



UNIVERSIDAD DE ALMERÍA

# **Grado en Ingeniería Informática**

## **Metodología de la programación**

### **2013**



# **Interfaces gráficas de usuario.**

**Eventos: Listeners, Registrations, y Handling**

**Clases internas**

**Alternativas para definir clases Listener**

**Menus, ToolBars y Dialogs**

**Applets**

# Eventos y fuente de eventos

En programación dirigida por eventos, el código es ejecutado cuando tiene lugar un evento:

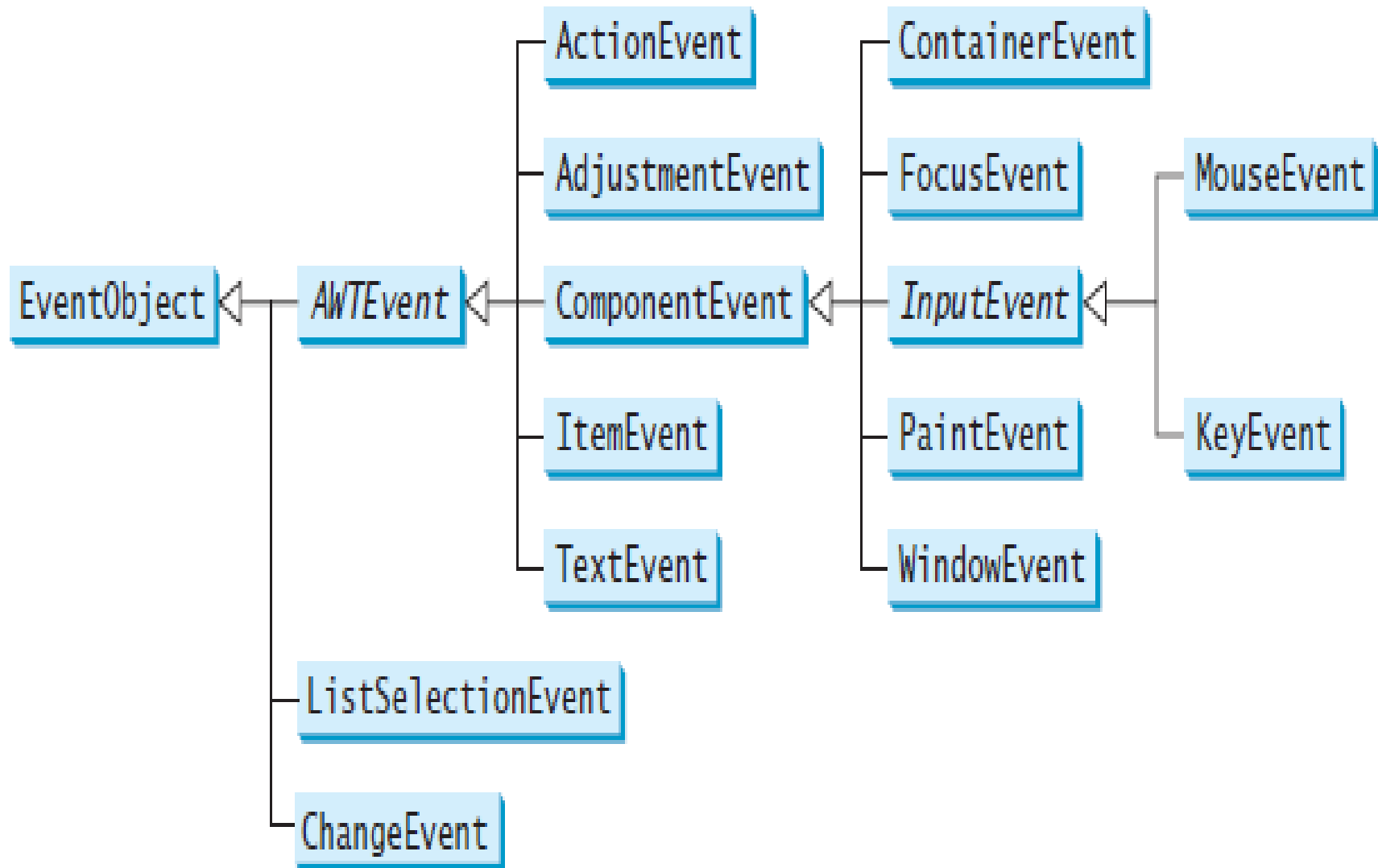
pulsar un botón, se mueve el ratón, etc....

Un ***evento*** se puede ver como una señal al programa de que algo ha sucedido.

El programa tiene que elegir si responde o ignora un evento

El componente que crea un evento y lo dispara se llama:  
***source component***

## Un evento es un objeto de la clase EventObject



<i>User Action</i>	<i>Source Object</i>	<i>Event Type Fired</i>
Click a button	JButton	ActionEvent
Press return on a text field	TextField	ActionEvent
Select a new item	JComboBox	ItemEvent, ActionEvent
Select item(s)	JList	ListSelectionEvent
Click a check box	JCheckBox	ItemEvent, ActionEvent
Click a radio button	JRadioButton	ItemEvent, ActionEvent
Select a menu item	JMenuItem	ActionEvent
Move the scroll bar	JScrollBar	AdjustmentEvent
Move the scroll bar	JSlider	ChangeEvent
Window opened, closed, iconified, deiconified, or closing	Window	WindowEvent
Mouse pressed, released, clicked, entered, or exited	Component	MouseEvent
Mouse moved or dragged	Component	MouseEvent
Key released or pressed	Component	KeyEvent
Component added or removed from the container	Container	ContainerEvent
Component moved, resized, hidden, or shown	Component	ComponentEvent
Component gained or lost focus	Component	FocusEvent

El objeto fuente del evento se identifica con el método `getSource()` de la clase `EventObject`

# Eventos: Listeners, Registrations, y Handling

Java utiliza un modelo de delegación de eventos para manejar los eventos.  
**Patrón Observer.**

Un objeto fuente dispara un evento y el objeto interesado lo maneja.  
Este último se llama **Listener**

Un Listener para un evento debe cumplir dos cosas:

1. El objeto Listener debe estar registrado por el objeto Source
2. El objeto Listener debe ser una instancia de la correspondiente interface event-listener para tener el método correcto que procese el evento

## El objeto Source notifica a los listeners del evento invocando el manejador (handler) del objeto listener

source: SourceClass

+addXListener(XListener listener)

An event is triggered

Store in a list

event: XEvent

Invoke

listener1.handler(event)

listener2.handler(event)

...

listenern.handler(event)

listener1  
listener2  
...  
listenern

source: javax.swing.JButton

+addActionListener(ActionListener listener)

An event is triggered

Store in a list

event:  
ActionEvent

Invoke

listener1.actionPerformed(event)

listener2.actionPerformed(event)

...

listenern.actionPerformed(event)

listener1  
listener2  
...  
listenern

# Interfaces gráficas de usuario.

**Eventos: Listeners, Registrations, y Handling**

 **Clases internas**

**Alternativas para definir clases Listener**

**Menus, ToolBars y Dialogs**

**Applets**



# Clases internas

Las clases internas son declaradas siempre dentro de otras clases que las contienen

Puede ser definidas en cualquier lugar

- \* Anidada dentro de otras clases
- \* En la invocación de un método


Tienen acceso a los miembros y métodos de todas las clases externas a ellas

Pueden tener nombres o ser anónimas

Pueden extender de otra clase o implementar interfaces

Muy útiles para el manejo de eventos

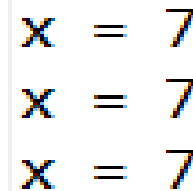
```
public class Test {  
    ...  
}  
  
public class A {  
    ...  
}
```



```
public class Test {  
    ...  
    // Inner class  
    public class A {  
        ...  
    }  
}
```

# Clases internas

```
public class Externa {  
    private int x = 7;  
  
    public static void main(String[] args) {  
        Externa e = new Externa(); //Intancia de la clase contenedora  
        Interna i1 = e.new Interna();  
        Externa.Interna i2 = e.new Interna();  
        Externa.Interna i3 = new Externa().new Interna();  
        i1.accesoExterna();  
        i2.accesoExterna();  
        i3.accesoExterna();  
    }  
  
    //definicion de una clase interna  
    class Interna {  
        public void accesoExterna() {  
            System.out.println("x = " + x);  
        }  
    }  
}
```




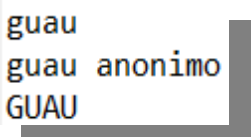
x = 7  
x = 7  
x = 7

# Clases internas anónimas

```
interface Ladrador{
    public void ladra();
}

public class Perro implements Ladrador {
    public void ladra() {
        System.out.println("guau");
    }
    public static void main(String[] args) {
        //Instancia de la clase contenedora
        Perro p = new Perro();
        p.ladra();
        PerroGuardia p1 = new PerroGuardia();
        p1.ladra();
    }
}

class PerroGuardia implements Ladrador{
    Ladrador perro = new Perro() {
        public void ladraladra() {
            System.out.println("GUAU");
        }
        public void ladra() {
            System.out.println("guau anonimo");
            ladraladra();
        }
    };
    public void ladra() {
        perro.ladra(); // tiene un metodo pop()
        //perro.ladraladra(); //no tiene un metodo ladraladra
    }
}
```



Solo se puede llamar a métodos que contenga la clase superior a la clase anónima

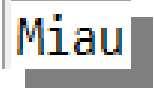

Las clases anónimas pueden estar basadas en interfaces

Se puede crear una clase anónima y pasar su instancia directamente en un método

```
public class Externa3 {
    public static void main(String[] args) {
        Externa3 e = new Externa3();
        e.doAlgo();
    }
    public void doAlgo() {
        Gato gato = new Gato();
        gato.maulla(new Gatuno() {
            public void maulla() {
                System.out.println("Miau");
            }
        });
    }
} // cierre de clase

interface Gatuno{
    public void maulla();
}

class Gato {
    public void maulla(Gatuno g){
        g.maulla();
    }
}
```



# Clases internas locales a métodos

```
public class Externa2 {
    private String x = "Externa2";

    public static void main(String[] args) {
        //Instancia de la clase contenedora
        Externa2 e = new Externa2();
        e.doAlgo();
    }

    void doAlgo() {
        final String z = "variable local";
        class Interna {
            public void miraExterna() {
                System.out.println("x de externa : " + x);
                System.out.println("variable local z : " + z);
                // compila porque z es final
            } // cierra método clase interna
        } // cierra definición clase interna

        Interna i = new Interna();
        i.miraExterna();

    } // cierra clase interna en metodo doAlgo
} // cierra clase externa

x de externa : Externa2
variable local z : variable local
```

Las clases internas locales no pueden ser instanciadas fuera del método donde se las declararon

Las clases internas locales pueden ser instanciadas solo después de la declaración de la clase

Las clases internas locales no puede utilizar las variables locales a los métodos en los cuales fueron declaradas a menos que esta variables sean marcadas como finales

```

public class AnonymousListenerDemo extends JFrame {
    public AnonymousListenerDemo() {
        JButton jbtAbrir = new JButton("Abrir");
        JButton jbtNuevo = new JButton("Nuevo");
        JButton jbtGuardar = new JButton("Guardar");
        JPanel panel = new JPanel();
        panel.add(jbtAbrir);
        panel.add(jbtNuevo);
        panel.add(jbtGuardar);
        getContentPane().add(panel);
        // Crea y registra: anonymous inner class listener
        jbtAbrir.addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.out.println("Procesa Abrir");
                }
            }
        );
        jbtNuevo.addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.out.println("Procesa Nuevo");
                }
            }
        );
        jbtGuardar.addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.out.println("Procesa Guardar");
                }
            }
        );
    }

    public static void main(String[] args) {
        JFrame frame = new AnonymousListenerDemo();
        frame.setTitle("AnonymousListenerDemo");
        frame.setLocationRelativeTo(null); // Centra el frame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}

```

```

Procesa Abrir
Procesa Nuevo
Procesa Abrir
Procesa Guardar

```



# Interfaces gráficas de usuario.

**Eventos: Listeners, Registrations, y Handling**

**Clases internas**

**→ Alternativas para definir clases Listener**

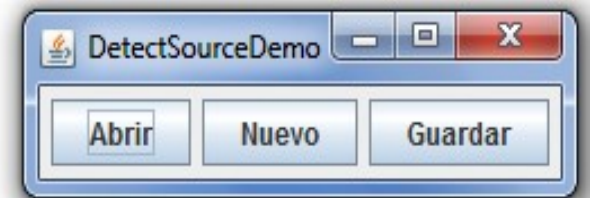
**Menus, ToolBars y Dialogs**

**Applets**

# Alternativas para definir clases Listener

```
public class DetectaSourceDemo extends JFrame {  
    private JButton jbtAbrir = new JButton("Abrir");  
    private JButton jbtNuevo = new JButton("Nuevo");  
    private JButton jbtGuardar = new JButton("Guardar");  
  
    public DetectaSourceDemo() {  
        JPanel panel = new JPanel();  
        panel.add(jbtAbrir);  
        panel.add(jbtNuevo);  
        panel.add(jbtGuardar);  
        getContentPane().add(panel);  
        // Create a listener  
        ButtonListener listener = new ButtonListener();  
        // Registra listener  
        jbtAbrir.addActionListener(listener);  
        jbtNuevo.addActionListener(listener);  
        jbtGuardar.addActionListener(listener);  
    }  
    class ButtonListener implements ActionListener {  
        public void actionPerformed(ActionEvent e) {  
            if (e.getSource() == jbtNuevo)  
                System.out.println("Procesa Nuevo");  
            else if (e.getSource() == jbtAbrir)  
                System.out.println("Procesa Abrir");  
            else if (e.getSource() == jbtGuardar)  
                System.out.println("Procesa Guardar");  
        }  
    }  
}
```

Procesa Nuevo  
Procesa Guardar  
Procesa Nuevo  
Procesa Abrir



```

public class FrameComoListenerDemo extends JFrame
    implements ActionListener { ←
    private JButton jbtAbrir = new JButton("Abrir");
    private JButton jbtNuevo = new JButton("Nuevo");
    private JButton jbtGuardar = new JButton("Guardar");

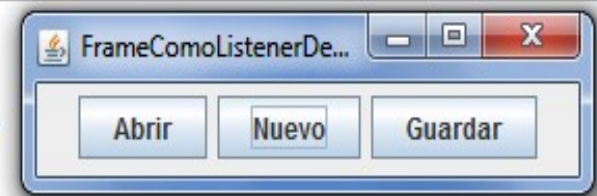
    public FrameComoListenerDemo() {
        JPanel panel = new JPanel();
        panel.add(jbtAbrir);
        panel.add(jbtNuevo);
        panel.add(jbtGuardar);
        getContentPane().add(panel);
        // Registra listener
        jbtAbrir.addActionListener(this);
        jbtNuevo.addActionListener(this);
        jbtGuardar.addActionListener(this);
    }
    /** Implementa actionPerformed */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == jbtNuevo)
            System.out.println("Procesa Nuevo");
        else if (e.getSource() == jbtAbrir)
            System.out.println("Procesa Abrir");
        else if (e.getSource() == jbtGuardar)
            System.out.println("Procesa Guardar");
    }
}

```

```

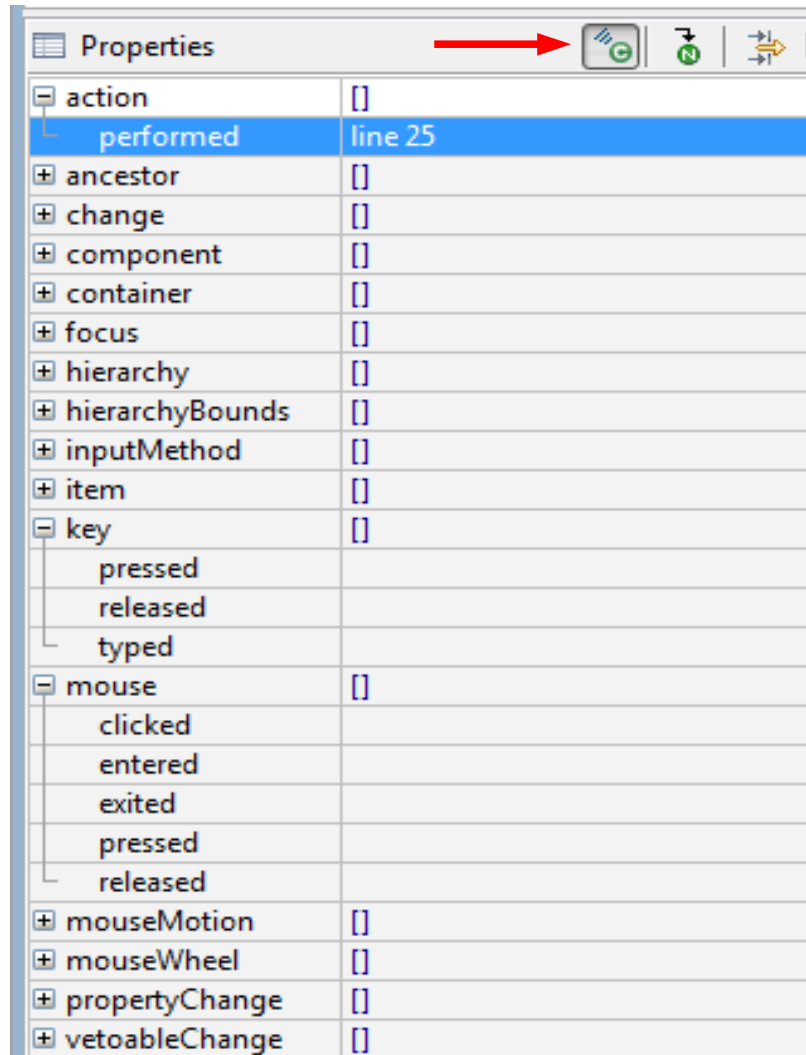
Procesa Abrir
Procesa Nuevo
Procesa Guardar
Procesa Nuevo

```





# Otros Eventos: Mouse, Key,...



# Interfaces gráficas de usuario.

**Eventos: Listeners, Registrations, y Handling**

**Clases internas**

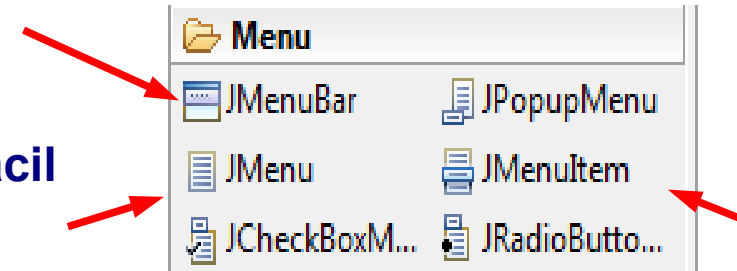
**Alternativas para definir clases Listener**

 **Menus, ToolBars y Dialogs**

**Applets**

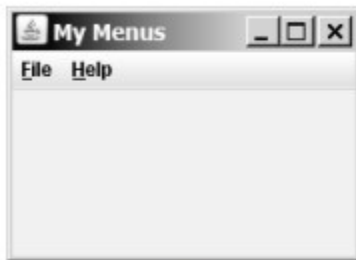
# Menus

Los menús son ampliamente utilizados en aplicaciones GUI para hacer la selección más fácil



La secuencia de implementación de menús en Java es como sigue

1. Crear una barra de menú y asociarlo con un JFrame utilizando el método setJMenuBar
2. Crear menus y a asociarlos con la barra de menú
3. Crear JMenuItem's y añadirlos a los menús



(a)



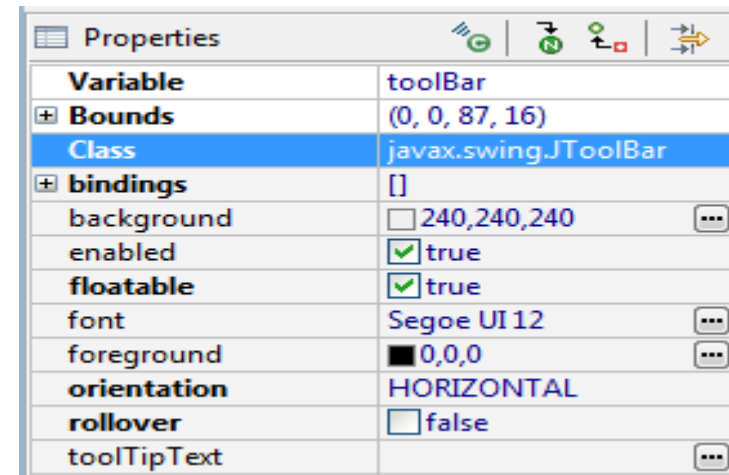
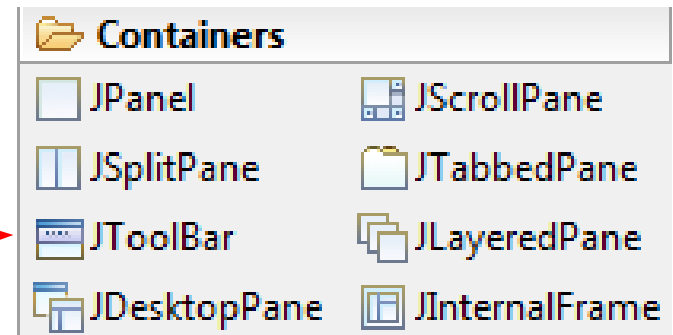
(b)



(c)

# JToolBar

Una barra de herramientas es un contenedor que se utiliza para mantener los comandos que también aparecen en los menús



# Dialogs: JOptionPane

Un cuadro de diálogo se utiliza normalmente como una ventana temporal para recibir información adicional por parte del usuario o para proporcionar una notificación de que un evento ha ocurrido

La clase JOptionPane puede ser utilizada para crear cuatro tipos de diálogos estándar:

**MessageDialog** muestra un mensaje y espera a que el usuario haga clic en Aceptar.

**ConfirmationDialog** muestra una pregunta y solicita la confirmación, por ejemplo, Aceptar o Cancelar.

**InputDialog** muestra una pregunta y recibe la entrada del usuario de un campo de texto, un cuadro combinado o lista.

**OptionDialog** muestra una pregunta y obtiene la respuesta del usuario a partir de un conjunto de opciones.

**JOptionPaneDemo**

?

Select annual interest rate:

5.0  
5.125  
5.25  
5.375  
5.5  
5.625  
5.75  
5.875  
6.0  
6.125

OK Cancel

**JOptionPaneDemo**

?

Select number of years:

15

OK Cancel

**JOptionPaneDemo**

?

Enter loan amount,  
for example, 150000 for \$150000

250000

OK Cancel

**JOptionPaneDemo**

i

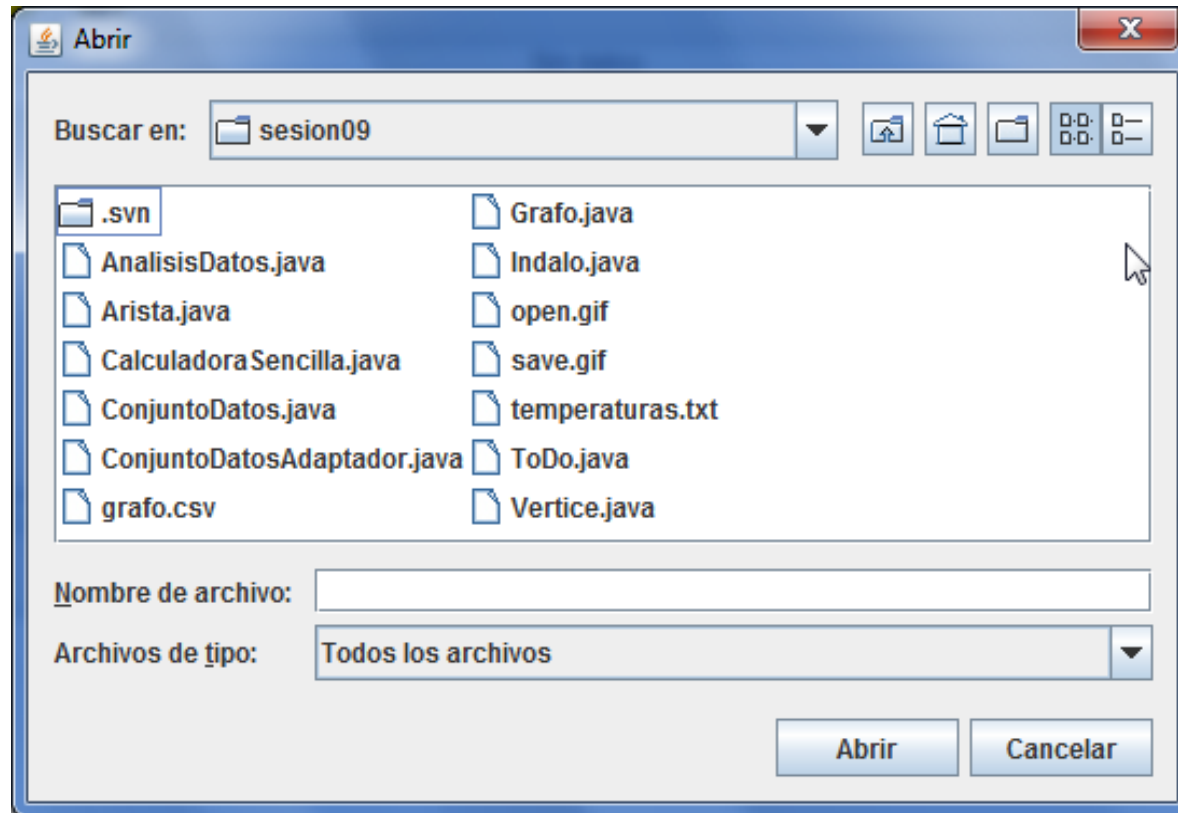
Interest Rate: 5.625% Number of Years: 15 Loan Amount: \$250000.0  
Monthly Payment: \$2059.32  
Total Payment: \$370679.26

Payment#	Interest	Principal	Balance
1	1171.87	887.45	249112.55
2	1167.71	891.61	248220.94
3	1163.53	895.79	247325.15
4	1159.33	899.99	246425.16
5	1155.11	904.21	245520.95
6	1150.87	908.45	244612.5
7	1146.62	912.7	243699.8

OK

# JFileChooser

Muestra un cuadro de diálogo en el que el usuario puede navegar a través del sistema de archivos y seleccionar los archivos para cargar o guardar



# JFileChooser

JFileChooser es una subclase de JComponent

Hay varias maneras de construir JFileChooser. La más simple es utilizar un constructor sin argumentos

JFileChooser puede aparecer en dos tipos: open y save. Open es para abrir un archivo, y save es para almacenar un archivo

```
public int showOpenDialog(Component parent)
```

```
public int showSaveDialog(Component parent)
```

## Propiedades importantes:

dialogType

dialogTitle

currentDirectory

selectedFile

selectedFiles

multiSelectionEnabled



```

btnJFChooser1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        visualizar1();
    }
});

```

```

btnJFChooser2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        visualizar2();
    }
});

```

```

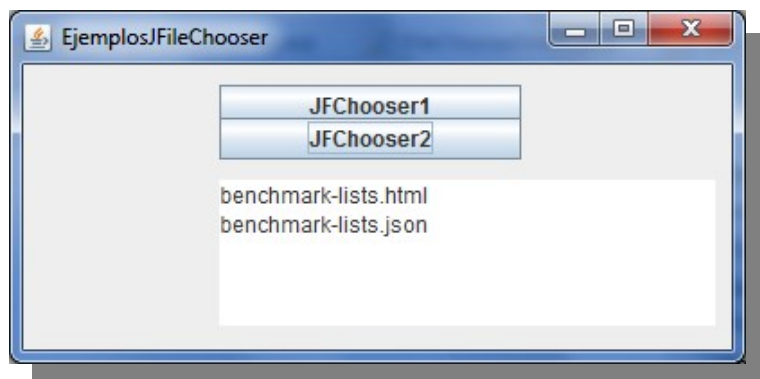
private void visualizar1() {
    JFileChooser fc = new JFileChooser();
    int resultado = fc.showOpenDialog(this);
    //int resultado = fc.showSaveDialog(contentPane);
    if(resultado == JFileChooser.APPROVE_OPTION) {
        textArea.append(fc.getSelectedFile().getName()+"\n");
    }
}

```

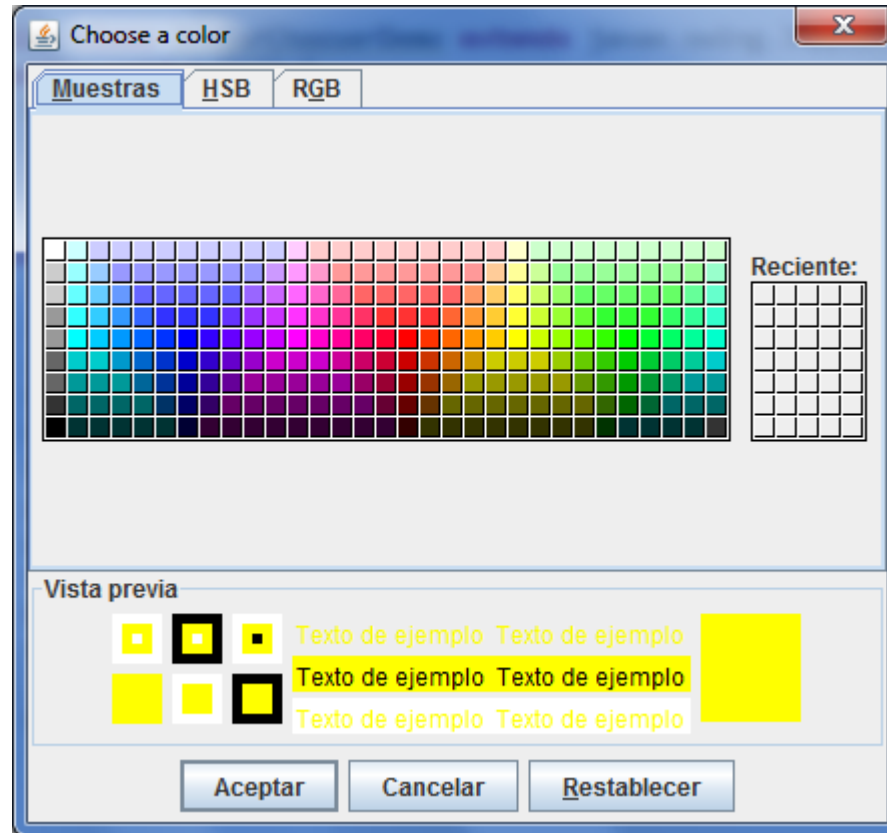
```

private void visualizar2() {
    JFileChooser fc = new JFileChooser();
    fc.setDialogTitle("Abierto con .showSaveDialog");
    fc.setCurrentDirectory(new File("."));
    fc.setDialogType(JFileChooser.APPROVE_OPTION);
    fc.setMultiSelectionEnabled(true);
    int resultado = fc.showDialog(this, "Elegir");
    if(resultado == JFileChooser.APPROVE_OPTION) {
        File[] files = fc.getSelectedFiles();
        for (int i = 0; i < files.length; i++) {
            textArea.append(files[i].getName()+"\n");
        }
    }
}

```



# JColorChooser



# Interfaces gráficas de usuario.

**Eventos: Listeners, Registrations, y Handling**

**Clases internas**

**Alternativas para definir clases Listener**

**Menus, ToolBars y Dialogs**

 **Applets**

# Applets

Son programas con interfaz gráfica que se ejecutan dentro de un navegador web

Para utilizar los componentes Swing en los applets Java es necesario crear una clase que extienda `javax.swing.JApplet`, que es una subclase de `java.applet.Applet`.

```
public class DisplayLabelApplet01 extends JApplet {
    public DisplayLabelApplet01() {
        add(new JLabel("¡Hola Mundo!", JLabel.CENTER));
    }
}

public class DisplayLabelApplet02 extends JApplet {
    public DisplayLabelApplet02() {
        add(new JLabel("¡Hola Mundo!", JLabel.CENTER));
    }
    public static void main(String[] args) {
        // Crea un frame
        JFrame frame = new JFrame("Applet en el frame");
        // Crea una instancia del applet
        DisplayLabelApplet02 applet = new DisplayLabelApplet02();
        // Añade el applet al frame
        frame.add(applet);
        // Visualiza el frame
        frame.setSize(300, 100);
        frame.setLocationRelativeTo(null); // Centrado
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

# HTML con el tag <applet>

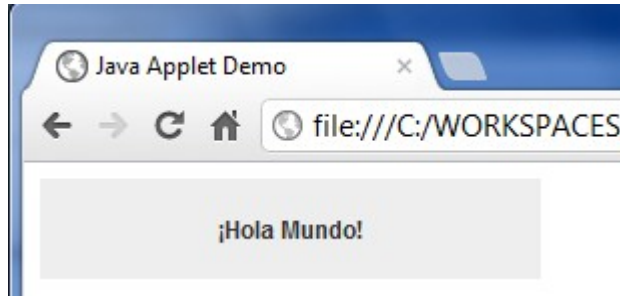
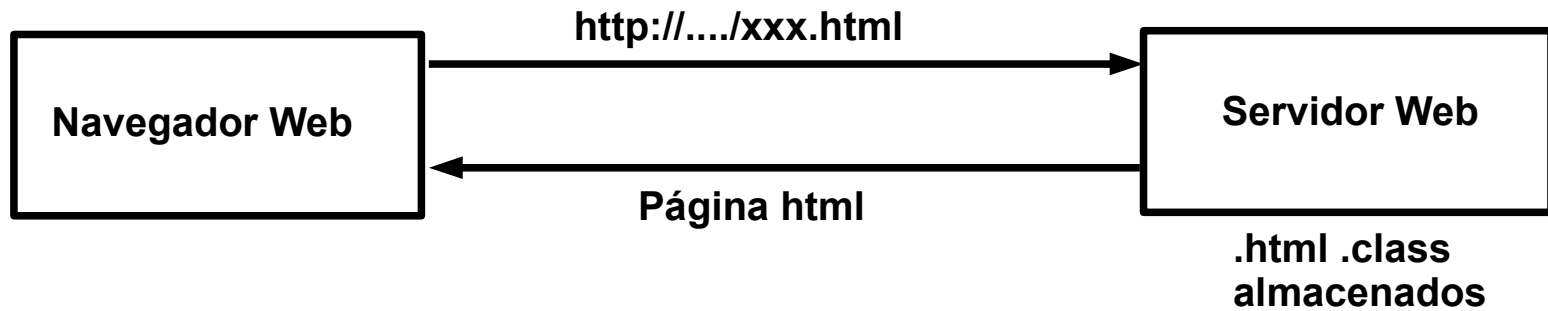
Para ejecutar un applet en el navegador es necesario crear una archivo HTML con el tag <applet>

```
<html>
  <head>
    <title>Java Applet Demo</title>
  </head>
  <body>
    <applet
      codebase="../../../bin"
      code = "org/mp/tema05/DisplayLabelApplet01.class"
      width = 250
      height = 50>
    </applet>
  </body>
</html>
```

## Sintaxis completa del tag <applet>tag

```
<applet
  [codebase = applet_url]
  code = classfilename.class
  width = applet_viewing_width_in_pixels
  height = applet_viewing_height_in_pixels
  [archive = archivefile]
  [vspace = vertical_margin]
  [hspace = horizontal_margin]
  [align = applet_alignment]
  [alt = alternative_text]
>
<param name = param_name1 value = param_value1>
<param name = param_name2 value = param_value2>
...
<param name = param_name3 value = param_value3>
</applet>
```

# Visualizar un applet en el navegador



## Visualizar un applet en Applet Viewer Utility



# Restricciones de seguridad para los applets

Java utiliza el llamado "modelo de seguridad sandbox" para la ejecución de applets para evitar que los programas "mal intencionados" dañen el sistema en el que el navegador se está ejecutando.

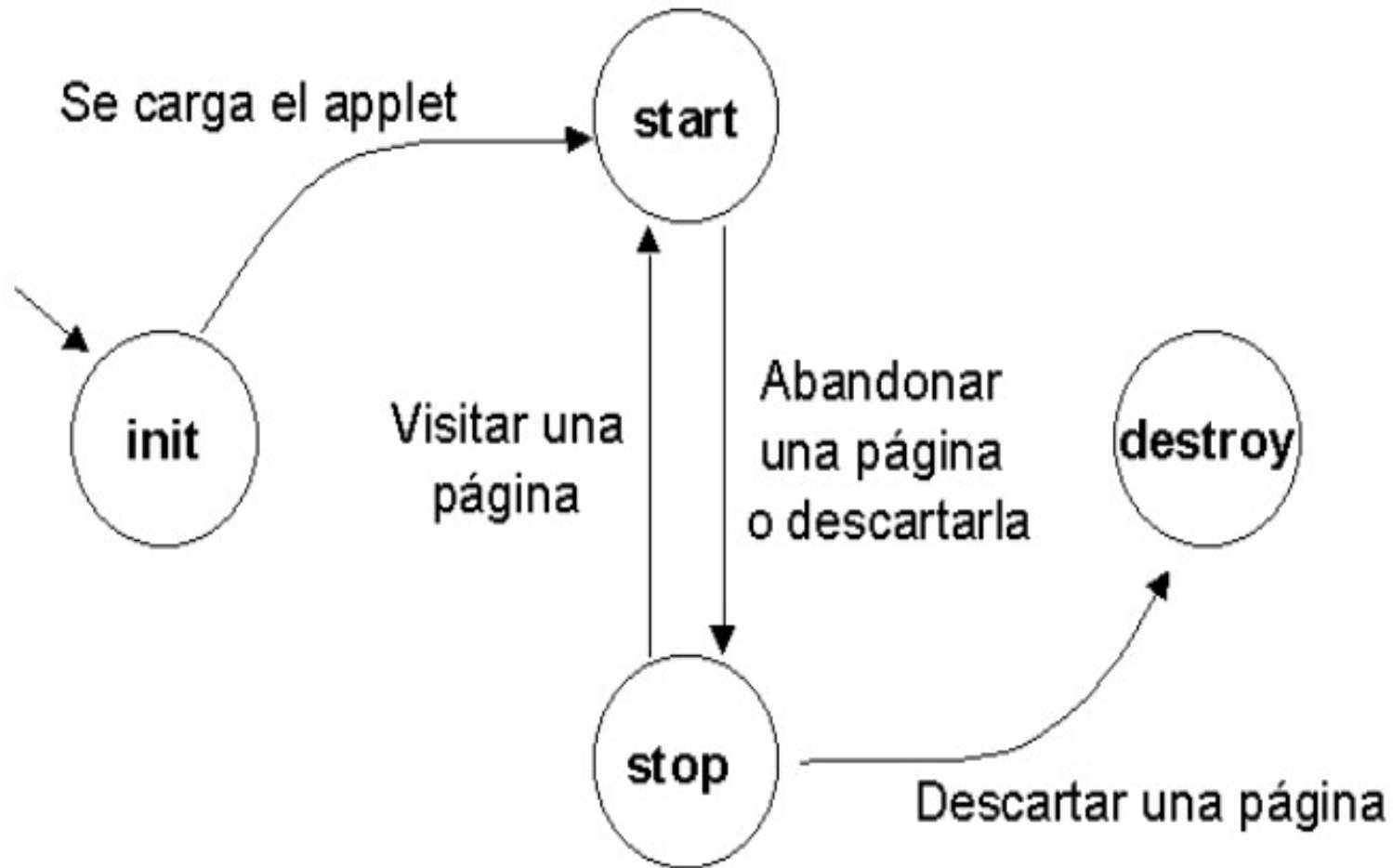
**Los applets no pueden leer o escribir, en el sistema de archivos**

**Los applets no se les permite ejecutar programas en el ordenador del navegador**

**Los applets no se les permite establecer conexiones entre el ordenador del usuario y cualquier otro equipo, a excepción de el servidor donde se almacenan los applets**

# Applet: Métodos del ciclo de vida

Los applets se ejecutan en el contenedor del applet, que es un plug-in de navegador Web. La clase Applet contiene los métodos el `init ()`, `start ()`, `stop ()` y `destroy ()`, conocidos como los métodos del ciclo de vida.





# Paso de cadenas (String) a un Applet

Para pasar un cadena a un applet, el parámetro se debe declarar en el archivo HTML y debe ser leído por el applet cuando se inicializa. Los parámetros se declaran con la etiqueta `<param>`

`<param name = parametername value = stringvalue />`

```
<html>
<head>
  <title>Java Applet </title>
</head>
<body>
  <applet
    codebase="../../../../bin"
    code = "org/mp/tema05/DisplayLabelApplet03.class"
    width = 250
    height = 150>
    <param name = Mensaje value = "Eurocopa 2012" />
  </applet>
</body>
</html>
```

`public String getParameter(String parametername);`

```
public class DisplayLabelApplet03 extends JApplet {
```

```
    private String mensaje;
```

```
    public void init() {
```

```
        // Obtiene los parametros desde el HTML
```

```
        mensaje = getParameter("Mensaje");
```

```
        add(new JLabel(mensaje, JLabel.CENTER));
```

```
    }
```

# Ejecutar un applets como una aplicación

En general, un Applet se puede convertir en una aplicación sin pérdida de funcionalidad. Una aplicación puede ser convertido a un Applet siempre que no viole las restricciones de seguridad impuestas a los applets

Debe implementar un método principal en un applet para permitir que el applet se ejecute como una aplicación. En teoría, se difumina la diferencia entre applets y aplicaciones.

Puede escribir una clase que es a la vez un applet y una aplicación

```
public static void main(String[] args) {  
    // Crea un frame  
    JFrame frame = new JFrame("Applet en el frame");  
    // Crea una instancia del applet  
    → DisplayLabelApplet02 applet = new DisplayLabelApplet02();  
    // Añade el applet al frame  
    → frame.add(applet);  
    // Visualiza el frame  
    frame.setSize(300, 100);  
    frame.setLocationRelativeTo(null); // Centrado  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setVisible(true);  
}
```