Close Window Example");
 cf.launchFrame();
 }
}
```

---



## Ejemplo clase Adapter

---

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

class CListener extends WindowAdapter {
 CloseFrameDemo ref;
 CListener(CloseFrameDemo ref){
 this.ref = ref;
 }

 public void windowClosing(WindowEvent e) {
 ref.dispose();
 System.exit(1);
 }
}
```



# Clases Inner

---

- Clases declaradas dentro de otras
- Se usan para simplificar los programas, especialmente en gestión de eventos



## Ejemplo clase inner para cerrar ventana

---

```
import java.awt.*;
import java.awt.event.*;
class CloseFrame extends Frame {
 Label label;
 CloseFrame (String title) {
 super(title); label = new Label("Close the frame.");
 this.addWindowListener(new CFListener());
 }
 void launchFrame() {
 setSize(300,300); setVisible(true);
 }
 class CFListener extends WindowAdapter {
 public void windowClosing(WindowEvent e) {
 dispose(); System.exit(1);
 }
 }
 public static void main(String args[]) {
 CloseFrame cf = new CloseFrame("Close Window Example");
 cf.launchFrame();
 } }
```

---



## Clases Inner anónimas

---

- Clases inner sin nombre
- Se usan para simplificar aun más los programas, especialmente en gestión de eventos



## Ejemplo clase inner anónima

---

```
import java.awt.*;
import java.awt.event.*;
class CloseFrameA extends Frame {
 Label label;
 CloseFrameA (String title) {
 super(title); label = new Label("Close the frame.");
 this.addWindowListener(new WindowAdapter() {
 public void windowClosing(WindowEvent e) {
 dispose();
 System.exit(1);
 }
 });
 }
 void launchFrame() {
 setSize(300,300); setVisible(true);
 }
 public static void main(String args[]) {
 CloseFrameA cf = new CloseFrameA("Close Window Example");
 cf.launchFrame();
 } }
```

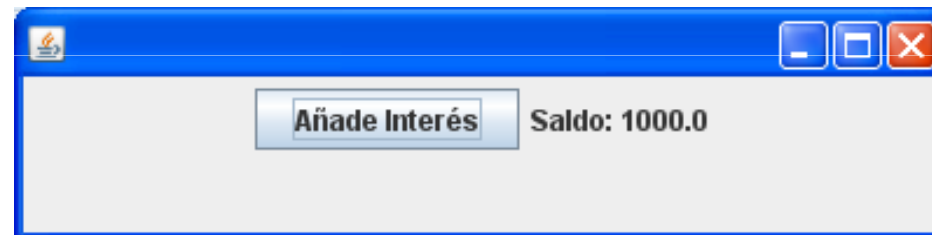
---



## Construyendo aplicaciones con botones

---

- Se requiere una aplicación para que al pulsar un botón se incremente el interés a una cuenta y lo muestre







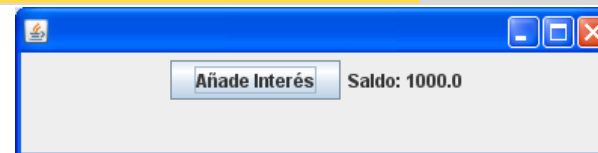
## Componentes JButton, JLabel y JPanel

- Los componentes siguientes heredan de la clase `Container` y se puede colocar en la ventana:
  - JButton: muestra un botón que se puede pulsar
  - JLabel: muestra un texto en una posición de la ventana
  - JPanel: se usa para agrupar múltiples componentes

```
JFrame frame = new JFrame();
JButton button = new JButton("Añade Interés");
JLabel label = new JLabel("Saldo: " + account.getBalance());
JPanel panel = new JPanel();
panel.add(button);
panel.add(label);
frame.add(panel);
```

Button y Label se añaden al Panel

Panel se añade al Frame





## Añadiendo el Event Listener

---

- El programa usa una clase separada para gestionar el botón "Añade Interés"
  - AddInterestListener
    - requiere acceso al objeto account
    - cambia el texto del objeto label

```
class AddInterestListener implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
 double interest = account.getBalance() * INTEREST_RATE / 100;
 account.deposit(interest);
 label.setText("Saldo: " + account.getBalance());
 }
}
```



## Acceso del Listener a las variables locales

---

- Declarar las variables necesarias dentro de main como final
  - account
  - label

```
public static void main(String[] args)
{
 . . .
 JButton button = new JButton("Añadir Interés");
 final BankAccount account = new BankAccount(INITIAL_BALANCE);
 final JLabel label = new JLabel("Saldo: " + account.getBalance());
 . . .
}
```



## Listener como clase interna

---

- Instalar la clase Listener como una clase interna en main
  - Instanciarlo y ‘registrarlo’ con el botón

```
public static void main(String[] args) {
 . . .
 class AddInterestListener implements ActionListener {
 public void actionPerformed(ActionEvent event) {
 double interest = account.getBalance() * INTEREST_RATE / 100;
 account.deposit(interest);
 label.setText("Saldo: " + account.getBalance());
 }
 }
 ActionListener listener = new AddInterestListener();
 button.addActionListener(listener);
}
```



## Código del ejemplo completo

---

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

/** Programa que muestar el incremento de una inversion. */
public class VisorInversion {
 private static final int FRAME_WIDTH = 400;
 private static final int FRAME_HEIGHT = 100;
 private static final double INTEREST_RATE = 10;
 private static final double INITIAL_BALANCE = 1000;

 public static void main(String[] args) {
 JFrame frame = new JFrame();
 JButton button = new JButton("Añade Interés");
 final BankAccount account = new BankAccount(INITIAL_BALANCE);
 final JLabel label = new JLabel("Saldo: " + account.getBalance());
 JPanel panel = new JPanel();
 panel.add(button);
 panel.add(label);
 frame.add(panel);
 }
}
```



# Código del ejemplo completo

---

```
class AddInterestListener implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
 double interest = account.getBalance() * INTEREST_RATE / 100;
 account.deposit(interest);
 label.setText("Saldo: " + account.getBalance());
 }
}

ActionListener listener = new AddInterestListener();
button.addActionListener(listener);

frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}
```



## Código del ejemplo completo

---

```
/** cuenta bancaria con saldo que cambia por ingresos y retiros. */
public class BankAccount {
 private double balance;
 public BankAccount() {
 balance = 0;
 }
 public BankAccount(double initialBalance) {
 balance = initialBalance;
 }
 public void deposit(double amount) {
 double newBalance = balance + amount;
 balance = newBalance;
 }
 public void withdraw(double amount) {
 double newBalance = balance - amount;
 balance = newBalance;
 }
 public double getBalance() {
 return balance;
 }
}
```

---



## Uso de eventos Timer para animación

---

- `javax.swing` proporciona una clase `Timer` que se puede usar para programar animaciones simples
  - Genera una serie de eventos en intervalos de tiempo
  - Especifica la frecuencia de los eventos y un objeto de la clase que implementa la interfaz `ActionListener`

```
class MyListener implements ActionListener {
 public void actionPerformed(ActionEvent event) {
 // accion del Listener (ejecutado en intervalos del timer)
 }
}
MyListener listener = new MyListener();
Timer t = new Timer(interval, listener);
t.start();
```

El timer se iniciará después de invocar el método `start`





# Ejemplo: RectangleComponent1

---

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import javax.swing.JComponent;

/** Este componente muestra un rectangulo rectangulo que se puede mover */
public class RectangleComponent1 extends JComponent {
 private static final int BOX_X = 100;
 private static final int BOX_Y = 100;
 private static final int BOX_WIDTH = 20;
 private static final int BOX_HEIGHT = 30;
 private Rectangle box;
 public RectangleComponent1() {
 // rectangulo que dibuja el metodo paintComponent
 box = new Rectangle(BOX_X, BOX_Y, BOX_WIDTH, BOX_HEIGHT);
 }
 public void paintComponent(Graphics g) {
 Graphics2D g2 = (Graphics2D) g;
 g2.draw(box);
 }
}
```



# Ejemplo: RectangleComponent1

---

```
/**
 * Desplaza el rectangulo una cierta cantidad.
 * @param x cantidad a mover en la direccion x
 * @param y cantidad a mover en la direccion y
 */
public void moveBy(int dx, int dy)
{
 box.translate(dx, dy);
 repaint();
}
}
```



# Ejemplo: RectangleMover

---

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.Timer;

/** Programa que mueve un rectangulo */
public class RectangleMover {
 private static final int FRAME_WIDTH = 300;
 private static final int FRAME_HEIGHT = 400;

 public static void main(String[] args) {
 JFrame frame = new JFrame();
 frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
 frame.setTitle("An animated rectangle");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 final RectangleComponent1 component = new RectangleComponent1();
 frame.add(component);
 frame.setVisible(true);
 }
}
```



## Ejemplo: RectangleMover

---

```
class TimerListener implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
 component.moveBy(1, 1);
 }
}

ActionListener listener = new TimerListener();

final int DELAY = 100; // Milisegundos entre ticks del timer
Timer t = new Timer(DELAY, listener);
t.start();
}
}
```



## Eventos Mouse

- Si se desea gestionar objetos con el ratón es necesario procesar eventos mouse, que son más complejos que los de los botones o timer
  - Un mouse listener debe implementar todos los métodos de la interface `MouseListener`

| método                     | Evento generado                                   |
|----------------------------|---------------------------------------------------|
| <code>mousePressed</code>  | Un botón del ratón se ha pulsado en un componente |
| <code>mouseReleased</code> | Un botón del ratón se ha soltado en un componente |
| <code>mouseClicked</code>  | El ratón se ha pulsado en un componente           |
| <code>mouseEntered</code>  | El ratón (cursor) entra en un componente          |
| <code>mouseExited</code>   | El ratón (cursor) sale de un componente           |



## Métodos MouseListener

---

- Los métodos `mousePressed` y `mouseReleased` se invocan cuando se pulsa o suelta el botón (ratón)
- Si se pulsa y suelta el botón del ratón sin moverlo se invoca también el método `mouseClicked`
- Los métodos `mouseEntered` y `mouseExited` se pueden usar para señalar componentes

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



## Implementación de MouseListener

---

- Otra diferencia es que se usa el método `addMouseListener` en lugar de `addActionListener` para añadirlo a un componente

```
public class MyMouseListener implements MouseListener
{
 // Implementa los cinco metodos
}
MouseListener listener = new MyMouseListener();
component.addMouseListener(listener);
```

- Como ejemplo si se pulsa en un componente que contiene un rectángulo y si se mantiene pulsado, el rectángulo se mueve a la posición del ratón



## Ejemplo: RectangleComponent2

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import javax.swing.JComponent;
public class RectangleComponent2 extends JComponent {
 private static final int BOX_X = 100;
 private static final int BOX_Y = 100;
 private static final int BOX_WIDTH = 20;
 private static final int BOX_HEIGHT = 30;
 private Rectangle box;
 public RectangleComponent2() {
 box = new Rectangle(BOX_X, BOX_Y, BOX_WIDTH, BOX_HEIGHT);
 }
 public void paintComponent(Graphics g) {
 Graphics2D g2 = (Graphics2D) g; g2.draw(box);
 }
 public void moveTo(int x, int y) {
 box.setLocation(x, y);
 repaint();
 }
}
```

### □ El método moveTo:

- se invocará por el handler del evento mousePressed
- `repaint` indica al componente redibujarse.





# Implementación de MouseListener

- `mousePressed` es el único método que se implementa dejando el resto vacío

```
class MousePressListener implements MouseListener
{
 public void mousePressed(MouseEvent event)
 {
 int x = event.getX();
 int y = event.getY();
 component.moveTo(x, y);
 }
 // métodos vacíos
 public void mouseReleased(MouseEvent event) {}
 public void mouseClicked(MouseEvent event) {}
 public void mouseEntered(MouseEvent event) {}
 public void mouseExited(MouseEvent event) {}
}
```

Algunos métodos útiles del objeto `event` obtiene la posición del ratón

- La referencia a `component` se obtiene con una clase interna con esa variable referencia



# Ejemplo: RectangleComponentViewer1

---

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;

/** Programa que muestra un RectangleComponent. */
public class RectangleComponentViewer1 {
 private static final int FRAME_WIDTH = 300;
 private static final int FRAME_HEIGHT = 400;
 public static void main(String[] args) {
 final RectangleComponent2 component = new RectangleComponent2();
 class MousePressListener implements MouseListener {
 public void mousePressed(MouseEvent event) {
 int x = event.getX();
 int y = event.getY();
 component.moveTo(x, y);
 }
 public void mouseReleased(MouseEvent event) {}
 public void mouseClicked(MouseEvent event) {}
 public void mouseEntered(MouseEvent event) {}
 public void mouseExited(MouseEvent event) {}
 }
 }
}
```



# Ejemplo: RectangleComponentViewer1

---

```
MouseListener listener = new MousePressListener();
component.addMouseListener(listener);

JFrame frame = new JFrame();
frame.add(component);

frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
 }
}
```



## Uso de herencia para personalizar Frames

---

- Para facilitar la comprensión de los programas con frames complejos se usa herencia
- Para ello se diseña una subclase de JFrame
  - Se almacena los componentes como variables de instancia
  - Se inicializan en el constructor de la subclase

```
public class InvestmentFrame extends JFrame
{
 private JButton button;
 private JLabel label;
 private JPanel panel;
 private BankAccount cuenta;
```



## Personalización de constructores Frames

- Inicializar los componentes en el constructor de la subclase apoyándose en métodos para los detalles

```
public InvestmentFrame()
{
 account = new BankAccount(INITIAL_BALANCE);
 label = new JLabel("Saldo: " + account.getBalance());
 // Uso de métodos de ayuda
 createButton();
 createPanel();
 setSize(FRAME_WIDTH, FRAME_HEIGHT);
}

private void createButton()
{
 ActionListener listener = new AddInterestListener();
 button.addActionListener(listener);
 button = new JButton("Add Interest");
}

private void createPanel()
{
 panel = new JPanel();
 panel.add(button);
 panel.add(label);
 add(panel);
}
```



## Uso del Frame personalizado

---

- Se crea un objeto del nuevo Frame

```
public class InvestmentFrameViewer
{
 public static void main(String[] args)
 {
 JFrame frame = new InvestmentFrame();
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setVisible(true);
 }
}
```



## Uso del Frame personalizado

---

- Se crea un objeto del nuevo Frame

```
public class InvestmentFrameViewer
{
 public static void main(String[] args)
 {
 JFrame frame = new InvestmentFrame();
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setVisible(true);
 }
}
```



## Proceso de entrada de texto

---

- La mayoría de los programas gráficos toma la entrada de texto a través de campos de texto (text fields)
- La clase `JTextField` proporciona un text field
  - Cuando se construye un text field se especifica el ancho en número de caracteres esperado. Si el usuario supera el número el texto se desplaza a la izquierda

```
final int FIELD_WIDTH = 10;
final JTextField rateField = new JTextField(FIELD_WIDTH);
```

A screenshot of a Java text field containing the text "5.0". The text is centered within the field, and the field has a thin border.

5.0





## Adición de una etiqueta y un botón

---

- Una etiqueta (Label) ayuda al usuario a conocer el contenido de la interfaz
  - Normalmente se colocan a la izquierda de un text field

```
JLabel rateLabel = new JLabel("Interest Rate: ");
```



Interest Rate: 5.0

- Un botón con un método `actionPerformed` se puede usar para leer el texto del text field con el método `getText` que devuelve un String

```
double rate = Double.parseDouble(rateField.getText());
double interest = account.getBalance() * rate / 100;
account.deposit(interest);
resultLabel.setText("balance: " + account.getBalance());
```



## Areas de texto

---

- Para crear áreas de texto multilínea se usa el objeto `JTextArea`
  - Se asigna el tamaño en filas y columnas

```
final int ROWS = 10;
final int COLUMNS = 30;
JTextArea textArea = new JTextArea(ROWS, COLUMNS);
```

- Para asignar un texto en un text field o text area se usa el método `setText`

```
textArea.setText("Account Balance");
```

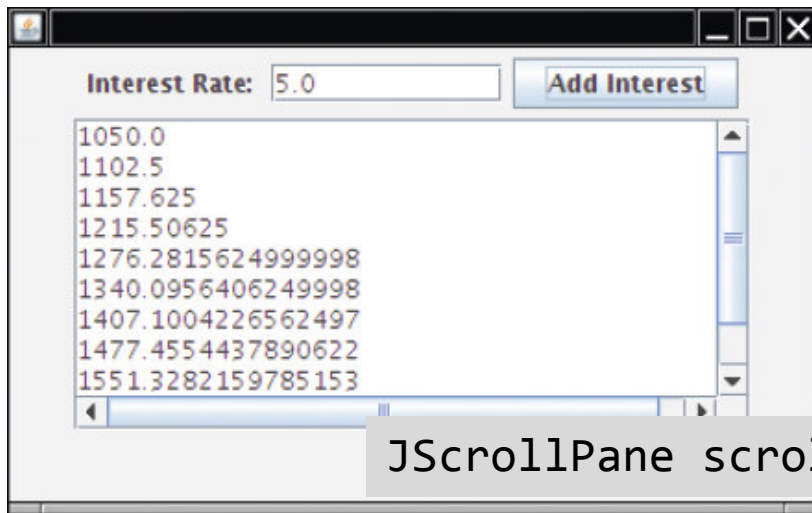
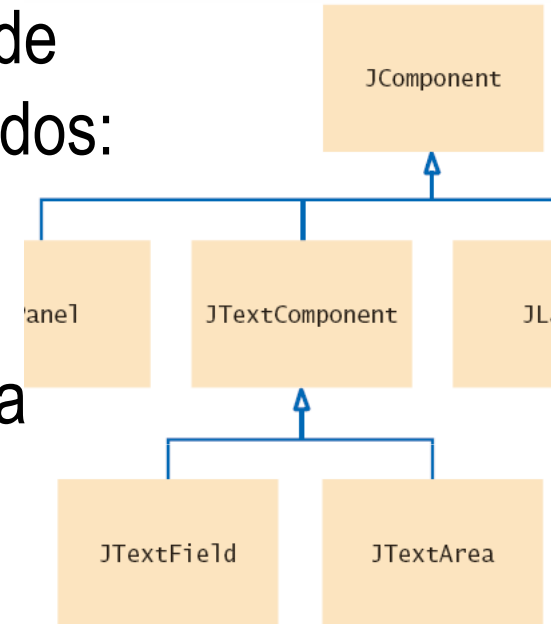
- Para añadir texto se usa el método `append` y para controlar la entrada de texto `setEditable`

```
textArea.append(account.getBalance() + "\n");
textArea.setEditable(false);
```



# JTextField y JTextArea

- JTextArea y JTextField heredan de JTextComponent que tiene como métodos:
  - setText
  - setEditable
  - El método append está declarado en la clase JTextArea



- Para añadir scrollbars se usa JScrollPane

```
JScrollPane scrollPane = new JScrollPane(textArea);
```



## Panel de contenido JFrame

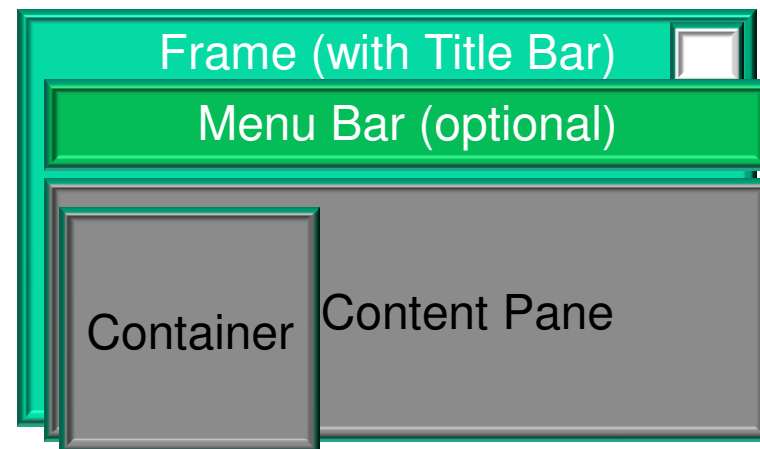
- Un `JFrame` tiene un panel de contenido que es un `Container` donde se puede colocar componentes
  - Usar el método `getContentPane()` para obtener su referencia

```
Container contentPane = getContentPane();
```

- Añadir componentes o un contenedor al panel de contenido y agregar componentes al contenedor

- **Frame**

- `MenuBar`
- `Content Pane`
  - `Components`
  - `Container`
    - » `Components`





## Componentes de selección

---

- En las interfaces de usuario es común usar componentes para hacer diferentes tipos de selección
  - Radio Buttons
    - Para pequeños conjuntos de opciones mutuamente exclusivos

Size

Small  Medium  Large

- Check Boxes
  - Para selecciones binarias

Style

Italic  Bold

- Combo Boxes
  - Para un gran conjunto de opciones

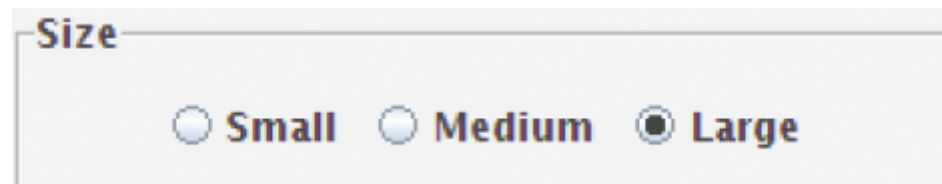
Serif



## Paneles de Radio Button

- Se puede añadir un borde con etiqueta a un panel para visible un grupo de radio buttons
  - Hay varios estilos de bordes disponibles (ver documentación de Swing)

```
JPanel panel = new JPanel();
panel.add(smallButton);
panel.add(mediumButton);
panel.add(largeButton);
panel.setBorder(new TitledBorder(new EtchedBorder(), "Size"));
```

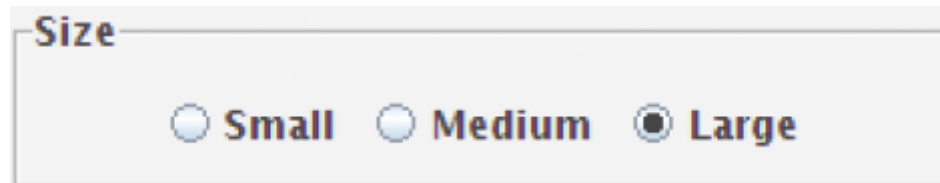




## Grupos de Radio Button

- Añadir Radio Buttons en un ButtonGroup para que sólo un botón del grupo sea seleccionado
  - Crear primero JRadioButtons y después agregarlos al ButtonGroup

```
JRadioButton smallButton = new JRadioButton("Small");
JRadioButton mediumButton = new JRadioButton("Medium");
JRadioButton largeButton = new JRadioButton("Large");
ButtonGroup group = new ButtonGroup();
group.add(smallButton);
group.add(mediumButton);
group.add(largeButton);
```

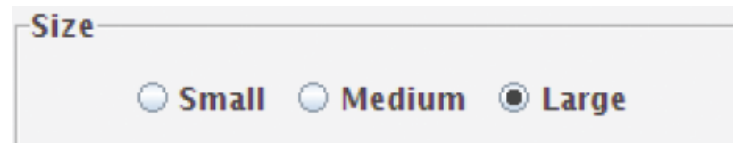




## Selección de Radio Buttons

- Es usual configurar un botón como seleccionado (por defecto) cuando se usan radio buttons
  - Usar el método `setSelected` en el botón antes de hacer visible el frame

```
JRadioButton largeButton = new JRadioButton("Large");
largeButton.setSelected(true);
```



- Invocar el método `isSelected` de un botón para saber si está seleccionado o no

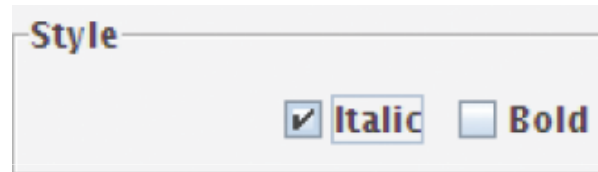
```
if (largeButton.isSelected())
{ size = LARGE_SIZE; }
```





## Check Boxes

- Un Check Box es un componente con dos estados: checked y unchecked
  - Usual en selecciones que no son mutuamente exclusivos



- Para asignar un Check Box se usa JCheckBox

```
JCheckBox italicCheckBox = new JCheckBox("Italic");
```

- Invocar el método `isSelected` del checkbox para saber si está seleccionado o no

```
if (italicCheckBox.isSelected())
{ style = style + Font.ITALIC }
```



## Combo Boxes

- Un Combo Box es una combinación de una lista y un campo de texto
  - Usar en caso de gran cantidad de opciones
  - Pueden ser:
    - Cerrados (muestran una selección)
    - Abiertos (muestran varias opciones)
  - Pueden ser editables
    - Escribir una selección en un línea en blanco
  - Cuando se pulsa en la flecha de la derecha del campo de texto del Combo Box, una lista se despliega y se puede seleccionar uno de los items





# Adición y selección de items en Combo Box

- Se puede añadir items de texto a un combo box que mostrará en la lista:

```
JComboBox facenameCombo = new JComboBox();
facenameCombo.addItem("Serif");
facenameCombo.addItem("SansSerif");
. . .
```

- Usar el método `getSelectedItem` para obtener el item seleccionado (como un Objeto)

```
String selectedString = (String)
facenameCombo.getSelectedItem();
```



## Gestión de eventos

---

- Los Radio Buttons, Check Boxes y Comobo Boxes generan eventos de acción como los botones
  - Agregar un `ActionListener` a cada componente
    - Crear clases internas en cada componente que implemente `ActionListener`
    - Proporcionar control preciso por componente
  - O usar un `ActionListener` para capturar todos los eventos
    - Crear objeto que impleneta `ActionListener` en el constructor del `Frame`
    - Puede solicitar un conjunto de items relacionados usando los métodos `isSelected` y `getSelectedItem`



# Ejemplo: FontViewer

---

```
import javax.swing.JFrame;

/**
 * Programa que permite al usuario visualizar cambios de font
 */
public class FontViewer
{
 public static void main(String[] args)
 {
 JFrame frame = new FontViewerFrame();
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setTitle("FontViewer");
 frame.setVisible(true);
 }
}
```



# Ejemplo: FontViewerFrame

```
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
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.JRadioButton;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

/** Frame para cambiar el font del texto */
public class FontViewerFrame extends JFrame {
 private static final int FRAME_WIDTH = 300;
 private static final int FRAME_HEIGHT = 400;

 private JLabel sampleField;
 private JCheckBox italicCheckBox;
 private JCheckBox boldCheckBox;
 private JRadioButton smallButton;
 private JRadioButton mediumButton;
 private JRadioButton largeButton;
 private JComboBox facenameCombo;
 private ActionListener listener;

 /**
 * Construye texto de ejemplo
 */
 public FontViewerFrame()
 {
 // Construct text sample
 sampleField = new JLabel("Gran Java");
 add(sampleField, BorderLayout.CENTER);

 // listener compartido entre todos los componentes
 class ChoiceListener implements ActionListener
 {
 public void actionPerformed(ActionEvent event)
 {
 setSampleFont();
 }
 }

 listener = new ChoiceListener();

 createControlPanel();
 setSampleFont();
 setSize(FRAME_WIDTH, FRAME_HEIGHT);
 }
}
```



## Ejemplo: FontViewerFrame

```
/** Crea panel de ctrl para cambiar el font */
public void createControlPanel() {
 JPanel facenamePanel = createComboBox();
 JPanel sizeGroupPanel = createCheckBoxes();
 JPanel styleGroupPanel = createRadioButtons();
 // colocar componentes en el panel
 JPanel controlPanel = new JPanel();
 controlPanel.setLayout(new GridLayout(3, 1));
 controlPanel.add(facenamePanel);
 controlPanel.add(sizeGroupPanel);
 controlPanel.add(styleGroupPanel);
 // Agrega panel al contenido
 add(controlPanel, BorderLayout.SOUTH);
}

/** Crea combo box con opciones de estilos font
 @return el panel conteniendo el combo box
 */
public JPanel createComboBox() {
 facenameCombo = new JComboBox();
 facenameCombo.addItem("Serif");
 facenameCombo.addItem("SansSerif");
 facenameCombo.addItem("Monospaced");
 facenameCombo.setEditable(true);
 facenameCombo.addActionListener(listener);

 JPanel panel = new JPanel();
 panel.add(facenameCombo);
 return panel;
}
```

```
/** @return el panel conteniendo el check box */
public JPanel createCheckBoxes() {
 italicCheckBox = new JCheckBox("Italic");
 italicCheckBox.addActionListener(listener);
 boldCheckBox = new JCheckBox("Bold");
 boldCheckBox.addActionListener(listener);
 JPanel panel = new JPanel();
 panel.add(italicCheckBox);
 panel.add(boldCheckBox);
 panel.setBorder(new TitledBorder(new EtchedBorder(),
 "Style")); return panel;
}

/** @return el panel con los radio buttons */
public JPanel createRadioButtons() {
 smallButton = new JRadioButton("Small");
 smallButton.addActionListener(listener);
 mediumButton = new JRadioButton("Medium");
 mediumButton.addActionListener(listener);
 largeButton = new JRadioButton("Large");
 largeButton.addActionListener(listener);
 largeButton.setSelected(true);
 // Add radio buttons to button group
 ButtonGroup group = new ButtonGroup();
 group.add(smallButton); group.add(mediumButton);
 group.add(largeButton); JPanel panel = new JPanel();
 panel.add(smallButton); panel.add(mediumButton);
 panel.add(largeButton);
 panel.setBorder(new TitledBorder(new EtchedBorder(),
 "Size")); return panel;
}
```



# Ejemplo: FontViewerFrame

---

```
/** Obtiene nombre, estilo y tamaño del font
 y asigna el font al texto de ejemplo
 */
public void setSampleFont() {
 // Obtiene nombre del font
 String facename = (String) facenameCombo.getSelectedItem();
 // Obtiene estilo del font
 int style = 0;
 if (italicCheckBox.isSelected())
 {
 style = style + Font.ITALIC;
 }
 if (boldCheckBox.isSelected())
 {
 style = style + Font.BOLD;
 }
 // Obtiene tamaño del font
 int size = 0;
 final int SMALL_SIZE = 24;
 final int MEDIUM_SIZE = 36;
 final int LARGE_SIZE = 48;
 if (smallButton.isSelected()) { size = SMALL_SIZE; }
 else if (mediumButton.isSelected()) { size = MEDIUM_SIZE; }
 else if (largeButton.isSelected()) { size = LARGE_SIZE; }
 // Asigna font al text field
 sampleField.setFont(new Font(facename, style, size));
 sampleField.repaint();
}
}
```





# Pasos para diseñar una Interface de Usuario

- Hacer un esquema de la disposición de componentes
  - Dibujar todos los botones, etiquetas, campos de texto y bordes en papel

Size

|                                        |                                               |
|----------------------------------------|-----------------------------------------------|
| <input checked="" type="radio"/> Small | <input checked="" type="checkbox"/> Pepperoni |
| <input type="radio"/> Medium           | <input checked="" type="checkbox"/> Anchovies |
| <input type="radio"/> Large            |                                               |

Your Price:

- Hallar agrupaciones de componentes adyacentes con la misma disposición
  - Empezar viendo los componentes adyacentes que están dispuestos de arriba abajo o izquierda derecha

Size

|                                        |                                               |
|----------------------------------------|-----------------------------------------------|
| <input checked="" type="radio"/> Small | <input checked="" type="checkbox"/> Pepperoni |
| <input type="radio"/> Medium           | <input checked="" type="checkbox"/> Anchovies |
| <input type="radio"/> Large            |                                               |

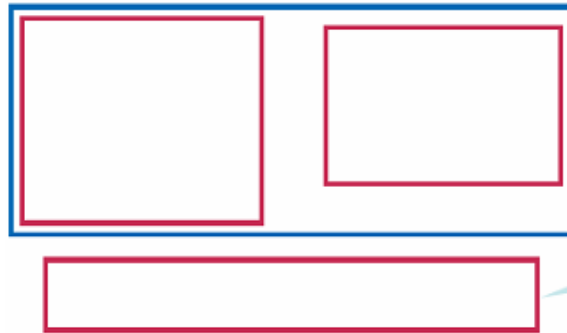
Your Price:



# Pasos para diseñar una Interface de Usuario

---

- Identificar la disposición de cada grupo
  - Para componentes horizontales, usar flow layout
  - Para componentes verticales, usar grid layout con una columna
- Agrupar los grupos
  - Considerar cada grupo como un globo y agrupar los globos en grupos más grandes, de la misma manera que se agrupan los componentes en el paso anterior





# Pasos para diseñar una Interface de Usuario

- Escribir el código para generar la interfaz

```
JPanel radioButtonPanel = new JPanel();
radioButtonPanel.setLayout(new GridLayout(3, 1));
radioButton.setBorder(new TitledBorder(new EtchedBorder(), "Size"));
radioButtonPanel.add(smallButton);
radioButtonPanel.add(mediumButton);
radioButtonPanel.add(largeButton);
```

```
JPanel checkBoxPanel = new JPanel();
checkBoxPanel.setLayout(new GridLayout(2, 1));
checkBoxPanel.add(pepperoniButton());
checkBoxPanel.add(anchoviesButton());
```

```
JPanel pricePanel = new JPanel(); // Usa FlowLayout por defecto
pricePanel.add(new JLabel("Your Price:"));
pricePanel.add(priceTextField);
JPanel centerPanel = new JPanel(); // Use FlowLayout
centerPanel.add(radioButtonPanel);
centerPanel.add(checkBoxPanel); // Frame usa BorderLayout por defecto
add(centerPanel, BorderLayout.CENTER);
add(pricePanel, BorderLayout.SOUTH);
```

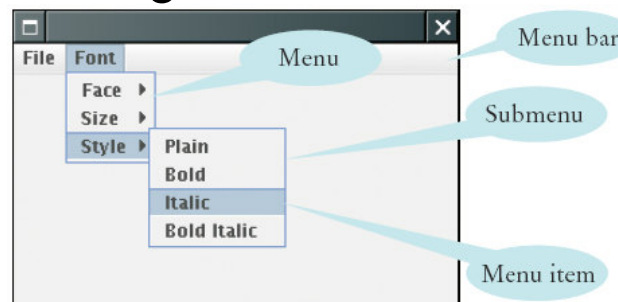


# Menús

- Un frame puede contener una barra de menú
  - El contenedor para los items del menú de alto nivel se llama barra de menú
  - Un menú contiene submenús y items
  - Para añadir items y submenus al menú se usa el método `add`

```
JMenuItem fileExitItem = new JMenuItem("Exit");
fileMenu.add(fileExitItem);
```

- Los items del menú generan eventos cuando se seleccionan





## Eventos de items de menú

---

- Los items de menú generan eventos de acción cuando se seleccionan
  - Añadir action listeners sólo a los items del menú
  - Añadir un listener a cada item del menú

```
fileExitItem.addActionListener(listener);
```



## Otros componentes

---

- Swing dispone muchos componentes que pueden ser útiles en el diseño y creación de interfaces de usuario
- Para usarlos es conveniente aprender a navegar por la documentación de la API de Swing, identificar el componente apropiado y responder:
  - Cómo se construye?
  - Cómo obtener la notificación cuando se actúa sobre él
  - Cómo obtener el valor del componente



## Multimedia con Java

---

- Java dispone de una serie de APIs que permiten desarrollar aplicaciones multimedia:
  - Java 2D
  - Java 3D
  - Java Advanced Imaging (JAI)
  - Java Binding for OpenGL (JOGL)
  - Java Image I/O
  - Java Media Framework (JMF)



## Trabajando con imágenes

---

- Tareas comunes cuando se trabaja con imágenes
    - Cargar una imagen externa en formato GIF, PNG, JPEG
    - Crear una imagen Java 2D y mostrarla
    - Dibujar el contenido de una imagen Java 2D en una superficie de dibujo
    - Guardar el contenido de una imagen Java 2D en un fichero externo GIF, PNG o JPEG
  - Hay dos clases que se usan para imágenes:
    - `java.awt.Image` es la superclase que representa las imágenes como arrays de píxeles
    - `java.awt.image.BufferedImage` que extiende la clase `Image` para operar con los datos de la imagen
-





## Lectura de imágenes

---

- Para cargar una imagen desde un fichero se usa el código:

```
BufferedImage img = null;
try {
 img = ImageIO.read(new File("imagen.jpg"));
} catch (IOException e) {
}
```

- Si el código es un applet :

```
try {
 URL url = new URL(getCodeBase(), "imagen.jpg");
 img = ImageIO.read(url);
} catch (IOException e) {
}
```



# Visualización de imágenes

---

- Para mostrar una imagen se usa el código:

```
public void paint(Graphics g) {
 g.drawImage(img, 0, 0, null);
}
```



## Ejemplo: LoadImageApp

---

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

/** Muestra como leer una imagen desde un fichero */
public class LoadImageApp extends Component {
 BufferedImage img;

 public void paint(Graphics g) {
 g.drawImage(img, 0, 0, null);
 }

 public LoadImageApp() {
 try {
 img = ImageIO.read(new File("strawberry.jpg"));
 } catch (IOException e) {
 }
 }
}
```



## Ejemplo: LoadImageApp

---

```
public Dimension getPreferredSize() {
 if (img == null) {
 return new Dimension(100,100);
 } else {
 return new Dimension(img.getWidth(null), img.getHeight(null));
 }
}

public static void main(String[] args) {
 JFrame f = new JFrame("Ejemplo de carga de imagen");
 f.addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent e) {
 System.exit(0);
 }
 });

 f.add(new LoadImageApp());
 f.pack();
 f.setVisible(true);
}
}
```



## Sonido en Java

---

- La Java Sound API permite controlar y realizar efectos sobre la entrada y salida de sonidos
- Soporta datos de audio digital (WAV, AIFF, AU) y MIDI. Paquetes
  - `javax.sound.sampled`. Contiene interfaces para capturar, mezclar, y reproducir audio digital
  - `javax.sound.midi`. Interfaces para la síntesis, secuenciación y transporte de eventos MIDI
- Para reproducir se usa un *clip*, que es un tipo de línea en la que se puede cargar los datos de audio previo a la reproducción



# Ejemplo: AudioClipTest

---

```
import java.io.*;
import java.net.URL;
import javax.sound.sampled.*;
import javax.swing.*;
// Reproduce un sonido usando Clip, el proceso requiere estar activo.
public class AudioClipTest extends JFrame {
 // Constructor
 public AudioClipTest() {
 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 this.setTitle("Test Sound Clip"); this.setSize(300, 200);
 this.setVisible(true);
 try {
 // Abre un audio input stream.
 URL url = this.getClass().getClassLoader().getResource("prueba.wav");
 AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
 // Get a sound clip resource.
 Clip clip = AudioSystem.getClip();
 // Abre audio clip y lo carga desde el audio input stream.
 clip.open(audioIn); clip.start();
 } catch (UnsupportedAudioFileException e) { e.printStackTrace();
 } catch (IOException e) { e.printStackTrace();
 } catch (LineUnavailableException e) { e.printStackTrace();
 }
 }
 public static void main(String[] args) {
 new AudioClipTest();
 }
}
```