

MÓDULO A

UNIDADES DIDÁCTICAS:

1. Paquete AWT
2. Paquete Swing
3. Applets

Paquete AWT

Tema 1.1

Índice de la unidad:

1. Introducción
2. Componentes visuales AWT
3. Layout Managers
4. Gestión de eventos
5. Otras clases
6. Editor visual AWT/Swing en Eclipse

En las siguientes Unidades, vamos a estudiar la programación visual en Java. Comenzaremos en esta Unidad repasando los fundamentos básicos sobre los que se apoya el funcionamiento de los interfaces gráfico en Java, partiendo del primer framework que se implementó e incluyó en la JDK: AWT. En la siguiente Unidad veremos otro framework posterior más flexible y potente llamado Swing (que en la JDK 1.2 se incluyó como parte del lenguaje), pero que se apoya y basa en todo lo estudiado en la AWT.

1. Introducción

AWT, son las siglas de: Abstract Window Toolkit

Es una librería de clases Java para el desarrollo de interfaces de usuario gráficas (GUI).

Por tratarse de código Java, las aplicaciones serán independientes de la plataforma. No así su apariencia visual, es decir, su *Look & Feel*, que será el de la plataforma donde se ejecute.

Es la librería visual básica. Sobre ella se construyó a posteriori otra más flexible y potente: JFC/Swing que trataremos con profundidad en la siguiente Unidad. Pero los fundamentos básicos los estableció la AWT.

La AWT se encuentra en el paquete: `java.awt.*`

Dispone de la mayoría de controles visuales estándar:

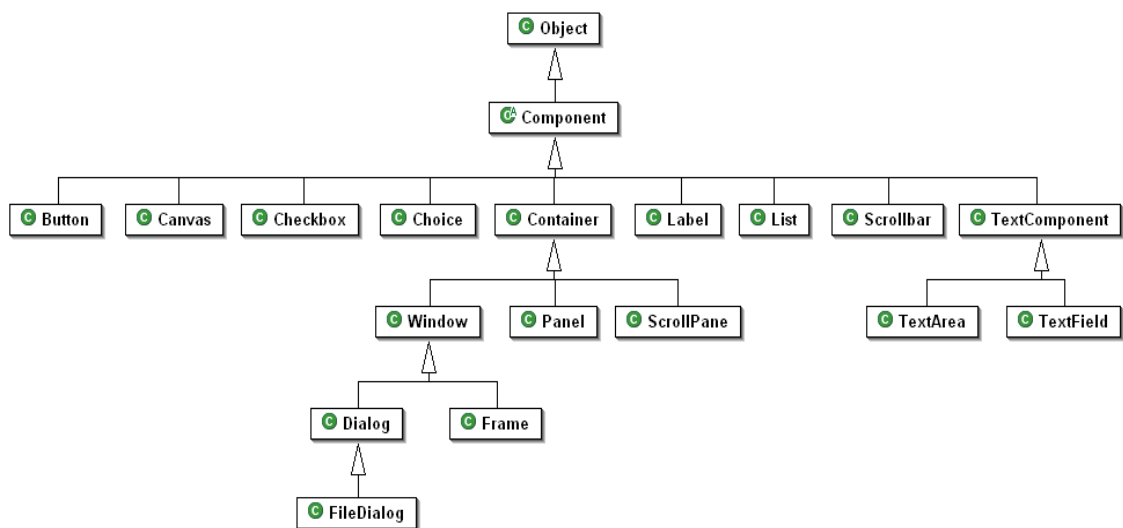
- Button (push, radio y check).
- Canvas.
- Frame, Dialog.
- Label.
- List, Choice.
- ScrollBar, ScrollPane.
- TextField, TextArea.

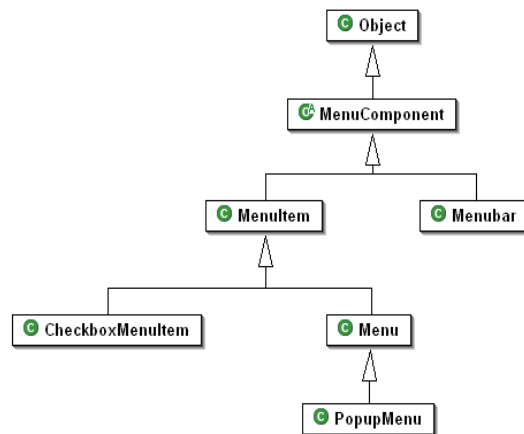
- Menu.

Los elementos básicos que componen la librería AWT son:

- Los componentes (**java.awt.Component**) como Buttons, Labels, Text-Fields, etc....
- Los contenedores (**java.awt.Container**) como los Frames, los Panels, etc.... que pueden contener componentes.
- Los gestores de posición (**java.awt.LayoutManager**) que gestionan la disposición de los componentes dentro de los contenedores.
- Los eventos (**java.awt.AWTEvent**) que avisan de las acciones del usuario.

Como ya hemos comentado en varias ocasiones, en la Programación Orientada Objetos, y por tanto en el lenguaje Java, siempre existe un diseño jerárquico de herencia. La AWT no es una excepción:





java.awt.Component es una clase abstracta que implementa toda la funcionalidad básica de las clases visuales.

Tiene métodos para:

- Mostrar y esconder.
- Rendering.
- Habilitar y deshabilitar, etc...

Y atributos como:

- Color del foreground y background.
- Tamaño y posición.

Por su lado, **java.awt.Container** es una clase que implementa la funcionalidad de contener a otros componentes. Y si nos fijamos en la jerarquía de clase, los contenedores son a su vez componentes.

Algunos contenedores típicos son:

- Window.
- Dialog y FileDialog.
- Frame.
- Panel: contenedor invisible.

Luego entonces, los contenedores sirven para agrupar componentes visuales. Pero, ¿cómo se distribuyen dichos componentes en su interior?

Para dicho cometido, se utilizan implementaciones del interface **java.awt.LayoutManager**

Cada contenedor tiene asociado un LayoutManager que distribuye los componentes en el interior del contenedor. Por ejemplo, un Panel tiene asociado por defecto una instancia de java.awt.FlowLayout

La posición de los componentes visuales es relativa al contenedor en el que se encuentra. La coordenada 0,0 es la esquina superior izquierda del contenedor.

La clase java.awt.Component implementa varios métodos para la gestión del tamaño y posicionamiento como por ejemplo:

- Rectangle getBounds();
- void setLocation(int x, int y);
- Point getLocation();
- void setBounds(int x, int y, int width, int height);
- Dimension getSize();
- void setSize(Dimension d);
- Container getParent();

Por su lado, la clase java.awt.Container posee varios métodos para acceder a los componentes que contiene, como por ejemplo:

- add(Component c); o add(Component c, Object o); // Inserta el componente c en el contenedor referenciado.
- remove(Component c); // Elimina el componente c del contenedor referenciado.
- Component[] getComponents(); // Devuelve un array con los componentes del contenedor referenciado.

A continuación, listamos los pasos habituales con un ejemplo, para crear un componente visual y añadirlo a un contenedor:

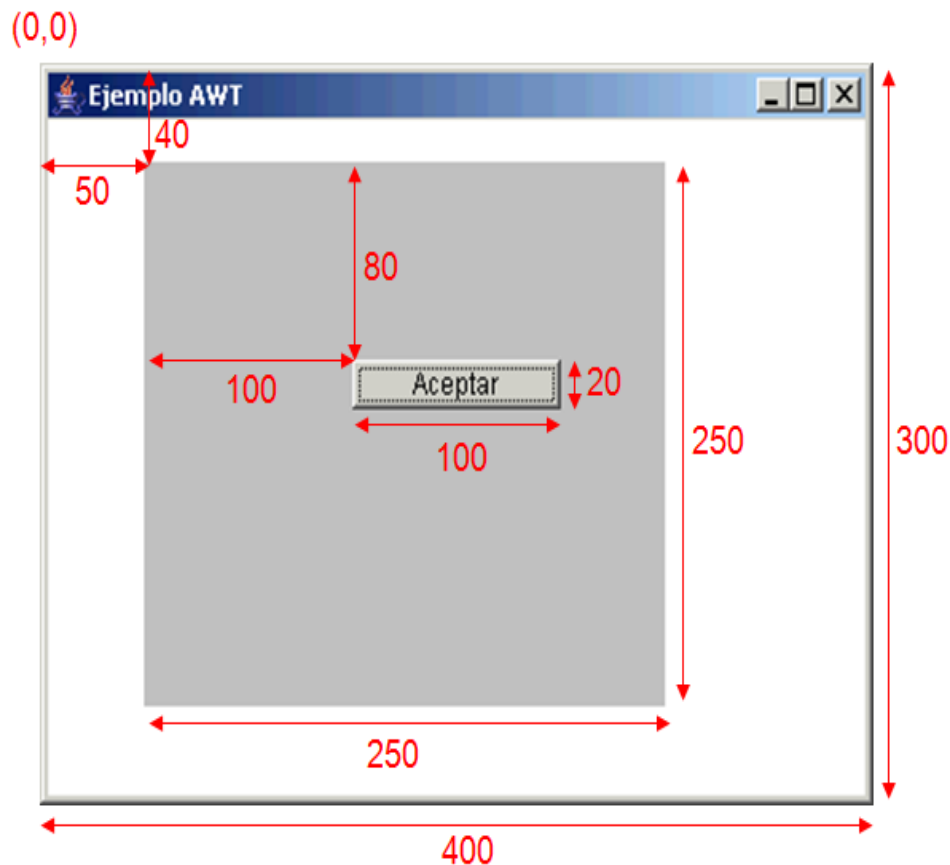
- Crear el componente: `Button b = new Button();`
- Añadir el componente al contenedor: `unContenedor.add(b);`
- Invocar métodos sobre el componente y manejar sus eventos:
`b.setText("Ok");`
- **Ejemplo:** Desarrollo de un ejemplo básico de contenedores con componentes:

```
import java.awt.*;

public class EjemploAWT
{
    public static void main(String[] args)
    {
        Frame frame = new Frame();
        frame.setLayout(null);
        frame.setBounds(0,0,400,300);
        frame.setTitle("Ejemplo AWT");
        Panel panel = new Panel();
        panel.setLayout(null);
        panel.setBounds(50,40,250,220);
        panel.setBackground(Color.LIGHT_GRAY);
        Button boton = new Button("Aceptar");
        boton.setBounds(100,80,100,20);

        panel.add(boton);
        frame.add(panel);

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



Para poder interactuar con los interfaces visuales, es decir, poder reaccionar y ejecutar código cuando se pulse un botón, se mueva el ratón, se teclee algo, etc... hay que gestionar los eventos. Trataremos este tema en un apartado posterior de esta misma Unidad.

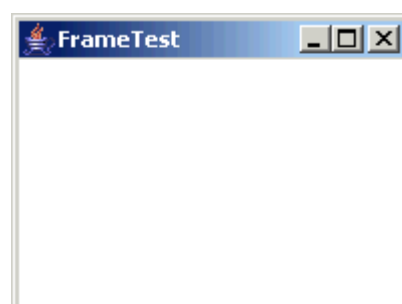
2. Componentes visuales AWT

Ya conocemos lo básico del funcionamiento de la programación visual. Ahora haremos un repaso mediante ejemplos de código de un gran porcentaje de los componentes visuales existentes en la AWT, mostrando visualmente a que corresponden. Para profundizar en cada uno de ellos, recomendamos examinar el API.

2.1 java.awt.Frame

```
import java.awt.Frame;

public class FrameTest
{
```



```

public static void main(String[] args)
{
    Frame f = new Frame();
    f.setTitle("FrameTest");
    f.setSize(200,150);
    f.setVisible(true);
}
}

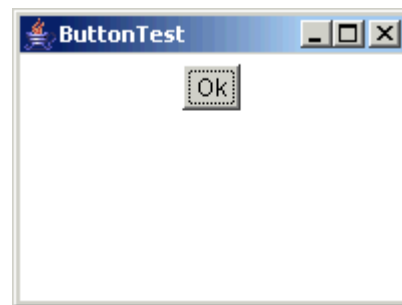
```

2.2 java.awt.Button

```

import java.awt.Button;
import java.awt.FlowLayout;
import java.awt.Frame;
public class ButtonTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("ButtonTest");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        Button b = new Button("Ok");
        f.add(b);
        f.setVisible(true);
    }
}

```

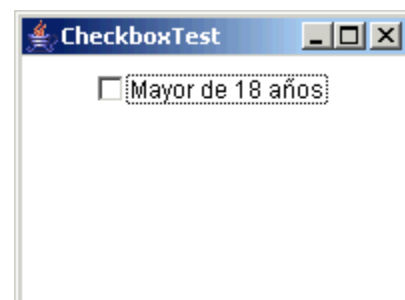


2.3 java.awt.Checkbox

```

import java.awt.Checkbox;
import java.awt.FlowLayout;
import java.awt.Frame;
public class CheckboxTest

```



```

{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("CheckboxTest");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        Checkbox c = new Checkbox("Mayor de 18 años");
        f.add(c);
        f.setVisible(true);
    }
}

```

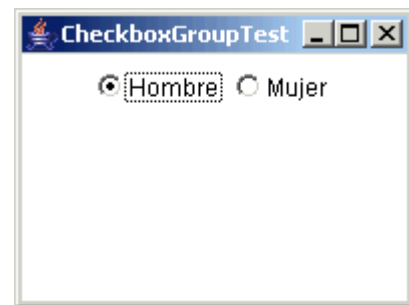
2.4 java.awt.CheckboxGroup

```

import java.awt.*;

public class CheckboxGroupTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("CheckboxGroupTest");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        CheckboxGroup cbg = new CheckboxGroup();
        Checkbox c1 = new Checkbox("Hombre",cbg,true);
        Checkbox c2 = new Checkbox("Mujer",cbg,false);
        f.add(c1);
        f.add(c2);
        f.setVisible(true);
    }
}

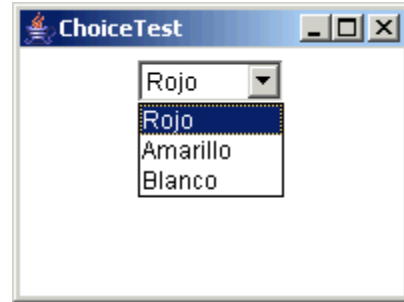
```



2.5 java.awt.Choice

```
import java.awt.*;

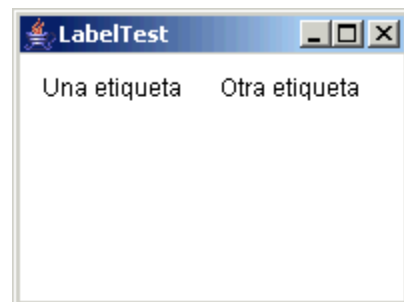
public class ChoiceTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("ChoiceTest");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        Choice cbg = new Choice();
        cbg.add("Rojo");
        cbg.add("Amarillo");
        cbg.add("Blanco");
        f.add(cbg);
        f.setVisible(true);
    }
}
```



2.6 java.awt.Label

```
import java.awt.*;

public class LabelTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("LabelTest");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        Label l1 = new Label("Una etiqueta");
        Label l2 = new Label();
        l2.setText("Otra etiqueta");
        f.add(l1);
        f.add(l2);
    }
}
```



```

        f.setVisible(true);
    }
}

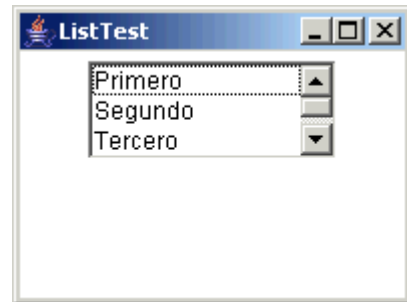
```

2.7 java.awt.List

```

import java.awt.*;
public class ListTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("ListTest");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        List l = new List(3);
        l.add("Primero");
        l.add("Segundo");
        l.add("Tercero");
        l.add("Cuarto");
        f.add(l);
        f.setVisible(true);
    }
}

```

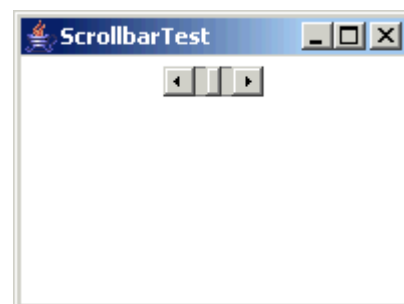


2.8 java.awt.Scrollbar

```

import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Scrollbar;
public class ScrollbarTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();

```



```

f.setTitle("ScrollbarTest");
f.setSize(200,150);
f.setLayout(new FlowLayout());
Scrollbar sb = new Scrollbar(Scrollbar.HORIZONTAL,0,5,-100,100);
f.add(sb);
f.setVisible(true);
}
}

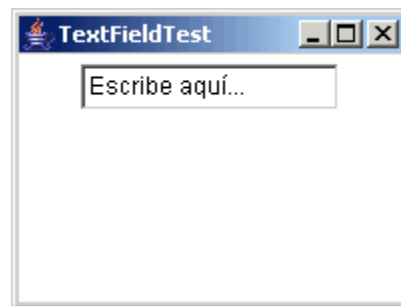
```

2.9 java.awt.TextField

```

import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.TextField;
public class TextFieldTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("TextFieldTest");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        TextField tf = new TextField("Escribe aquí...");
        f.add(tf);
        f.setVisible(true);
    }
}

```

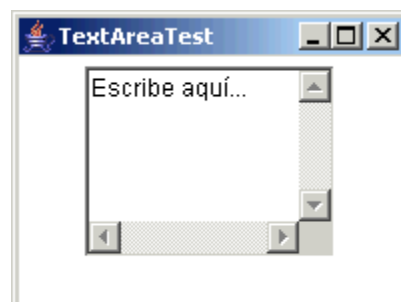


2.10 java.awt.TextArea

```

import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.TextArea;
public class TextAreaTest

```



```

{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("TextAreaTest");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        TextArea ta = new TextArea("Escribe aquí...",5,15);
        f.add(ta);
        f.setVisible(true);
    }
}

```

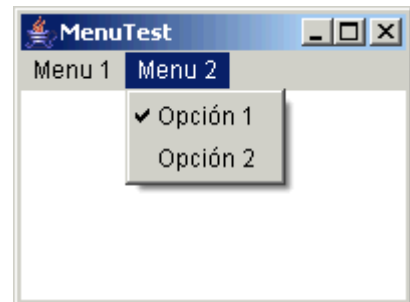
2.11 java.awt.Menu

```

import java.awt.*;

public class MenuTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("MenuTest");
        MenuBar mb = new MenuBar();
        Menu m1 = new Menu("Menu 1");
        m1.add(new MenuItem("Opción 1"));
        m1.add(new MenuItem("Opción 2"));
        Menu m2 = new Menu("Menu 2");
        m2.add(new CheckboxMenuItem("Opción 1", true));
        m2.add(new CheckboxMenuItem("Opción 2"));
        mb.add(m1);
        mb.add(m2);
        f.setMenuBar(mb);
        f.setSize(200,150);
        f.setVisible(true);
    }
}

```



2.12 java.awt.Dialog

```
import java.awt.Dialog;
import java.awt.Frame;
public class DialogTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("FrameTest");
        f.setSize(200,150);
        f.setVisible(true);
        Dialog d = new Dialog(f);
        d.setTitle("DialogTest");
        d.setBounds(50,50,70,50);
        d.setVisible(true);
    }
}
```



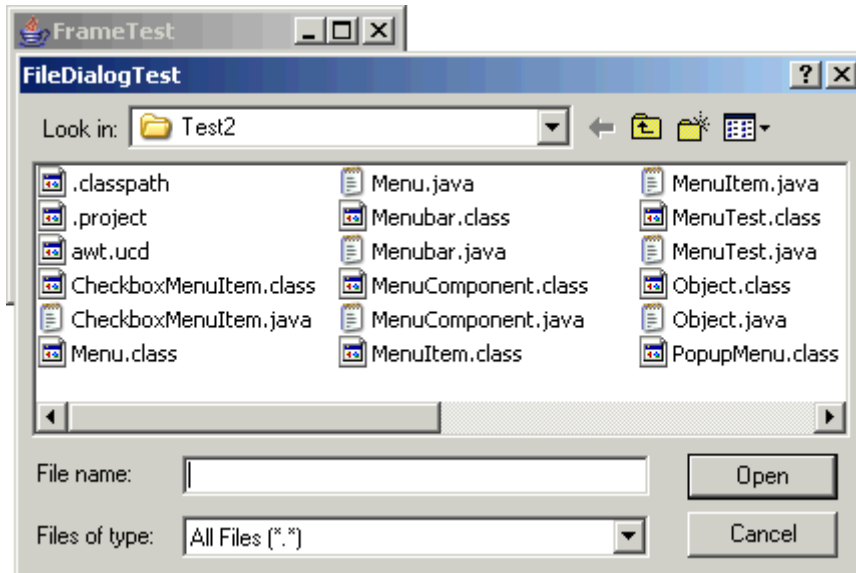
2.13 java.awt.FileDialog

```
import java.awt.FileDialog;
import java.awt.Frame;
public class DialogTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("FrameTest");
        f.setSize(200,150);
        f.setVisible(true);
        FileDialog d = new FileDialog(f);
        d.setTitle("FileDialogTest");
        d.setBounds(50,50,70,50);
    }
}
```

```

d.setVisible(true);
System.out.println(d.getFile());
    // Recibir el nombre del fichero seleccionado.
}
}

```



3. Layout Managers

Todos los contenedores AWT tienen asociado un LayoutManager para coordinar el tamaño y la situación de sus componentes. Por ejemplo:

- Panel tiene asociado un FlowLayout
- Frame tiene asociado un BorderLayout

Cada Layout se caracteriza por el estilo que emplea para situar los componentes en su interior:

- Alineación de izquierda a derecha.
- Alineación en rejilla.
- Alineación del frente a atrás.

¿Por qué debemos usar los Layout Managers?

Porque determinan el tamaño y la posición de los componentes en un contenedor. Tiene un API que permite al contenedor y al LayoutManager gestionar el cambio de tamaño del contenedor de manera transparente.

Por último, pero no menos importante, consiguen que la aplicación sea independiente de la resolución de las máquinas donde se ejecuta.

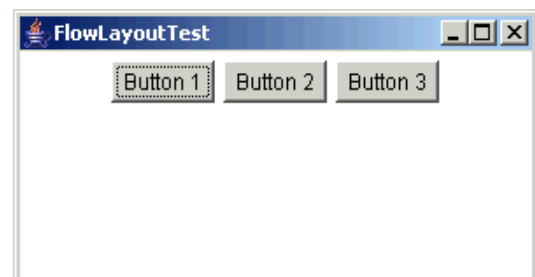
Todos los layout Managers implementan el interface `java.awt.LayoutManager`:

3.1 `java.awt.FlowLayout`

Sitúa los componentes de izquierda a derecha. Les modifica la posición pero no les modifica el tamaño.

```
import java.awt.*;

public class FlowLayoutTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("FlowLayoutTest");
        f.setSize(300,150);
        f.setLayout(new FlowLayout());
        Button b1 = new Button("Button 1");
        f.add(b1);
        Button b2 = new Button("Button 2");
        f.add(b2);
        Button b3 = new Button("Button 3");
        f.add(b3);
        f.setVisible(true);
    }
}
```

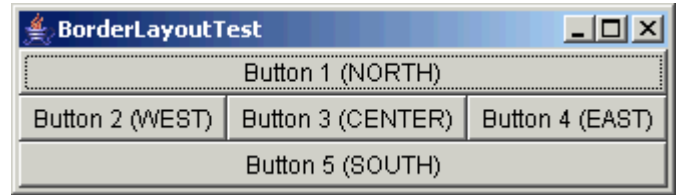


3.2 `java.awt.BorderLayout`

Se basa en los puntos cardinales. Modifica tanto la posición como el tamaño de los componentes.

```
import java.awt.*;

public class BorderLayoutTest
{
    public static void main(String[]
args)
    {
        Frame f = new Frame();
        f.setTitle("BorderLayoutTest");
        f.setLayout(new BorderLayout());
        Button b1 = new Button("Button 1 (NORTH)");
        f.add(b1, BorderLayout.NORTH);
        Button b2 = new Button("Button 2 (WEST)");
        f.add(b2, BorderLayout.WEST);
        Button b3 = new Button("Button 3 (CENTER)");
        f.add(b3, BorderLayout.CENTER);
        Button b4 = new Button("Button 4 (EAST)");
        f.add(b4, BorderLayout.EAST);
        Button b5 = new Button("Button 5 (SOUTH)");
        f.add(b5, BorderLayout.SOUTH);
        f.pack(); // El método pack, hace que el contenedor pregunte a su
                // LayoutManager el tamaño mínimo para que todos sus
                // componentes se puedan ver. Y se ajusta a ese tamaño
        f.setVisible(true);
    }
}
```



3.3 java.awt.CardLayout

Permite al desarrollador intercambiar distintas vistas como si se tratase de una baraja. Modifica tanto la posición como el tamaño de los componentes.

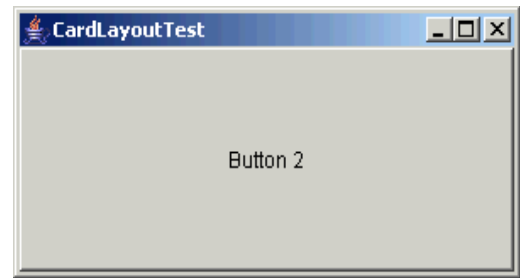
```
import java.awt.*;

public class CardLayoutTest
```

```

{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("CardLayoutTest");
        f.setSize(300,150);
        CardLayout cl = new CardLayout();
        f.setLayout(cl);
        Button b1 = new Button("Button 1");
        f.add(b1,"uno");
        Button b2 = new Button("Button 2");
        f.add(b2,"dos");
        Button b3 = new Button("Button 3");
        f.add(b3,"tres");
        f.setVisible(true);
        cl.show(f,"dos"); // Otras posibilidades: cl.first(f), cl.last(f) y cl.next(f);
    }
}

```



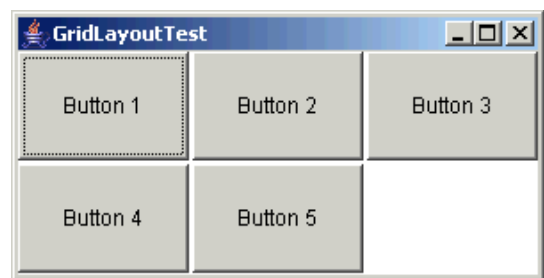
3.4 java.awt.GridLayout

Usa una matriz en la que sitúa cada uno de los componentes. El tamaño de todas las celdas es igual.

```

import java.awt.Button;
import java.awt.Frame;
import java.awt.GridLayout;
public class GridLayoutTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("GridLayoutTest");
        f.setSize(300,150);
        f.setLayout(new GridLayout(2,3,2,2));
        f.add(new Button("Button 1"));
    }
}

```



```

        f.add(new Button("Button 2"));
        f.add(new Button("Button 3"));
        f.add(new Button("Button 4"));
        f.add(new Button("Button 5"));
        f.setVisible(true);
    }
}

```

3.5 java.awt.GridBagLayout

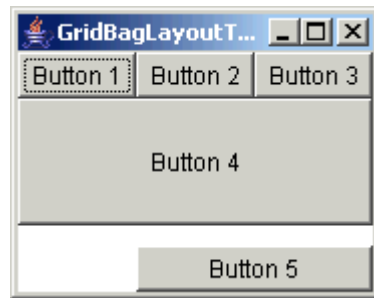
Similar al anterior, pero no fuerza a que todos los componentes tengan el mismo tamaño.

```

import java.awt.*;

public class GridBagLayoutTest
{
    public static void main(String[] args)
    {

```



```

        Frame frame = new Frame("GridBagLayoutTest");
        frame.setLayout(new GridBagLayout());

```

```

        Button button = new Button("Button 1");
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;
        c.weightx = 0.5;
        c.gridx = 0;
        c.gridy = 0;
        frame.add(button, c);

```

```

        button = new Button("Button 2");
        c.gridx = 1;
        c.gridy = 0;
        frame.add(button, c);
        button = new Button("Button 3");
        c.gridx = 2;
        c.gridy = 0;

```

```

        frame.add(button, c);
        button = new Button("Button 4");
        c.ipady = 40;
        c.weightx = 0.0;
        c.gridwidth = 3;
        c.gridx = 0;
        c.gridy = 1;
        frame.add(button, c);
        button = new Button("Button 5");
        c.ipady = 0;
        c.weighty = 1.0;
        c.anchor = GridBagConstraints.PAGE_END;
        c.insets = new Insets(10,0,0,0);
        c.gridx = 1;
        c.gridwidth = 2;
        c.gridy = 2;
        frame.add(button, c);

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

```

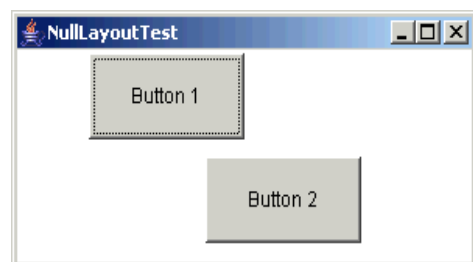
3.6 null LayoutManager

Es otra opción también válida, que consiste en prescindir del uso de los Layout Managers. En tal caso, el desarrollador es responsable de ajustar todas las posiciones y tamaños de los distintos componentes visuales utilizados en la aplicación.

```

import java.awt.Button;
import java.awt.Frame;
public class NullLayoutTest
{
    public static void main(String[] args)
    {

```



```
Frame f = new Frame();
f.setTitle("NullLayoutTest");
f.setSize(300,150);
f.setLayout(null);
Button b1 = new Button("Button 1");
b1.setBounds(50,25,100,50);
f.add(b1);
Button b2 = new Button("Button 2");
b2.setBounds(125,85,100,50);
f.add(b2);
f.setVisible(true);
}
```

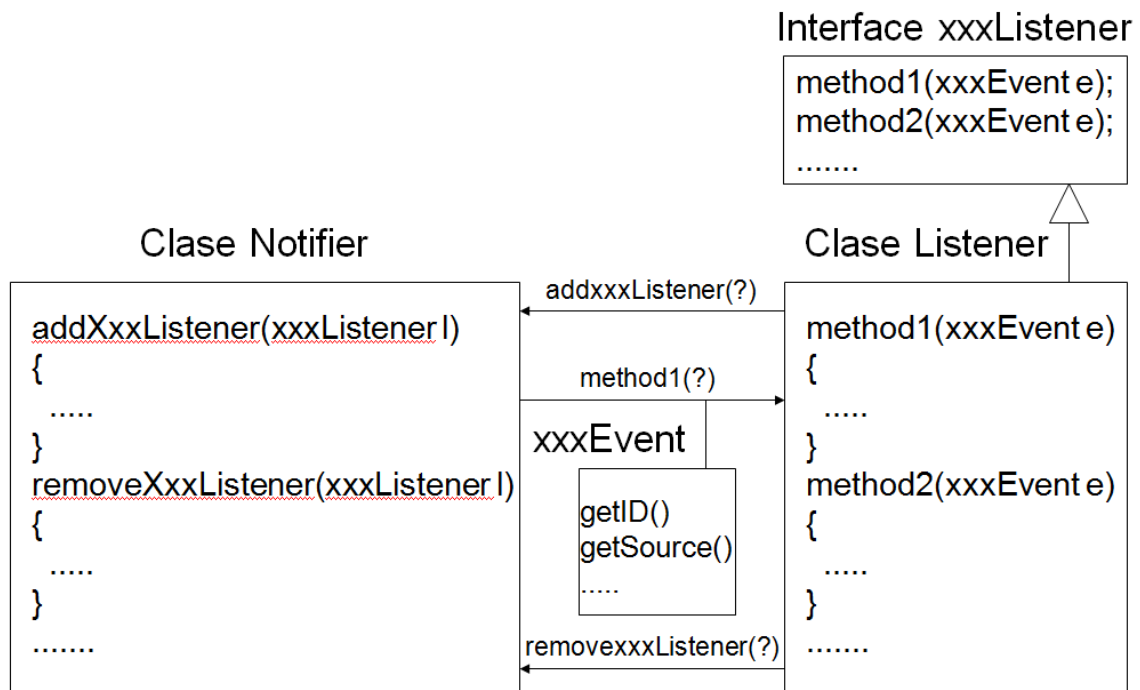
4. Gestión de eventos

Un evento es una encapsulación de una información que puede ser enviada a la aplicación de manera asíncrona.

Los eventos pueden corresponder a acciones físicas (ratón y teclado) y a acciones lógicas.

java.util.EventObject es la clase padre de todos los eventos. Su subclase **java.awt.AWTEvent** es la clase padre de todos los eventos AWT.

En el siguiente gráfico, podemos ver reflejado el funcionamiento de la gestión de eventos en Java:



Básicamente, contamos con dos piezas fundamentales. Un notificador (*Notifier*), que es quién generará los eventos, y un escuchador (*Listener*) que es quien está interesado en dichos eventos para reaccionar y actuar en consecuencia.

Para que un escuchador, pueda escuchar los eventos de un notificador, ha de subscribirse a estos. En caso de estar suscrito, el notificador, cuando ocurra el evento, se lo notificará al escuchador. En Java, esto significa enviar un mensaje concreto al escuchador (ejecutar un método concreto).

El mecanismo que tenemos en Java de poder garantizar que el escuchador implementa dichos métodos, es forzándole a implementar un interfaz. Y esto lo consigue mediante el API del método de subscripción, que solo permite recibir parámetros de un tipo concreto. Luego el escuchador interesado ha de ser de dicho tipo para poder subscribirse.

Un escuchador, igual que se subscribe, puede de subscribirse.

Los notificadores pasan la información a los escuchadores invocando un método concreto como ya hemos comentado, pasándole como parámetro un evento que encapsula toda la información necesaria para conocer lo que ha ocurrido.

Los eventos contienen un identificador, de tipo int, que describe el tipo de evento. También contiene información sobre el origen del evento, mediante el método getSource();.

Por tanto, el manejo de eventos se consigue mediante el uso de interfaces definidos en el paquete java.awt.event.*:

- ActionListener.
- WindowListener.
- KeyListener.
- MouseListener.
- etc...

Los eventos pueden ser físicos (informan de una acción física que ha ocurrido) o semánticos (informan de algún cambio o situación conceptual no necesariamente física).

Los eventos físicos de la AWT son:

ComponentEvent	Esconder, mover, redimensionar, mostrar.
ContainerEvent	Añadir o eliminar un componente.
FocusEvent	Obtener o perder foco.
KeyEvent	Pulsar, liberar o teclear (ambos) una tecla.
MouseEvent	Entrar, salir, pulsar, soltar o clicar (ambos).
MouseMotionEvent	Arrastrar o mover.
WindowEvent	Maximizar, minimizar, abrir, cerrar, activar o desactivar.

Los eventos semánticos de la AWT son:

ActionEvent	Una acción se ha ejecutado.
AdjustmentEvent	Un valor se ha ajustado.

ItemEvent	Un estado ha cambiado.
TextEvent	Un texto ha cambiado.

Y ¿qué componente puede notificar de qué evento? Para contestar a esta pregunta, añadimos la siguiente tabla, marcando con una X aquellas casillas cuyo componente (fila) pueda lanzar su evento (columna):

Componente AWT	Tipos de eventos que pueden generar					
	Action	Adjustment	Component	Container	Focus	Item
Button	X		X		X	
Canvas			X		X	
Checkbox			X		X	X
Choice			X		X	X
Component			X		X	
Container			X	X	X	
Dialog			X	X	X	
Frame			X	X	X	
Label			X		X	
List	X		X		X	X
MenuItem	X					
Panel			X	X	X	
Scrollbar		X	X		X	
ScrollPane			X	X	X	
TextArea			X		X	
TextField	X		X		X	
Window			X	X	X	

Componente AWT	Tipos de eventos que pueden generar				
	Key	Mouse	Mouse Motion	Text	Window
Button	X	X	X		
Canvas	X	X	X		
Checkbox	X	X	X		
Choice	X	X	X		
Component	X	X	X		
Container	X	X	X		
Dialog	X	X	X		X
Frame	X	X	X		X
Label	X	X	X		
List	X	X	X		
MenuItem					
Panel	X	X	X		
Scrollbar	X	X	X		
ScrollPane	X	X	X		
TextArea	X	X	X	X	
TextField	X	X	X	X	
Window	X	X	X		X

Ya hemos visto, que un escuchador, ha de implementar un interfaz específico dependiendo del tipo de evento que quiera escuchar.

Implementar un interfaz significa implementar todos sus métodos. Y puede haber ocasiones en los que no estemos interesados en todos. Pero para que nuestro código compile, deberemos implementar también aquellos métodos en los que no estemos interesados (habitualmente dejándolos vacíos).

Para evitar en la medida de lo posible esto, existe lo que se llama Adaptadores. Son clases que tienen definidos todos los métodos de un interface concreto.

La implementación de dichos métodos está vacía.

Heredando de un Adapter, y sobrescribiendo los métodos necesarios conseguimos el mismo resultado que implementar directamente el interface. Evidentemente esto no siempre es posible porque ya sabemos que en Java no existe herencia múltiple.

En la siguiente tabla se enumeran todos los métodos (su nombre es auto explicativo) de los distintos interfaces y los adaptadores si los hubiere:

Listener interface	Adapter class	Métodos
ActionListener		actionPerformed
AdjustmentListener		adjustmentValueChanged
ComponentListener	ComponentAdapter	componentHidden componentMoved componentResized componentShown
ContainerListener	ContainerAdapter	componentAdded componentRemoved
FocusListener	FocusAdapter	focusGained focusLost
ItemListener		itemStateChanged
KeyListener	KeyAdapter	keyPressed keyReleased keyTyped
MouseListener	MouseAdapter	mouseClicked mouseEntered mouseExited mousePressed mouseReleased
MouseMotionListener	MouseMotionAdapter	mouseDragged mouseMoved
TextListener		textValueChanged
WindowListener	WindowAdapter	windowActivated windowClosed

		windowClosing windowDeactivated windowDeiconified windowIconified windowOpened
--	--	--

A continuación, veremos varios ejemplos que mediante distintas posibilidades escuchan el evento de pulsación del botón X de una ventana y terminan la aplicación.

- **Ejemplo:** Gestión de eventos donde el escuchador es una clase distinta a la clase visual:

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class WindowListenerTest extends WindowAdapter
{
    public void windowClosing(WindowEvent ev)
    {
        System.exit(0);
    }
}
import java.awt.Frame;
public class WindowEventTest1
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("FrameTest");
        f.setSize(200,150);
        f.addWindowListener(new WindowListenerTest());
        f.setVisible(true);
    }
}
```

- **Ejemplo:** Gestión de eventos donde el escuchador es la propia clase visual:

```
import java.awt.*;
import java.awt.event.*;
public class WindowEventTest2 implements WindowListener
{
    public static void main(String[] args)
    {
```

```

    WindowEventTest2 w = new WindowEventTest2();
}
public WindowEventTest2()
{
    Frame f = new Frame();
    f.setTitle("FrameTest");
    f.setSize(200,150);
    f.addWindowListener(this);
    f.setVisible(true);
}
public void windowActivated(WindowEvent ev) {}
public void windowClosed(WindowEvent ev) {}
public void windowClosing(WindowEvent ev) { System.exit(0); }
public void windowDeactivated(WindowEvent ev) {}
public void windowDeiconified(WindowEvent ev) {}
public void windowIconified(WindowEvent ev) {}
public void windowOpened(WindowEvent ev) {}
}

```

- **Ejemplo:** Gestión de eventos donde el escuchador es una clase anónima (Nota. Las Clases Anónimas no se tratan en este curso, pero añadimos un ejemplo de su uso para que no le sorprenda al lector si ve algún ejemplo de su uso en otros sitios):

```

import java.awt.*;
import java.awt.event.*;
public class WindowEventTest3
{
    public static void main(String[] args)
    {
        WindowEventTest3 w = new WindowEventTest3();
    }
    public WindowEventTest3()
    {
        Frame f = new Frame();
        f.setTitle("FrameTest");
        f.setSize(200,150);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent ev)
            {
                System.exit(0);
            }
        }
    }
}

```

```

    }
    );
    f.setVisible(true);
}
}

```

Resumiendo, lo que ocurre en ambos tres ejemplos es:

1. Cuando invocamos el método `addWindowListener`, estamos estableciendo un 'callback'.
 2. Como parámetro se manda un escuchador, el cual debe implementar el interfaz correspondiente.
 3. Cuando se genera un `WindowEvent` como consecuencia de pulsar el botón con la X, el método `windowClosing()` del escuchador es invocado.
 4. Se pueden añadir varios escuchadores a un mismo notificador de eventos.
- **Ejemplo:** Un nuevo ejemplo de gestión de eventos:

```

import java.awt.*;
public class ButtonEventTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("ButtonEventTest");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        Button b1 = new Button("Aceptar");
        b1.addActionListener(new ActionListenerTest());
        f.add(b1);
        Button b2 = new Button("Cancelar");
        b2.addActionListener(new ActionListenerTest());
        f.add(b2);
        f.setVisible(true);
    }
}

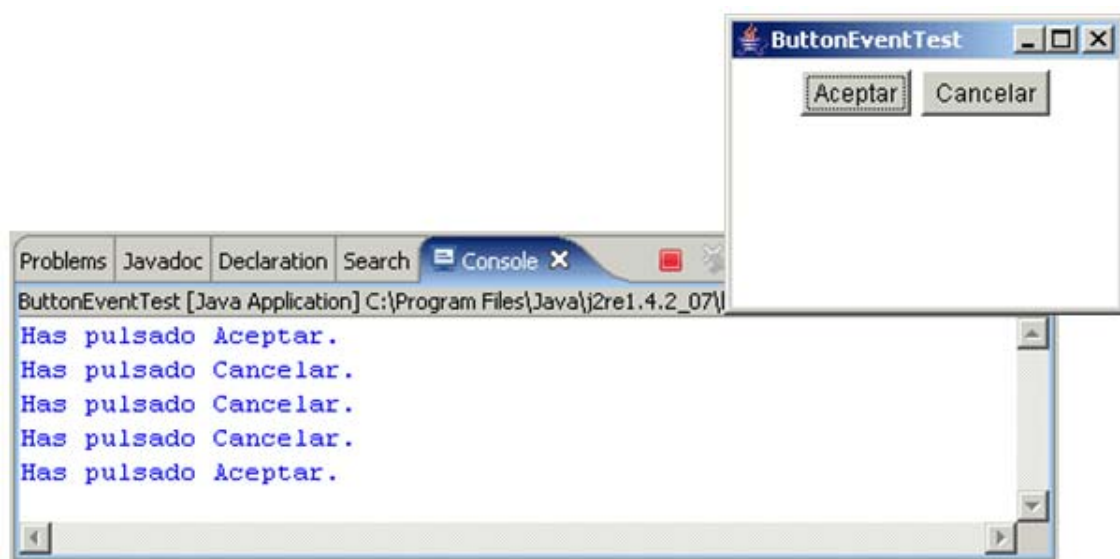
import java.awt.Button;
import java.awt.event.ActionEvent;

```



```
import java.awt.event.ActionListener;

public class ActionListenerTest implements ActionListener
{
    public void actionPerformed(ActionEvent ev)
    {
        if(((Button)ev.getSource()).getLabel().equals("Aceptar"))
            System.out.println("Has pulsado Aceptar.");
        else
            System.out.println("Has pulsado Cancelar.");
    }
}
```



5. Otras clases

En este último apartado, comentaremos algunas otras clases que nos ayudarán a completar y perfeccionar las aplicaciones visuales.

5.1 java.awt.Color

Implementa un color descrito según el RGB.

RGB: Red-Green-Blue es un sistema de definición de colores, donde se especifica el nivel de saturación de cada uno de esos tres colores mediante valores entre 0 y 255.

Se puede construir mediante un valor RGB:

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

O utilizar colores predefinidos mediante constantes:

```
Color amarillo = Color.YELLOW;
```

Soporta transparencias (alpha) mediante un valor entre 0.0 y 1.0

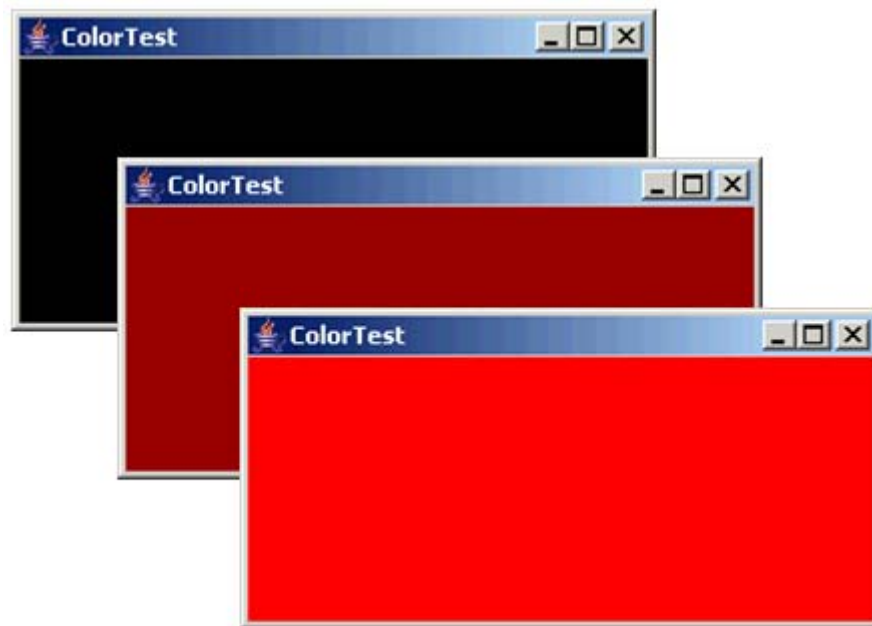
- **Ejemplo:** Uso de la clase `java.awt.Color`:

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
public class ColorTest
{
    Frame f = null;

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

    public Test()
    {
        f = new Frame();
        f.setTitle("ColorTest");
        f.setSize(300,150);
        f.setBackground(Color.BLACK);
        f.setVisible(true);
        f.addMouseMotionListener(new MouseMotionAdapter()
        {
            public void mouseMoved(MouseEvent ev)
            {
                int r = f.getBackground().getRed();
                if(r < 255)
                    f.setBackground(new Color(r+1,0,0));
                else
                    f.setBackground(Color.BLACK);
            }
        });
    }
}
```

```
}  
}
```



5.2 java.awt.Font

Implementa la representación gráfica del tipo de letra.

Se define mediante:

- Familia: nombre del tipo de letra.
- Estilo: normal o negrita y/o cursiva.
- Tamaño: pixels del punto utilizado para pintar el tipo de letra.

Existen dos clases de nombres de familia:

- Lógico: Serif, SansSerif, Monospaced, Dialog y DialogInput. La JVM se encarga de mapear el nombre lógico con un nombre físico.
- Físico: cualquier familia instalada en el sistema.

Para definir el estilo se utilizan estas constantes:

- Font.PLAIN: normal.
- Font.BOLD: negrita.
- Font.ITALIC: cursiva.

Para combinar varios estilos se utiliza el OR lógico a nivel de bit. Por ejemplo:

Font.BOLD | Font.ITALIC

- **Ejemplo:** Uso de la clase java.awt.Font:

```
import java.awt.*;
public class FontTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("FontTest");
        f.setSize(200,200);
        f.setLayout(new FlowLayout());
        Label l1 = new Label("Serif");
        l1.setFont(new Font("Serif", Font.PLAIN, 20));
        Label l2 = new Label("SansSerif");
        l2.setFont(new Font("SansSerif", Font.PLAIN, 20));
        Label l3 = new Label("Monospaced");
        l3.setFont(new Font("Monospaced", Font.ITALIC, 20));
        Label l4 = new Label("Dialog");
        l4.setFont(new Font("Dialog", Font.BOLD,20));
        Label l5 = new Label("DialogInput");
        l5.setFont(new Font("DialogInput", Font.BOLD | Font.ITALIC, 20));
        f.add(l1); f.add(l2); f.add(l3); f.add(l4); f.add(l5);
        f.setVisible(true);
    }
}
```



5.3 java.awt.Cursor

Implementa la representación gráfica del cursor.

Existen varios predefinidos, que se pueden usar mediante el uso de constantes. Algunos cursores:

- Cruz: `Cursor.CROSSHAIR_CURSOR`
- Mano: `Cursor.HAND_CURSOR`
- Mover: `Cursor.MOVE_CURSOR`
- Texto: `Cursor.TEXT_CURSOR`
- Espera: `Cursor.WAIT_CURSOR`

La clase `java.awt.Component` tiene el método:

```
public void setCursor(Cursor cursor);
```

- **Ejemplo:** Uso de la clase `java.awt.Cursor`:

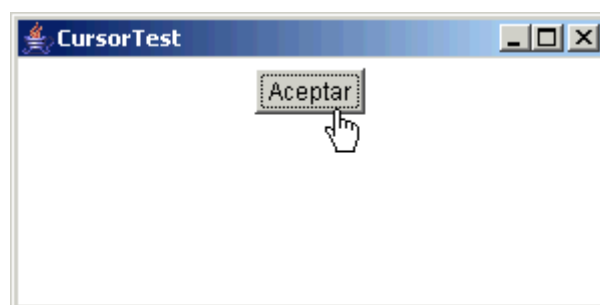
```
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
public class CursorTest
{
    Frame f = null;
```

```

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

public Test()
{
    f = new Frame();
    f.setTitle("CursorTest");
    f.setSize(300,150);
    f.setLayout(new FlowLayout());
    Button b1 = new Button("Aceptar");
    b1.addMouseListener(new MouseAdapter()
    {
        public void mouseEntered(MouseEvent ev)
        {
            f.setCursor(Cursor.HAND_CURSOR);
        }
        public void mouseExited(MouseEvent ev)
        {
            f.setCursor(Cursor.DEFAULT_CURSOR);
        }
    });
    f.add(b1);
    f.setVisible(true);
}
}

```



5.4 java.awt.Graphics

Sirve para dibujar (se suele usar con los Canvas).

Todos los componentes AWT son dibujados en pantalla a través del método de la clase `java.awt.Component`:

```
public void paint(Graphics g);
```

La clase `Graphics` permite:

- Trabajar con primitivas gráficas: figuras, colores, textos, ...
- Trabajar con imágenes: GIF y JPEG

Un componente puede pedir que sea repintado mediante el método:

```
public void repaint();
```

Posibilidades para dibujar figuras:

- Líneas: `drawLine()`.
- Rectángulos: `drawRect()`, `fillRect()`, `clearRect()`.
- Rectángulos 3D: `draw3DRect()`, `fill3DRect()`.
- Rectángulos redondeados: `drawRoundRect()`, `fillRoundRect()`.
- Óvalos: `drawOval()`, `fillOval()`.
- Arcos: `drawArc()`, `fillArc()`.
- Polígonos: `drawPolygon()`, `fillPolygon()`.

Escribiendo texto:

- `drawBytes()`, `drawChars()`, `drawString()`.
- **Ejemplo:** Uso de la clase `java.awt.Graphics`:

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

```

public class GraphicsTest extends Frame
{
    public static void main(String[] args)
    {
        new Test().setVisible(true);
    }

    public Test()
    {
        this.setTitle("GraphicsTest");
        this.setBackground(Color.LIGHT_GRAY);
        this.setSize(300,150);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.LIGHT_GRAY);
        g.draw3DRect(10,30,this.getWidth()-20,this.getHeight()-40,true);
        g.setColor(Color.BLACK);
        g.drawLine(15,35,65,65);
        g.drawRect(70,35,50,30);
        g.drawRoundRect(125,35,50,30,10,10);
        g.drawOval(180,35,50,30);
        g.drawArc(235,35,50,30,25,200);
        int[] x = { 15,65,15,65};
        int[] y = { 90,90,120,120};
        g.drawPolygon(x,y,x.length);
        g.setColor(Color.RED);
        g.fillRect(70,90,50,30);
        g.fillRoundRect(125,90,50,30,10,10);
        g.fillOval(180,90,50,30);
        g.fillArc(235,90,50,30,25,200);
        g.setColor(Color.BLACK);
        g.setFont(new Font("SansSerif",Font.PLAIN,9));
        g.drawString("Línea",30,80);
        g.drawString("Rectángulos",95,80);
        g.drawString("Óvalo",192,80);
        g.drawString("Arco",250,80);
        g.drawString("Polígono",22,135);
        g.drawString("Rectángulos",95,135);
        g.drawString("Óvalo",192,135);
        g.drawString("Arco",250,135);
    }
}

```



```

    }
}

```



Esta clase también permite trabajar con imágenes, representadas por la clase `java.awt.Image`

Soporta los formatos: GIF y JPEG.

Para cargar una imagen se utiliza la clase `java.awt.Toolkit`:

```
Toolkit.getDefaultToolkit().getImage(...);
```

Para pintar una imagen:

```
drawImage();
```

- **Ejemplo:** Uso de la clase `java.awt.Graphics`:

```

import java.awt.*;

public class ImageTest extends Frame
{
    public static void main(String[] args)
    {
        new Test().setVisible(true);
    }

    public Test()
    {
        this.setTitle("ImageTest");
        this.setSize(150,110);
    }
}

```

```
public void paint(Graphics g)
{
    Image img = Toolkit.getDefaultToolkit().getImage("duke.gif");
    g.drawImage(img,45,25,this);
}
}
```



6. Editor visual AWT/Swing en Eclipse

La creación de interfaces visuales, es un trabajo bastante repetitivo, que no tiene sentido hacerlo a mano. Los profesionales del sector, suelen hacer uso de herramientas visuales para "pintar" más que codificar el diseño de dichos interfaces visuales.

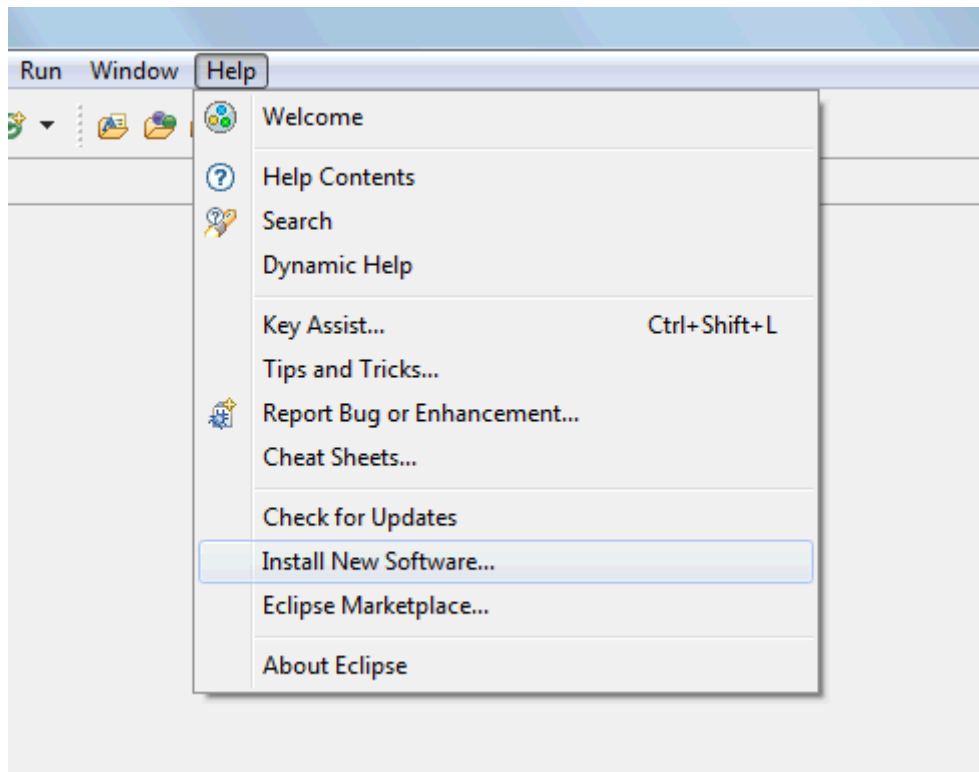
Nota: No obstante, desde un punto de vista meramente didáctico, recomendamos al comienzo desarrollar los interfaces visuales a mano para de verdad entender y cimentar correctamente los conocimientos. Una vez el lector se sienta cómodo y seguro con este tipo de desarrollo puede pasar al uso de estas herramientas para ser más eficientes y productivos. Es importante tener presente que durante el examen de certificación de este curso, no se contará con editores visuales.

La comunidad Eclipse.org, creadora del IDE que venimos utilizando en este curso, tiene un proyecto encargado del desarrollo y mantenimiento de un Editor Visual para el desarrollo de AWT (entre otros frameworks de programación visual): <http://www.eclipse.org/windowbuilder/>

A continuación, comentaremos los pasos para su instalación y uso.

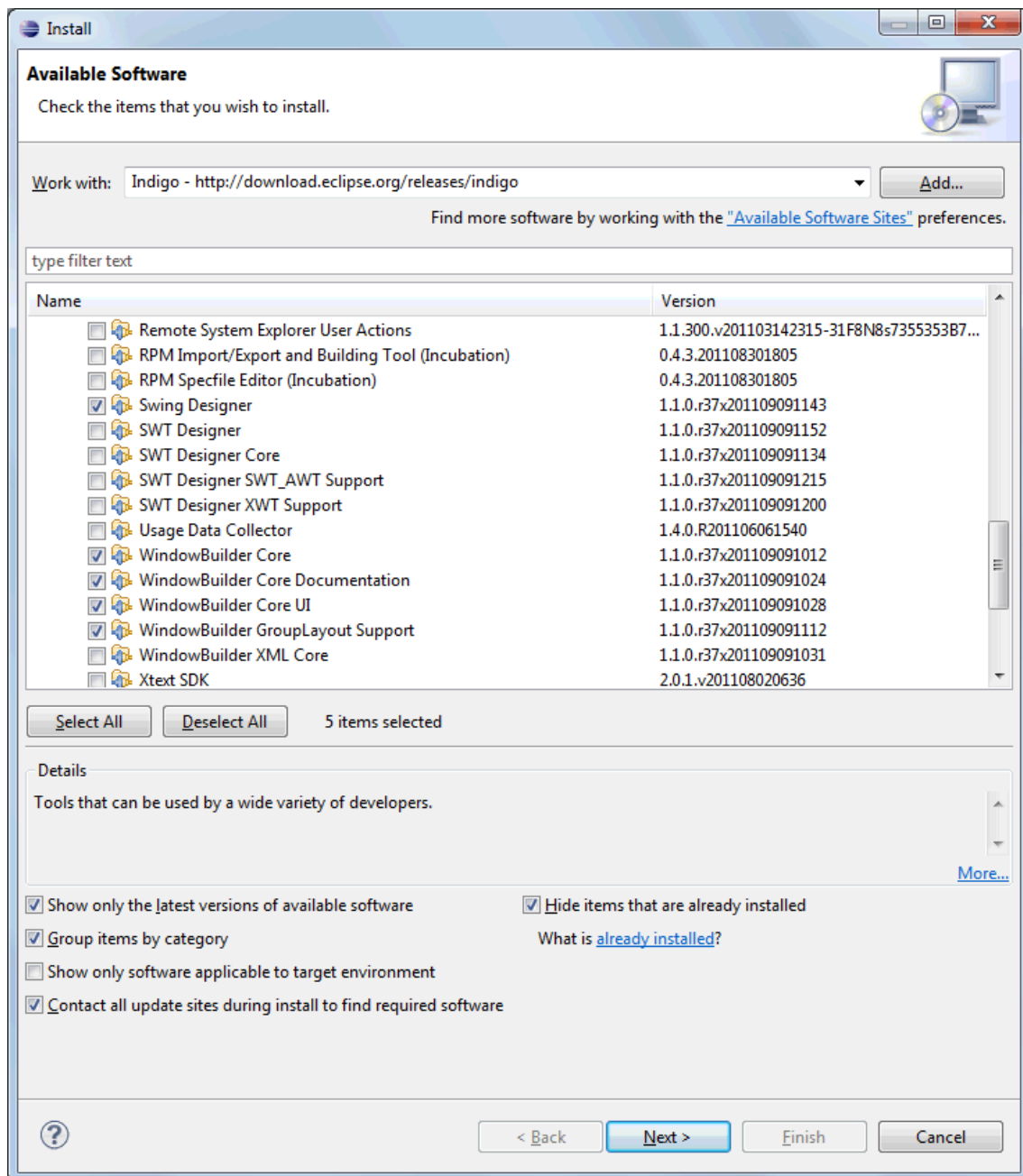
Nota: Dependiendo de la edición de Eclipse que se esté utilizando, puede que Window Builder ya esté incluido. Por ejemplo, la edición "for Java Developers" lo incluye por defecto.

Para realizar la instalación, utilizaremos el "Update Manager" de Eclipse mediante la opción de menú "Install New Software...":

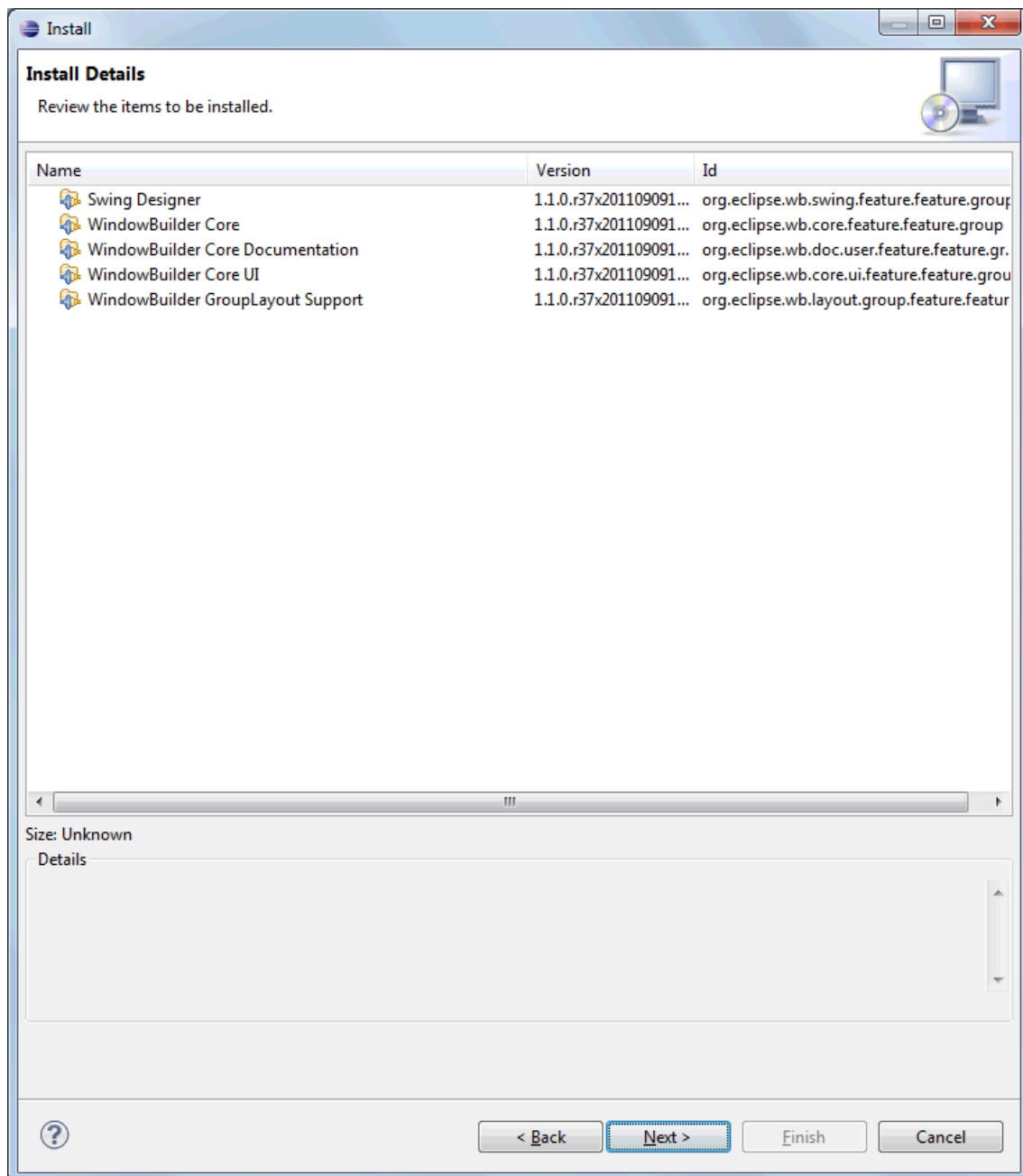


Seleccionar el site "Indigo – <http://download.eclipse.org/releases/indigo>". Y seleccionar las siguientes "features" y pulsar el botón "Next >":

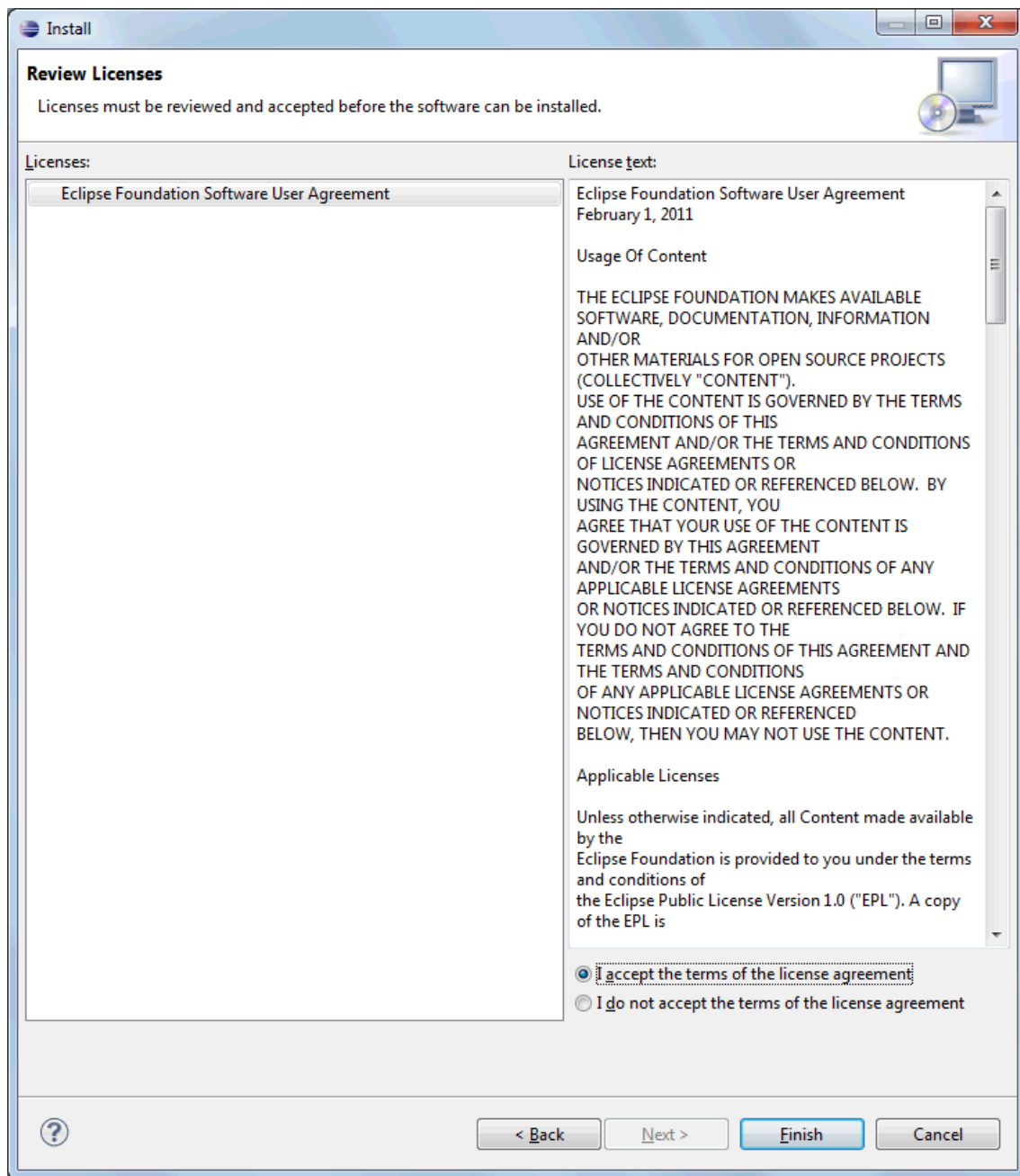
- Swing Designer.
- WindowBuilder Core.
- WindowBuilder Core Documentation.
- WindowBuilder Core UI.
- WindowBuilder GroupLayout Support.



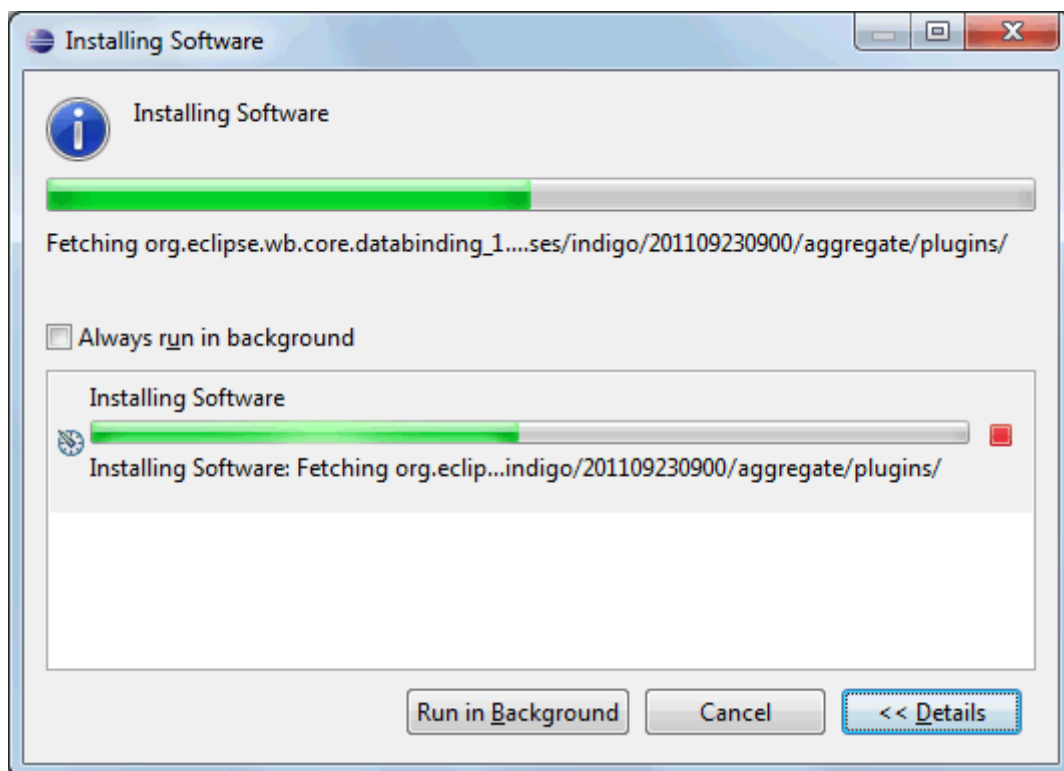
Aceptar los detalles pulsando el botón "Next >":



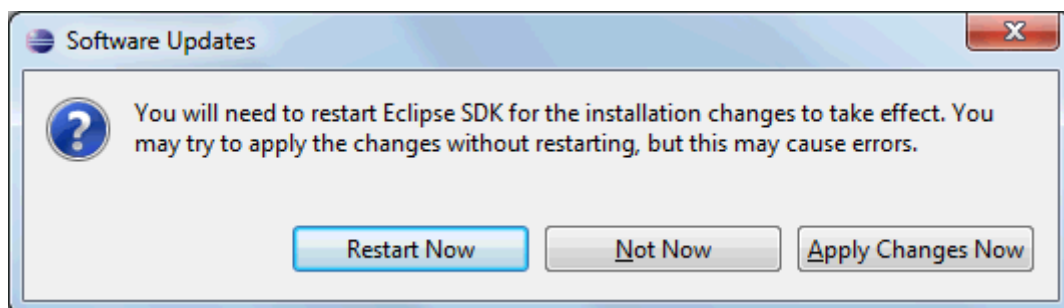
Aceptar los términos de la licencia pulsando el botón "Finish":



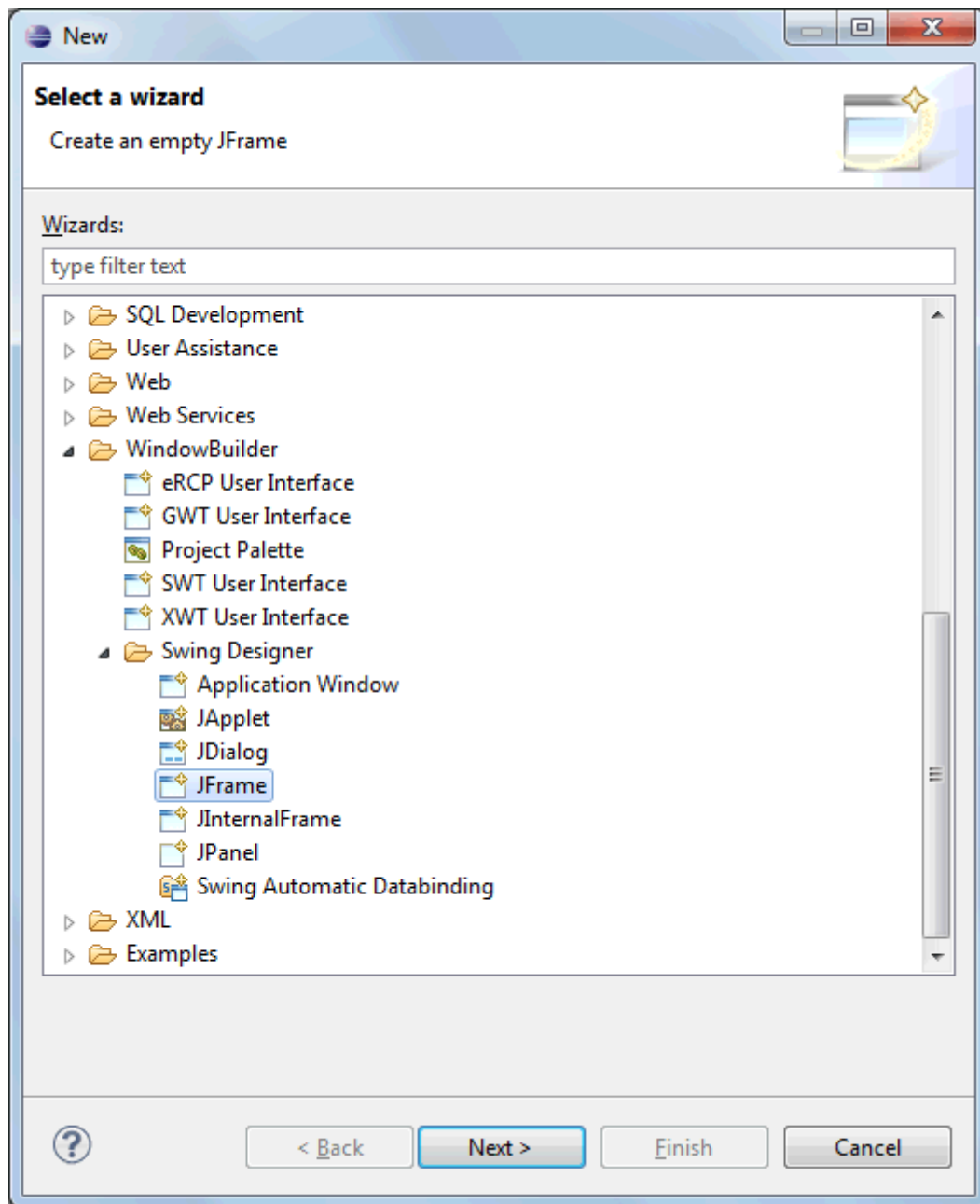
Eclipse comenzará a descargar el código al disco local y posteriormente a instalarlo:



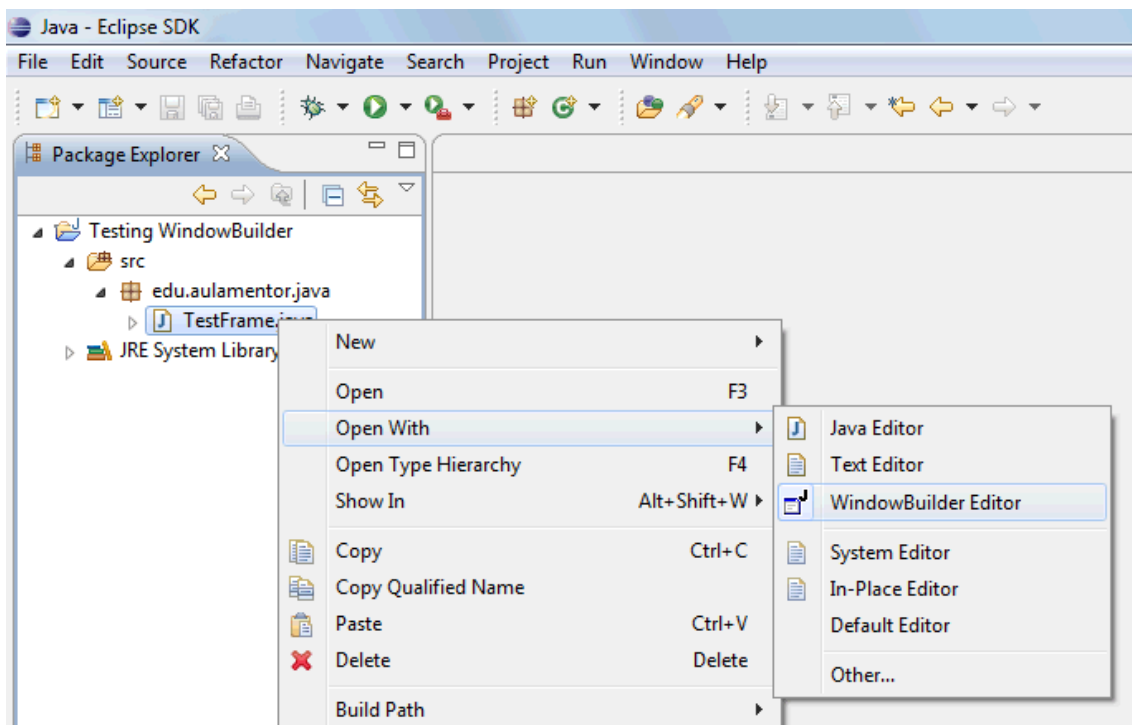
Por último, nos pedirá rearrancar Eclipse.



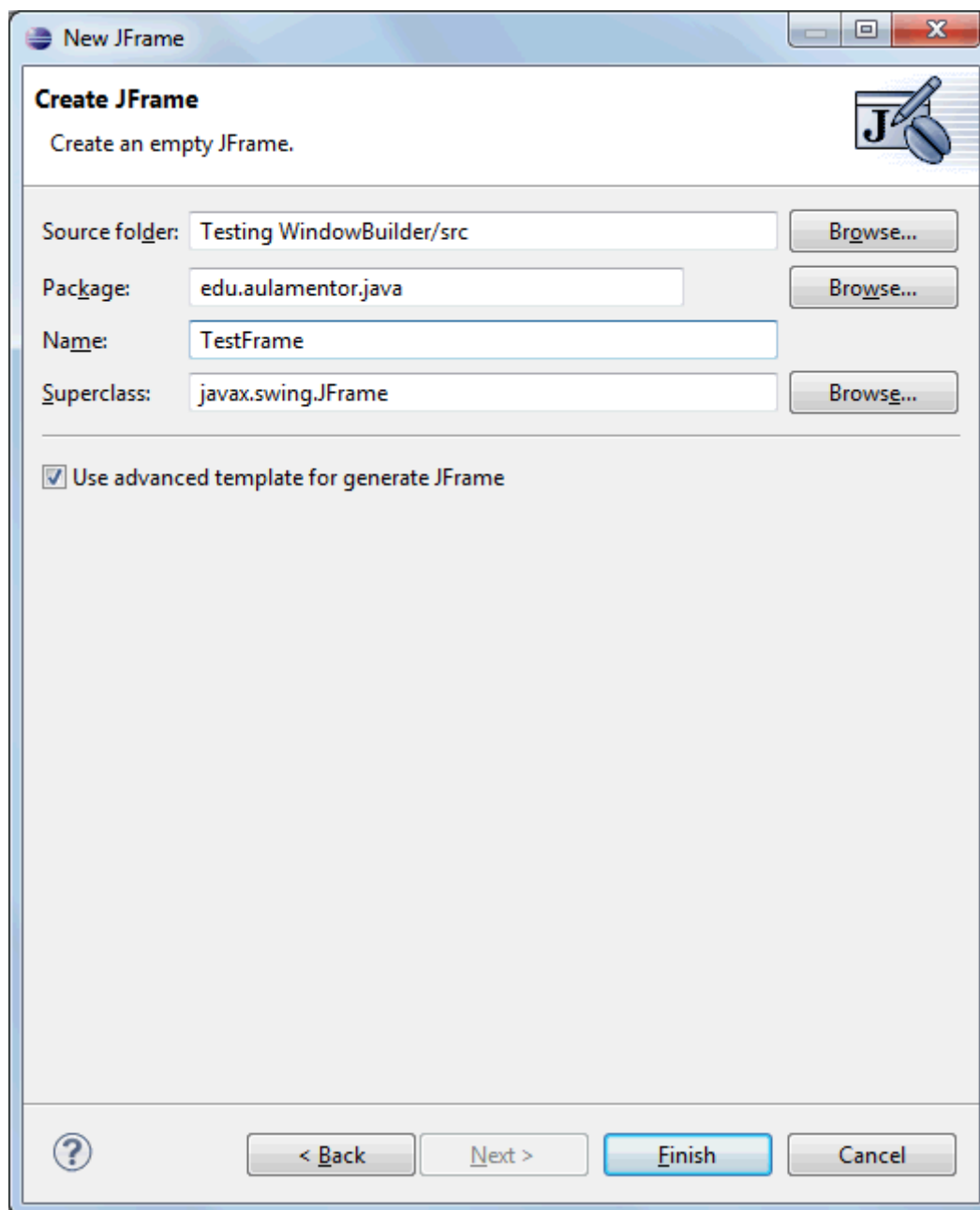
Para crear una clase visual nueva, podemos utilizar el nuevo asistente:



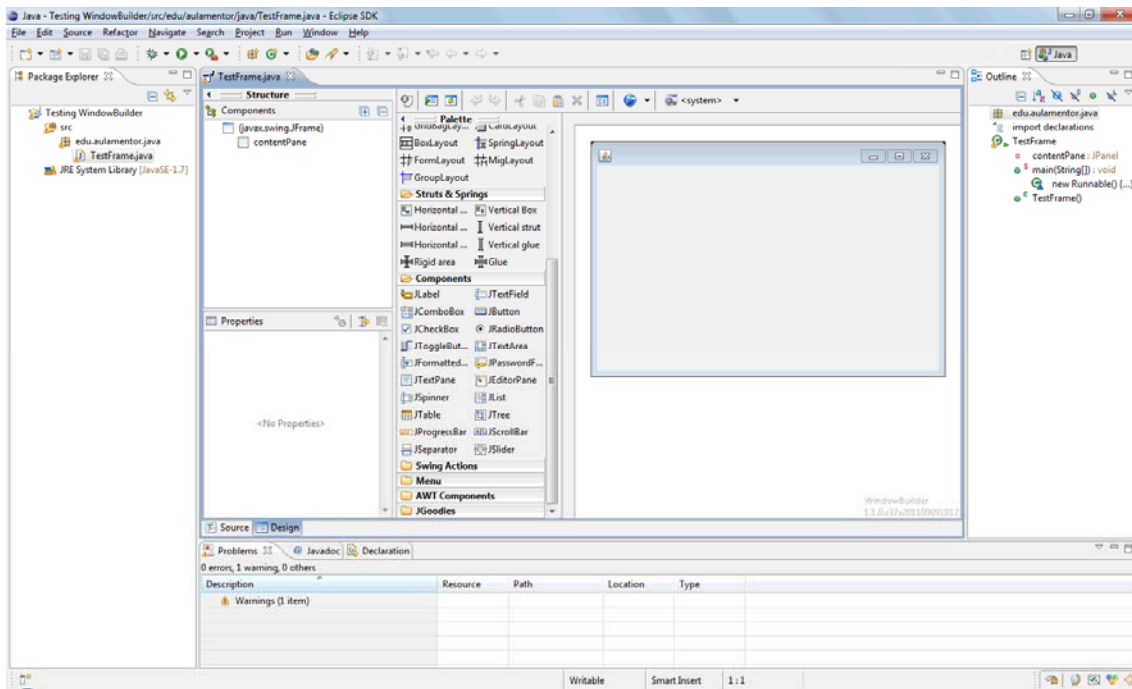
o abrir una ya existente con la nueva opción del menú de contexto “WindowBuilder Editor”:



La apariencia de dicho asistente nuevo es esta:



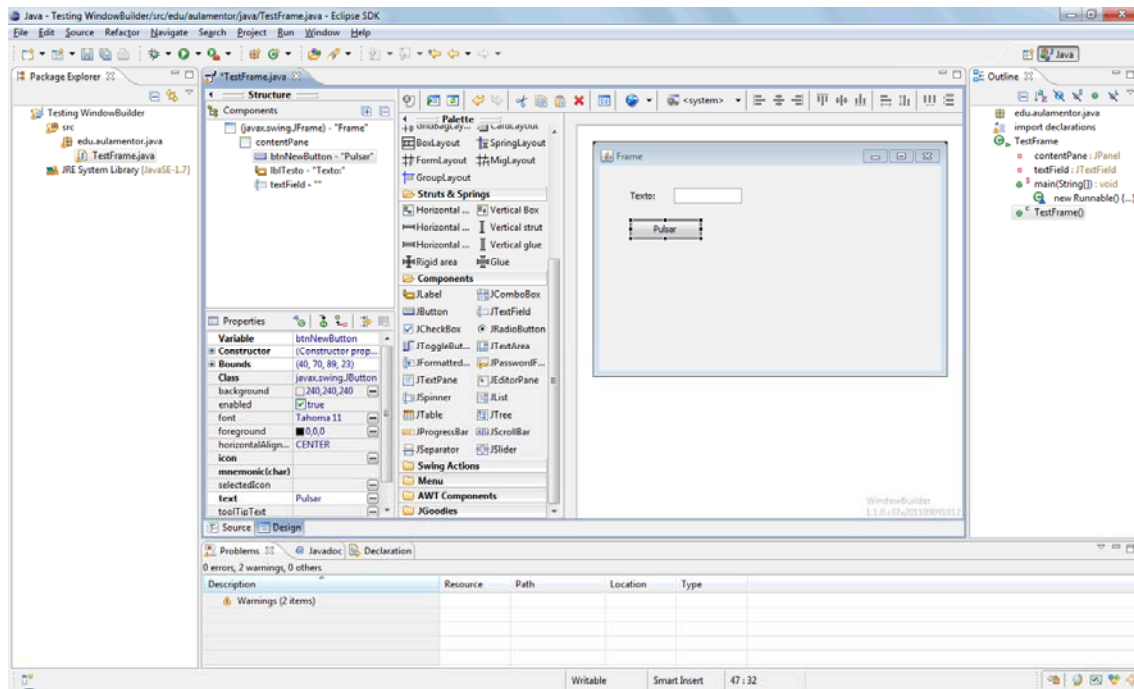
La apariencia del nuevo editor visual es esta:



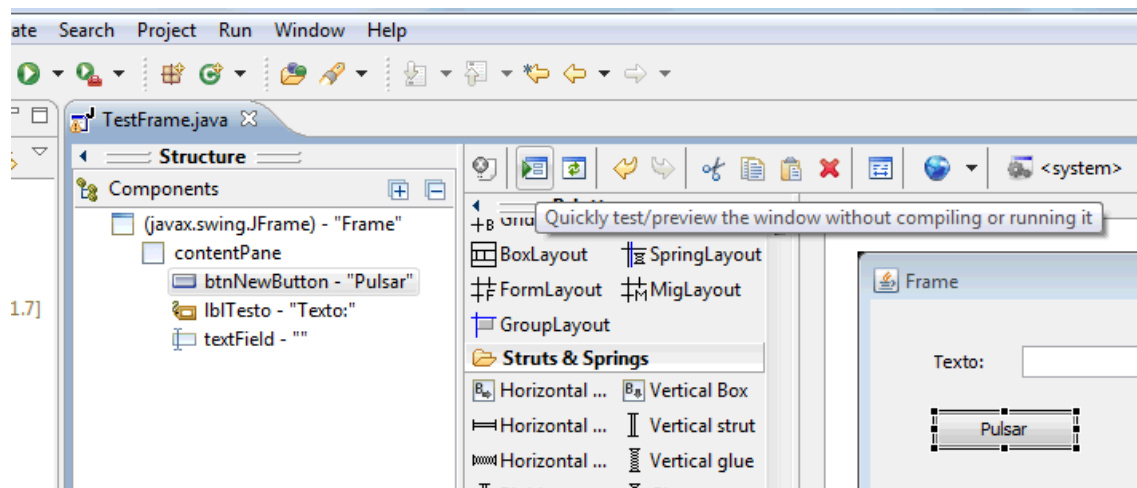
Como se puede observar en el pantallazo, existe una barra de herramientas con todos los componentes visuales que se pueden arrastrar y soltar con el ratón en el editor. Existe una vista del código Java que se va generando, donde también se puede escribir código directamente y ver su reflejo automáticamente en el editor visual. Por último, existe una vista con las propiedades de todos los componentes utilizados donde se pueden ir añadiendo o modificando sus valores y ver automáticamente su reflejo tanto en el editor visual como en el de código Java.

Para moverse del editor visual al de código fuente, utilizar las dos pestañas que hay debajo: "Source" y "Design".

Ejemplo de un desarrollo a medias con este editor nuevo:



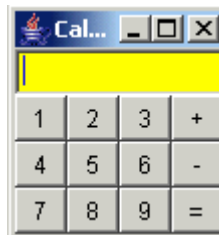
Para poder probar el código visual sin la necesidad de desarrollar la aplicación completa, o un método main para instanciar y mostrar la clase, existe una opción de ejecución nueva en la barra de herramientas del editor:





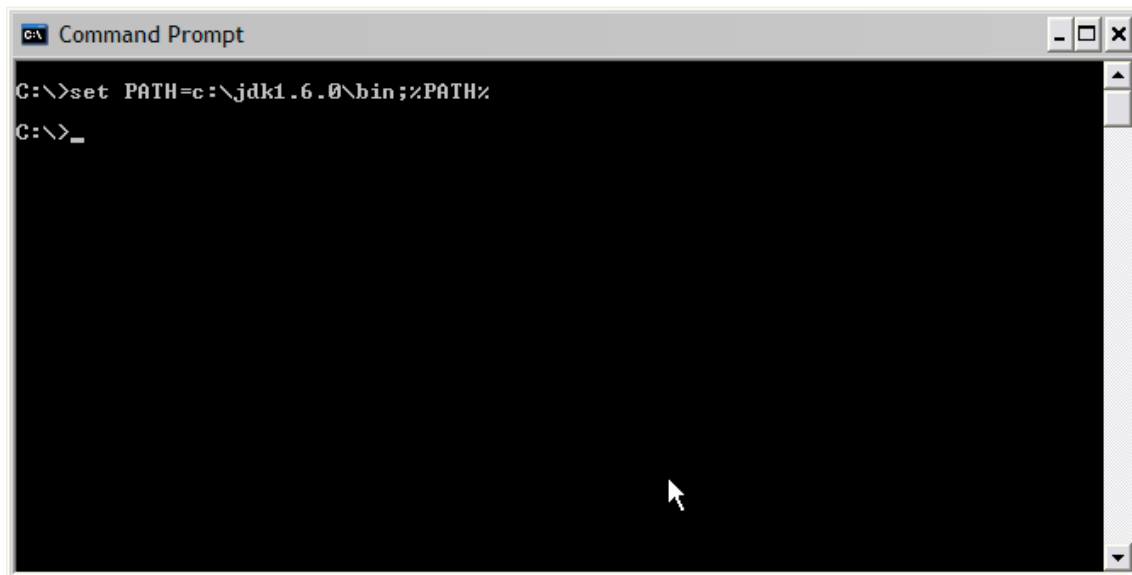
para practicar

PRÁCTICA A: Desarrollar, tanto con la JDK de Oracle como con Eclipse, solamente el interfaz visual de una calculadora mediante la AWT. El resultado debe ser algo parecido a:

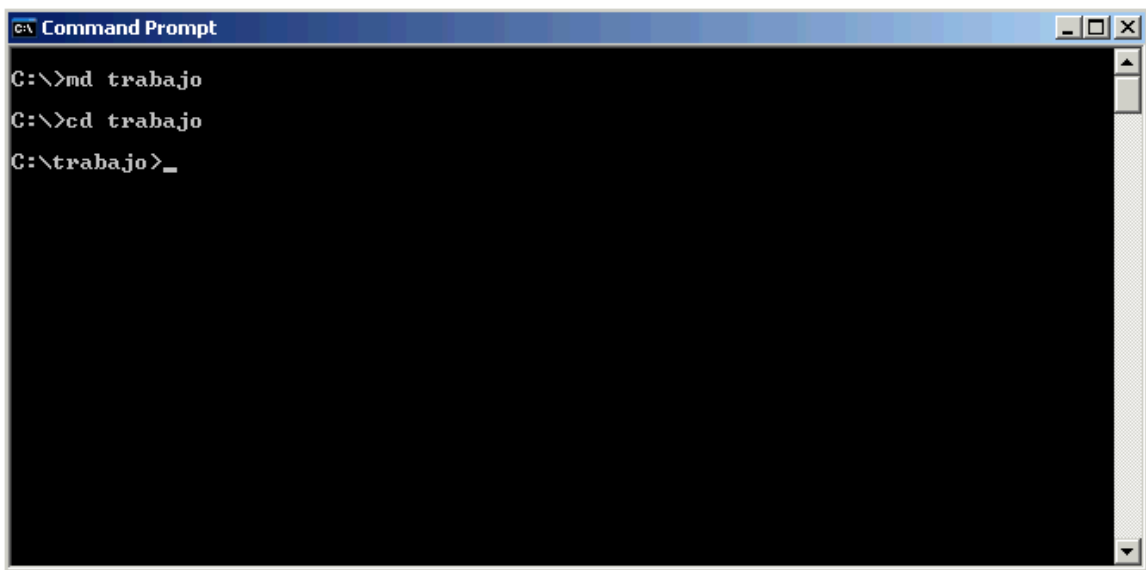


Solución con la JDK de Oracle:

En el caso de que no esté puesta la variable de entorno PATH, abrir una sesión DOS y ajustar la variable de entorno PATH para que el Sistema Operativo sepa encontrar las herramientas del JDK. Para respetar el valor que ya tuviese la variable PATH le añadimos %PATH%.



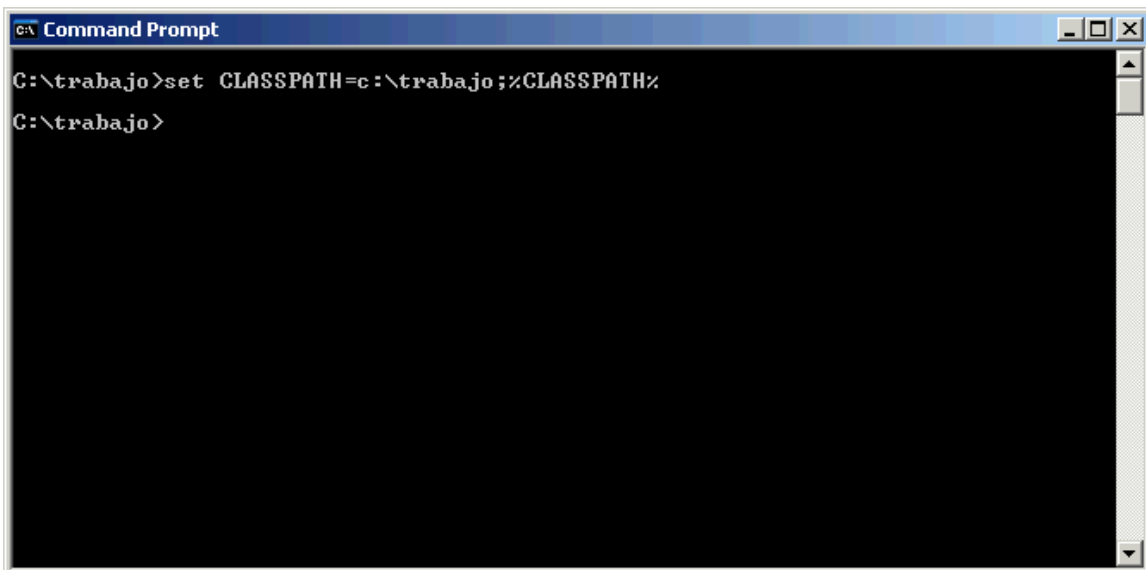
Creamos un directorio de trabajo donde guardar el programa Java.



```
C:\>md trabajo
C:\>cd trabajo
C:\trabajo>_
```

A screenshot of a Windows Command Prompt window. The title bar reads 'C:\ Command Prompt'. The command history shows: 'C:\>md trabajo', 'C:\>cd trabajo', and 'C:\trabajo>_' with a cursor at the end of the last line.

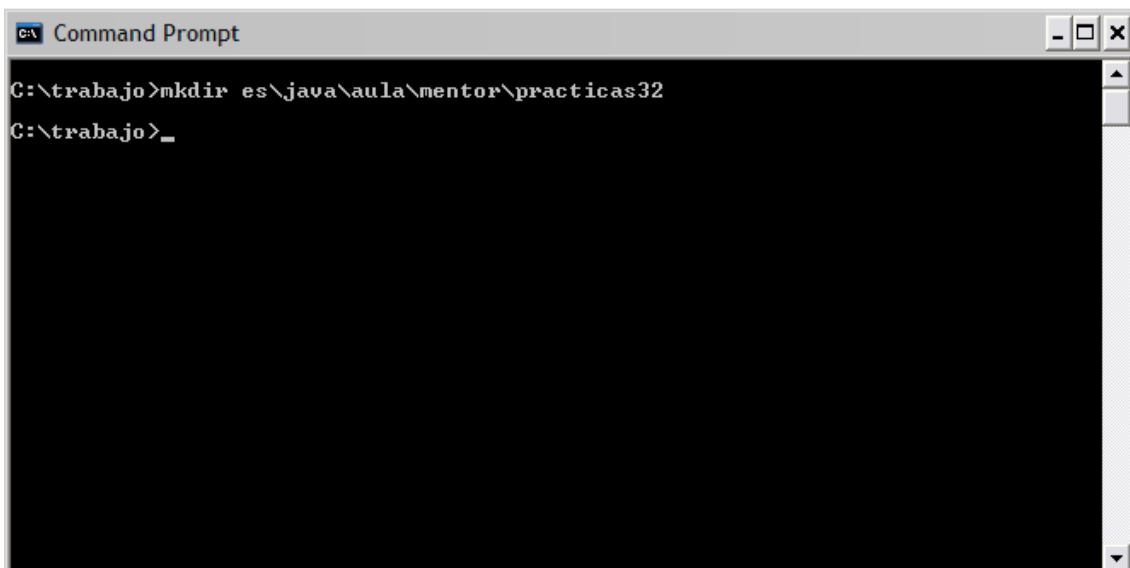
Ajustar la variable de entorno CLASSPATH para que las herramientas del JDK sepan encontrar nuestras clases Java. Tenemos dos opciones, o añadir el . (punto) y siempre ejecutar las herramientas en el directorio donde se encuentre el código, o añadir el directorio de trabajo y ejecutar las herramientas donde queramos. Para respetar el valor que ya tuviese la variable CLASSPATH le añadimos %CLASSPATH%



```
C:\trabajo>set CLASSPATH=c:\trabajo;%CLASSPATH%
C:\trabajo>
```

A screenshot of a Windows Command Prompt window. The title bar reads 'C:\ Command Prompt'. The command history shows: 'C:\trabajo>set CLASSPATH=c:\trabajo;%CLASSPATH%' and 'C:\trabajo>'.

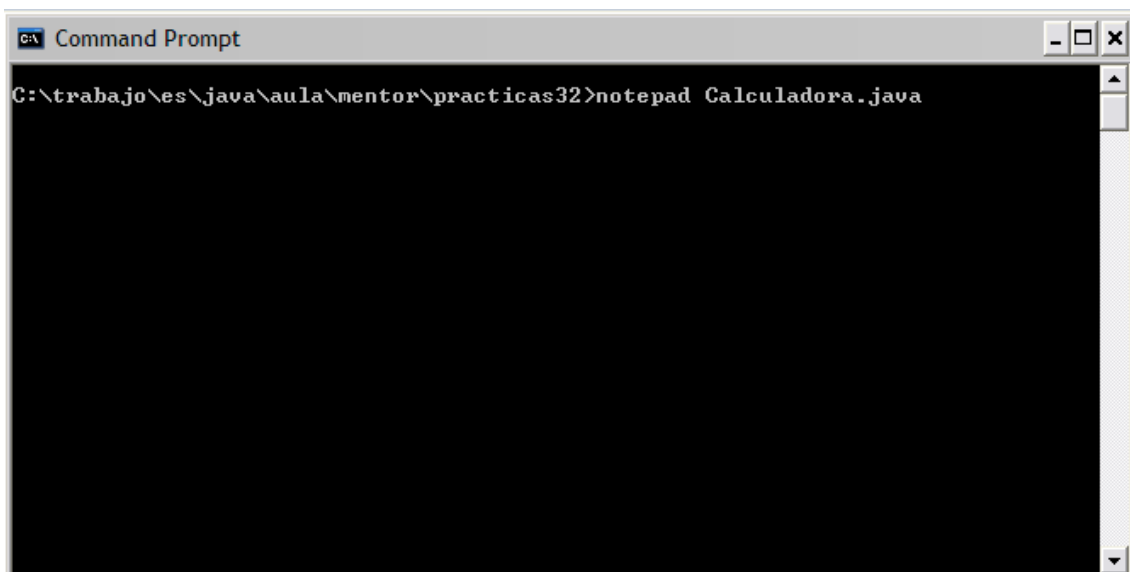
Creamos el directorio donde va a estar nuestra clase:
es\java\aula\mentor\practicass32



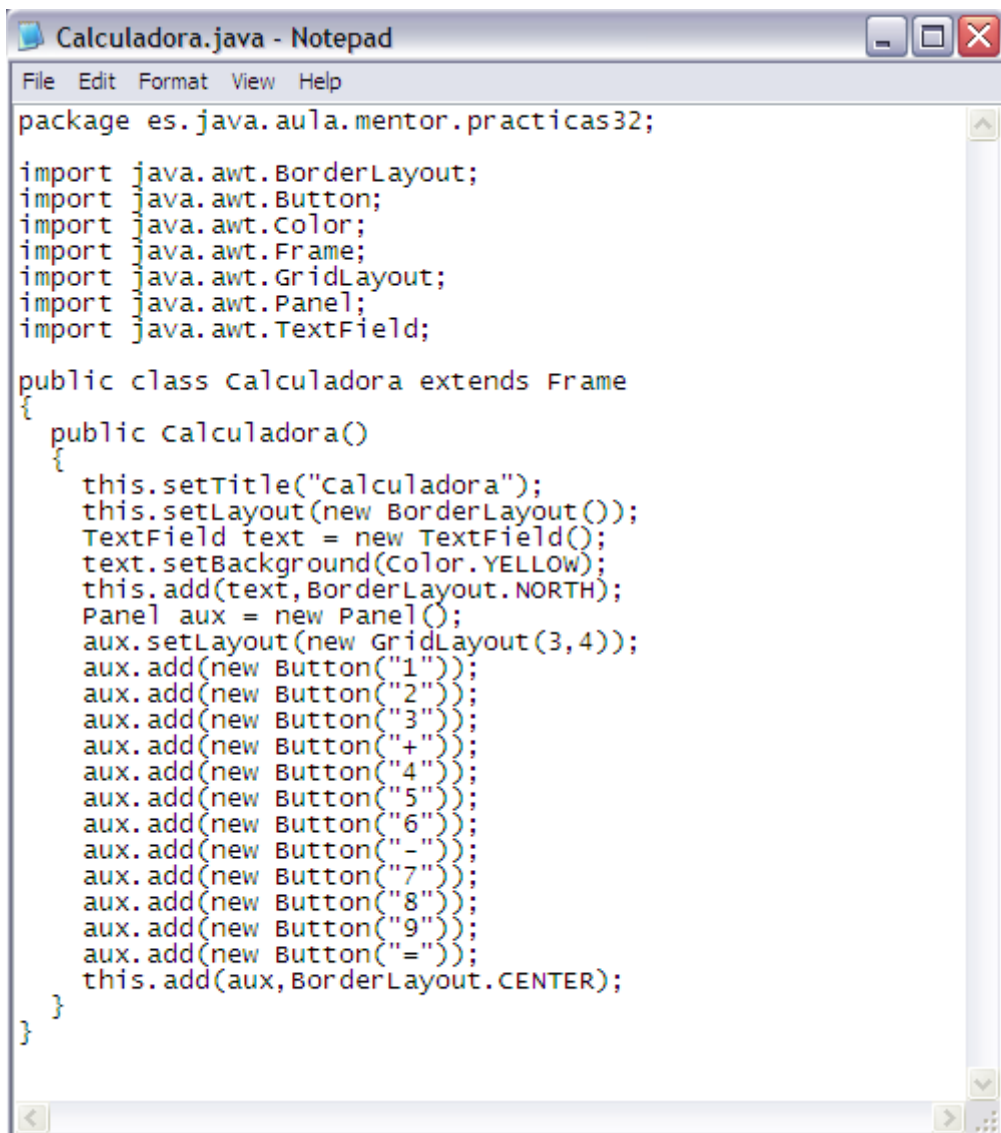
```
C:\> Command Prompt
C:\trabajo>mkdir es\java\aula\mentor\practicass32
C:\trabajo>_
```

Desde el directorio `es\java\aula\mentor\practicass32`, con un editor de texto (por ejemplo Notepad) vamos a escribir el código fuente de nuestra clase java; el nombre del fichero de la clase debe ser exactamente igual (incluyendo mayúsculas y minúsculas) al de la clase Java que vamos a desarrollar.

Creamos la clase `Calculadora.java`



```
C:\> Command Prompt
C:\trabajo\es\java\aula\mentor\practicass32>notepad Calculadora.java
```



```

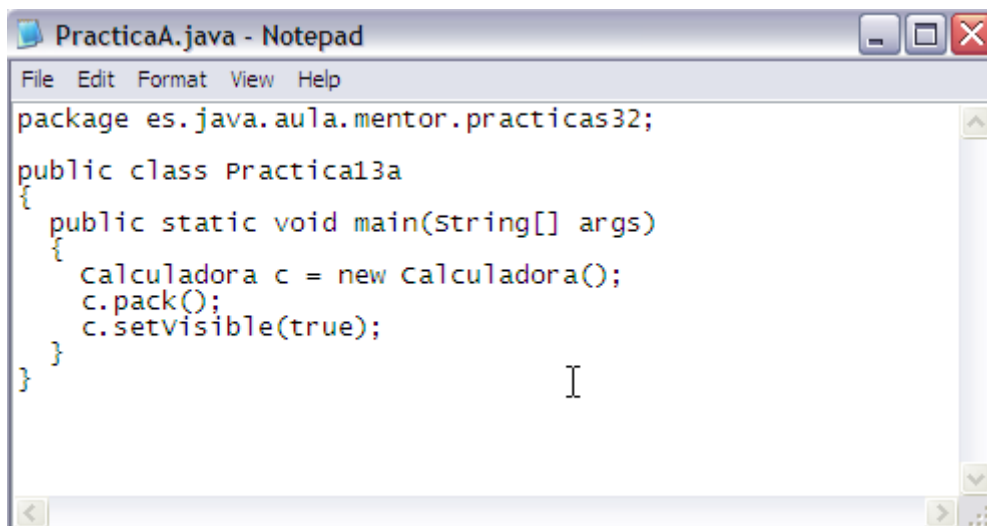
Calculadora.java - Notepad
File Edit Format View Help
package es.java.aula.mentor.practicas32;

import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Color;
import java.awt.Frame;
import java.awt.GridLayout;
import java.awt.Panel;
import java.awt.TextField;

public class Calculadora extends Frame
{
    public calculadora()
    {
        this.setTitle("calculadora");
        this.setLayout(new BorderLayout());
        TextField text = new TextField();
        text.setBackground(Color.YELLOW);
        this.add(text, BorderLayout.NORTH);
        Panel aux = new Panel();
        aux.setLayout(new GridLayout(3,4));
        aux.add(new Button("1"));
        aux.add(new Button("2"));
        aux.add(new Button("3"));
        aux.add(new Button("+"));
        aux.add(new Button("4"));
        aux.add(new Button("5"));
        aux.add(new Button("6"));
        aux.add(new Button("-"));
        aux.add(new Button("7"));
        aux.add(new Button("8"));
        aux.add(new Button("9"));
        aux.add(new Button("="));
        this.add(aux, BorderLayout.CENTER);
    }
}

```

Creamos ahora la clase PracticaA que contendrá el método main:



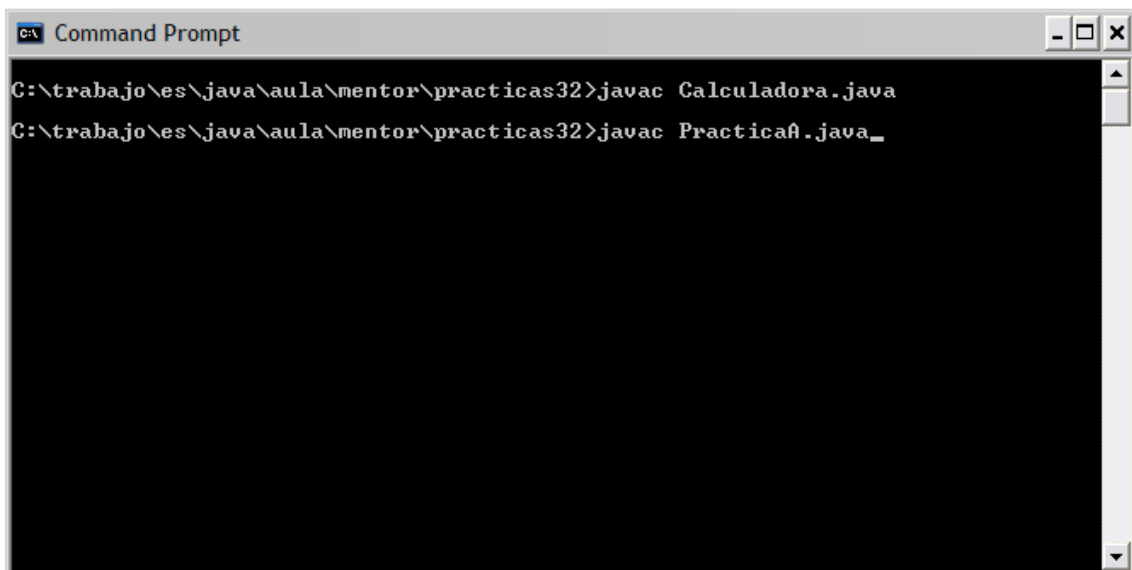
```

PracticaA.java - Notepad
File Edit Format View Help
package es.java.aula.mentor.practicas32;

public class Practica13a
{
    public static void main(String[] args)
    {
        calculadora c = new calculadora();
        c.pack();
        c.setVisible(true);
    }
}

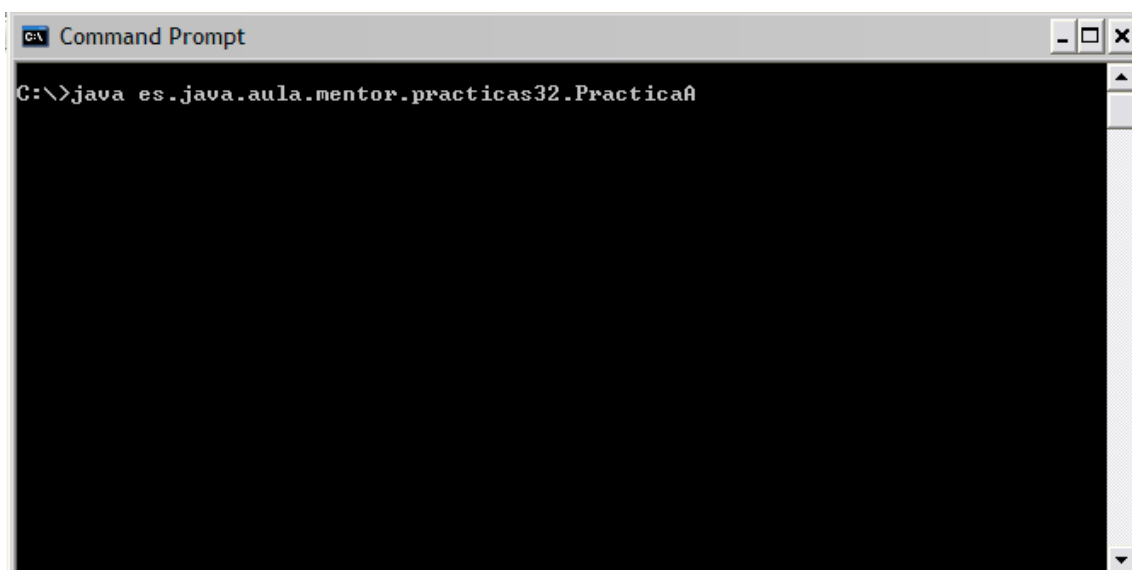
```


Compilamos las clases en el orden siguiente:



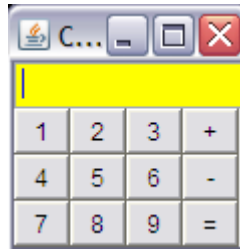
```
Command Prompt
C:\trabajo\es\java\aula\mentor\practicass32>javac Calculadora.java
C:\trabajo\es\java\aula\mentor\practicass32>javac Practica0.java_
```

Y ejecutamos la aplicación:



```
Command Prompt
C:\>java es.java.aula.mentor.practicass32.Practica0
```

Obteniendo como resultado:

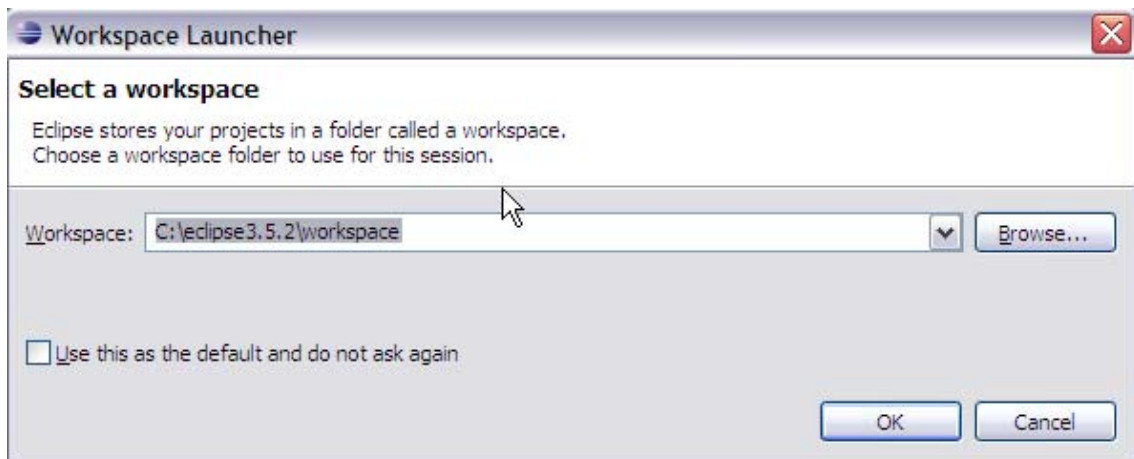


Como se puede observar, no se puede parar la aplicación cerrando el Frame de la calculadora, porque no hemos tenido en cuenta este tipo de eventos. Para parar la aplicación, pulsar Ctrl + C en la ventana de DOS.

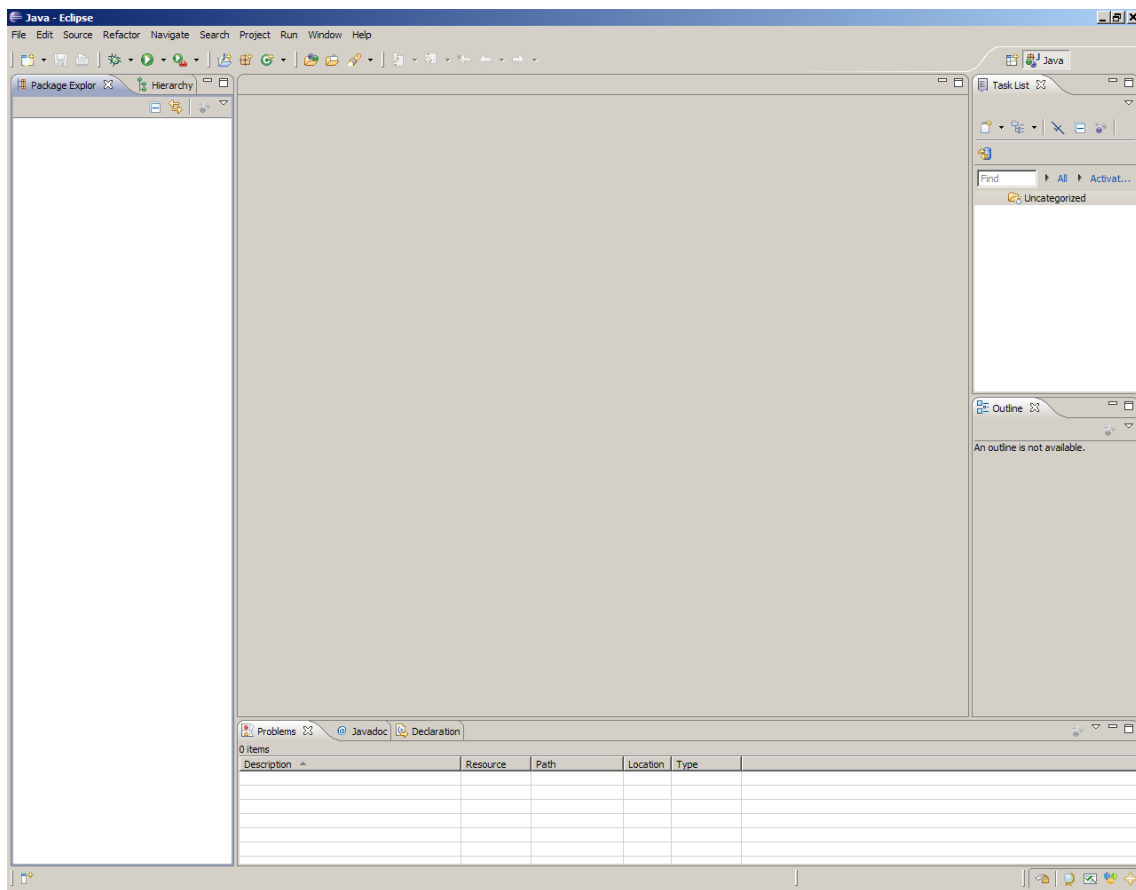
Solución con Eclipse:

Arrancar Eclipse, ejecutando C:\eclipse3.7.1\eclipse.exe

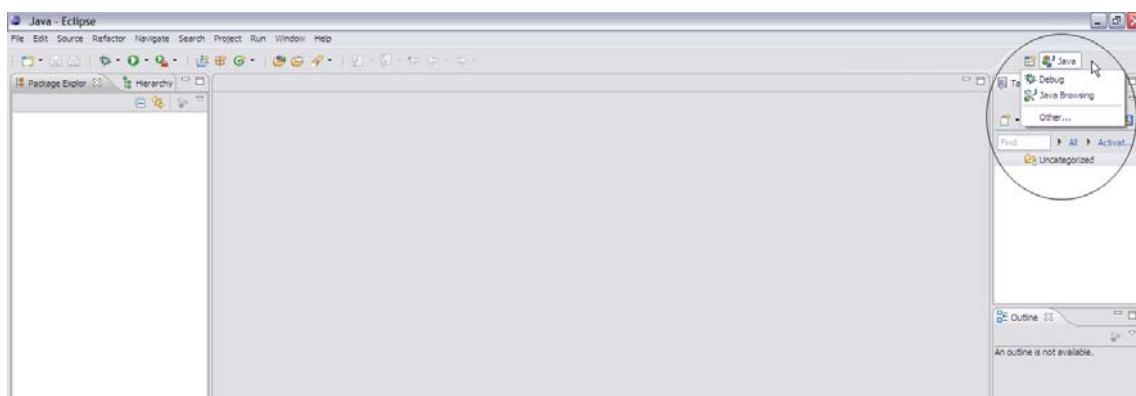
Seleccionar la ubicación del "workspace" (o área de trabajo):



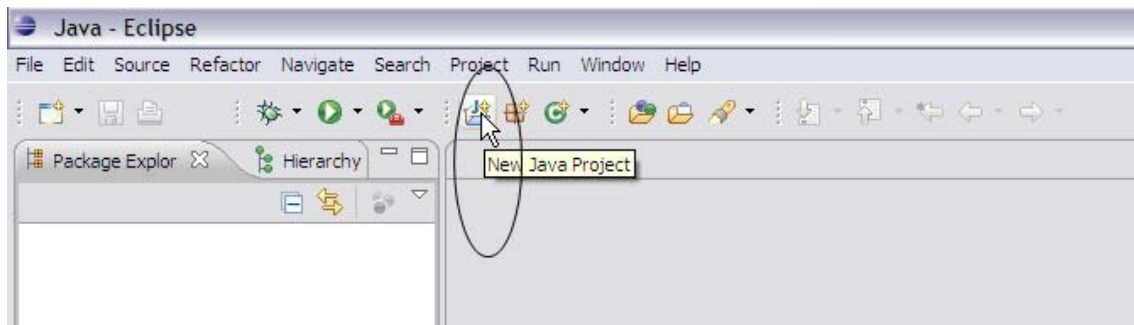
Aparecerá la pantalla para empezar a trabajar.



Verificar que la perspectiva Java está abierta, y si no cambiar a ella:

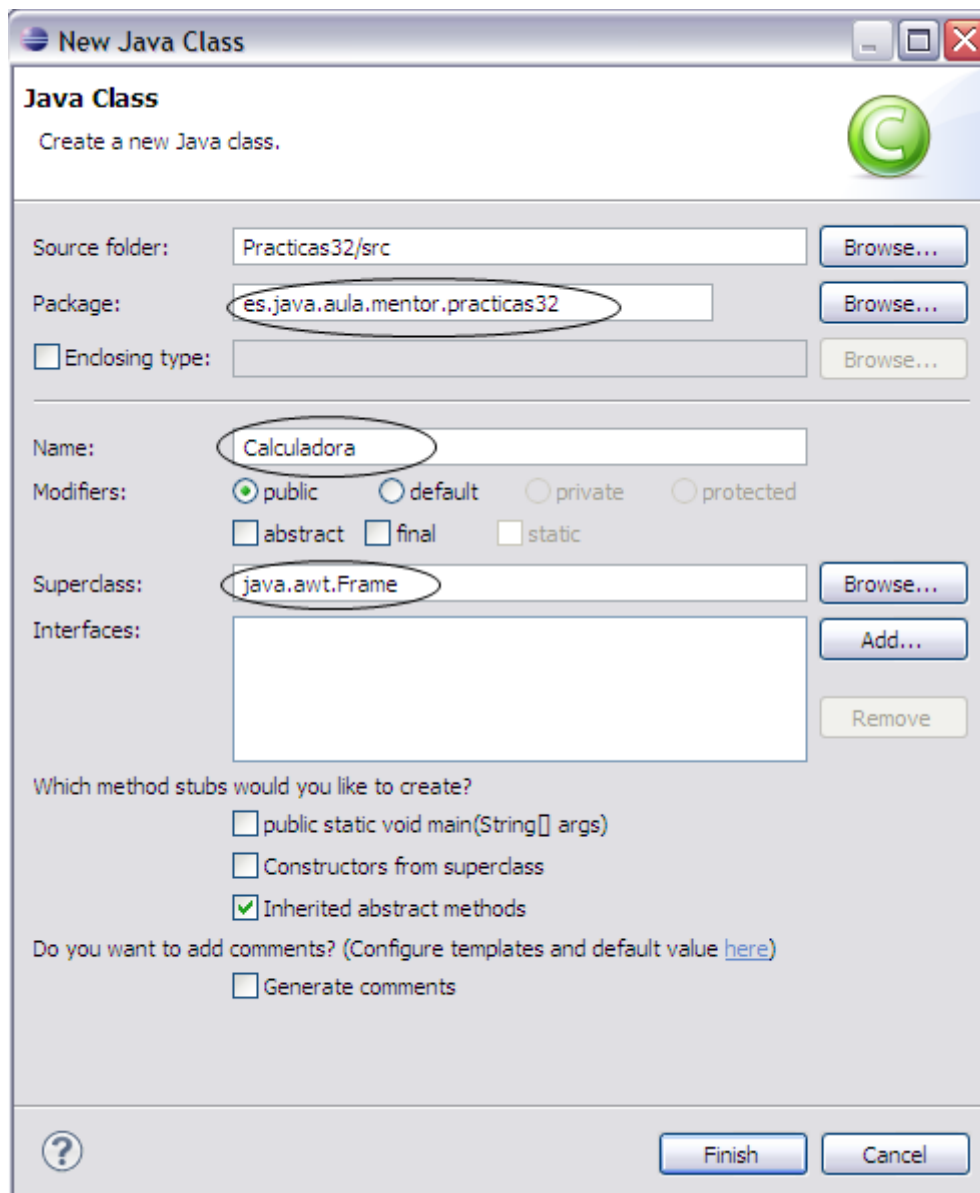


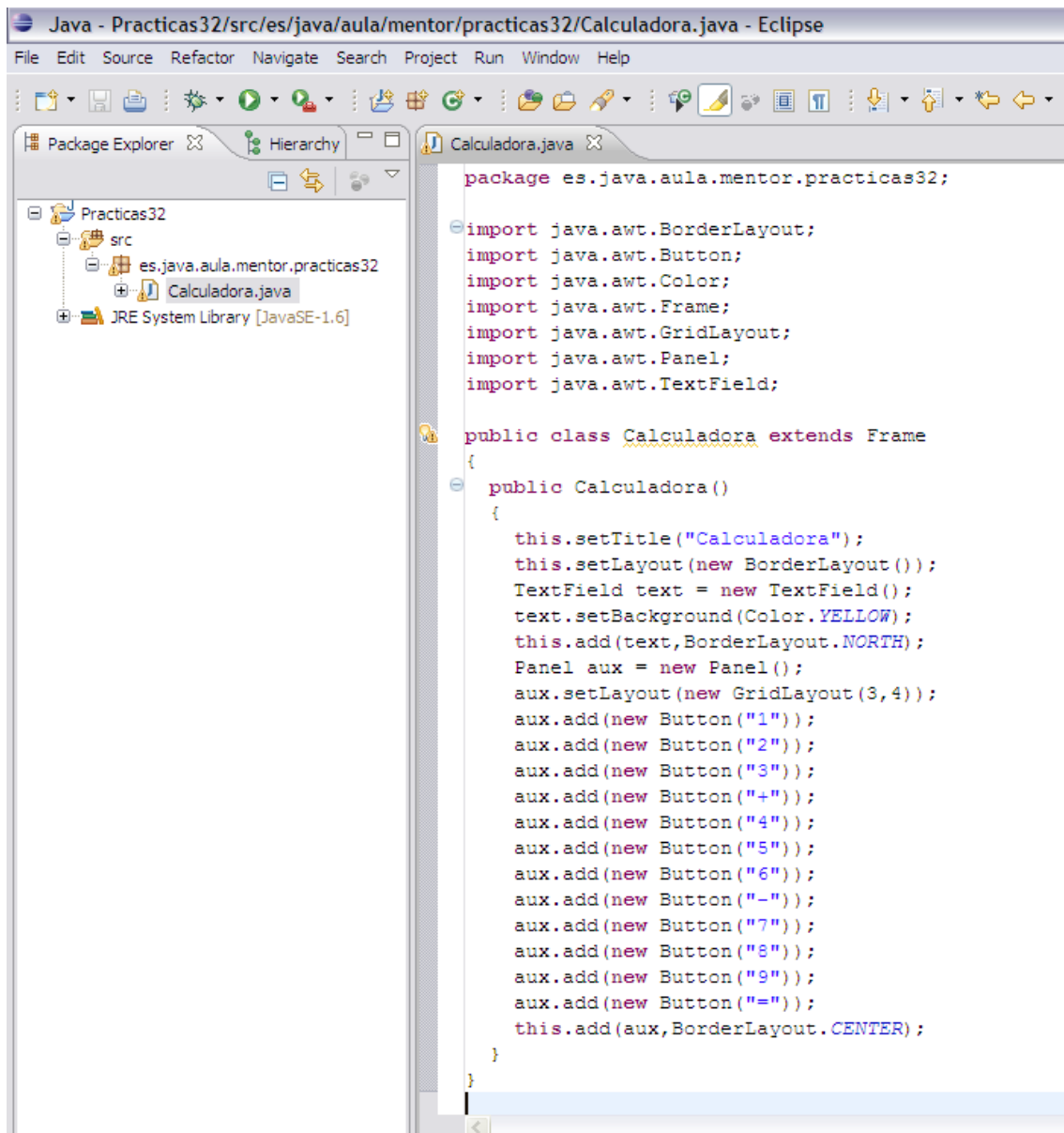
Crear un proyecto nuevo de nombre Practicas32:



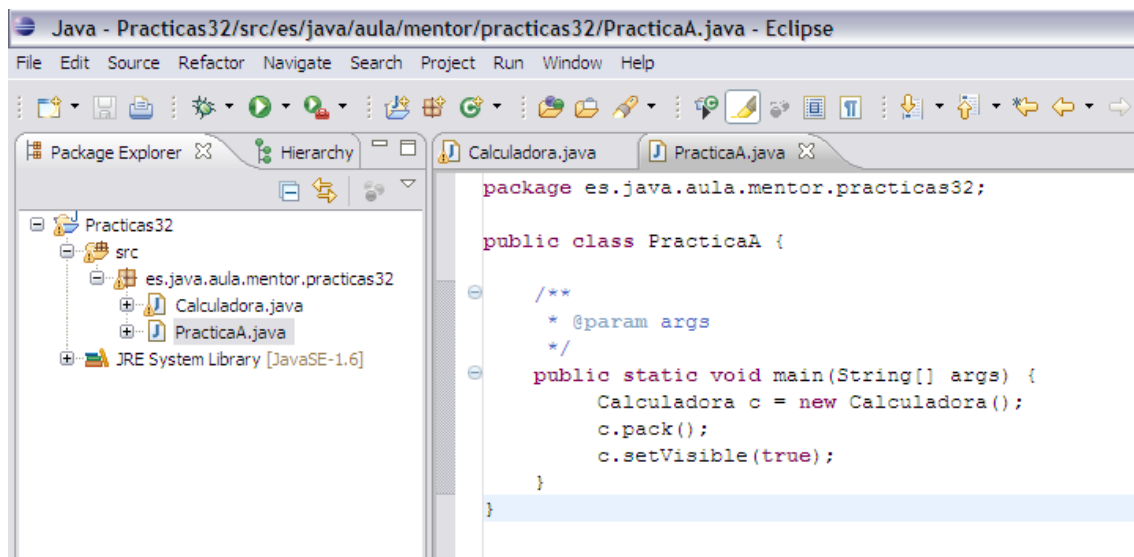
Darle el nombre y seleccionar "Finish".

Creamos la clase Calculadora, en el paquete es.java.aula.mentor.practicas32, indicando que heredamos de la clase java.awt.Frame:

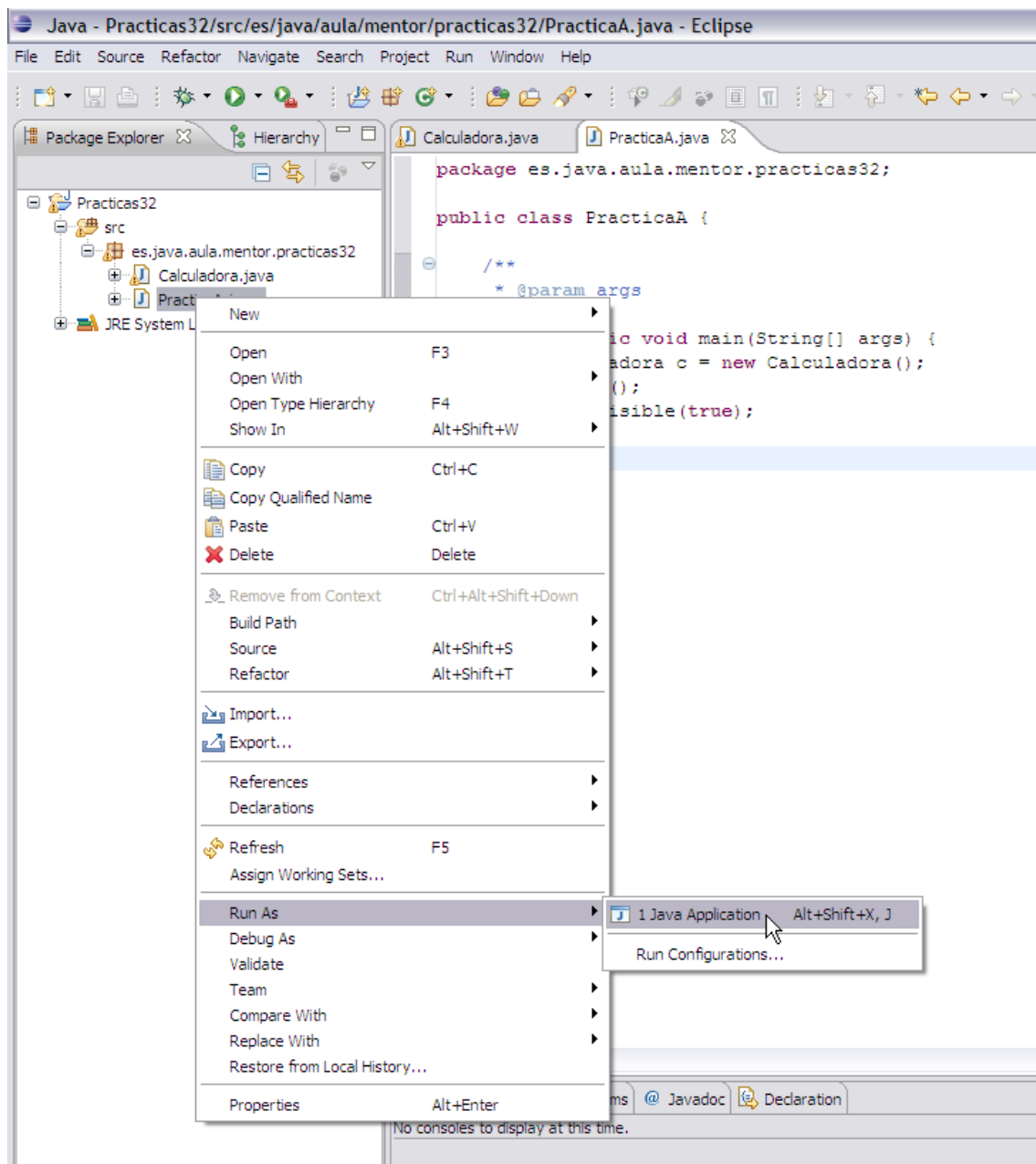


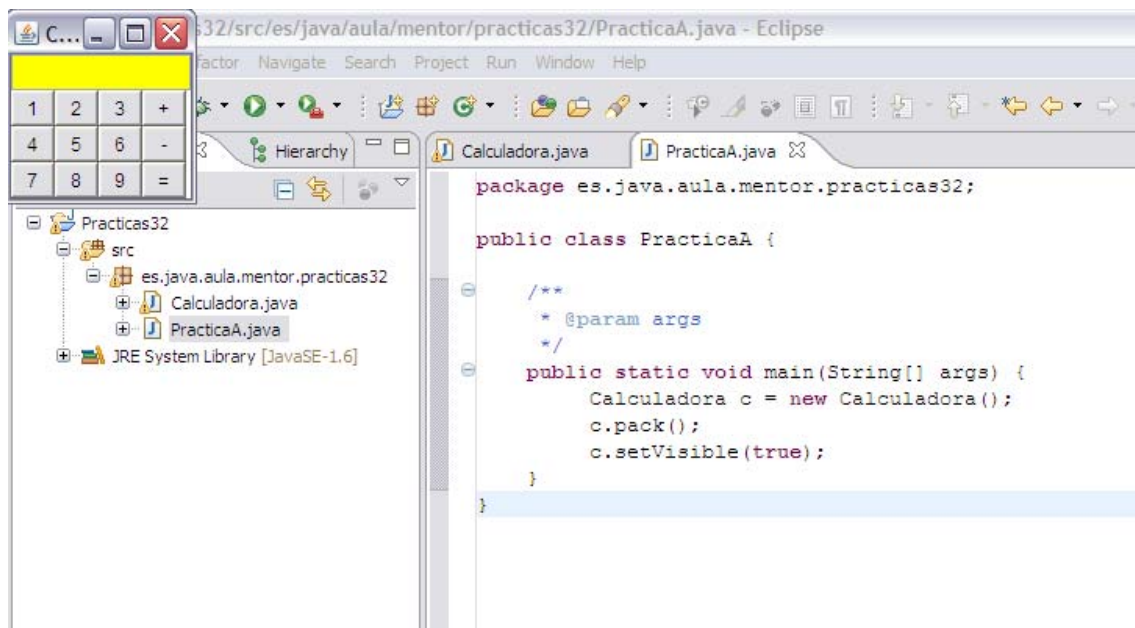


Creemos la clase PracticaA, en el mismo paquete e indicando que va a implementar el método main:

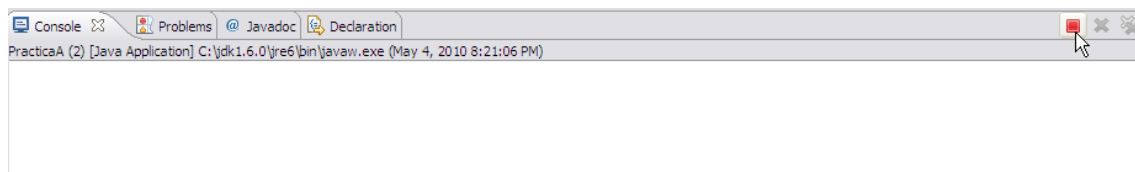


Situándonos en la clase PracticaA, con el botón derecho del ratón, ejecutar la clase mediante "Run As" -> "Java Application":





Como se puede observar, no se puede parar la aplicación cerrando el Frame de la calculadora, porque no hemos tenido en cuenta este tipo de eventos. Para poder parar la Calculadora, seleccionar el botón Rojo que está al nivel de la Vista Consola.

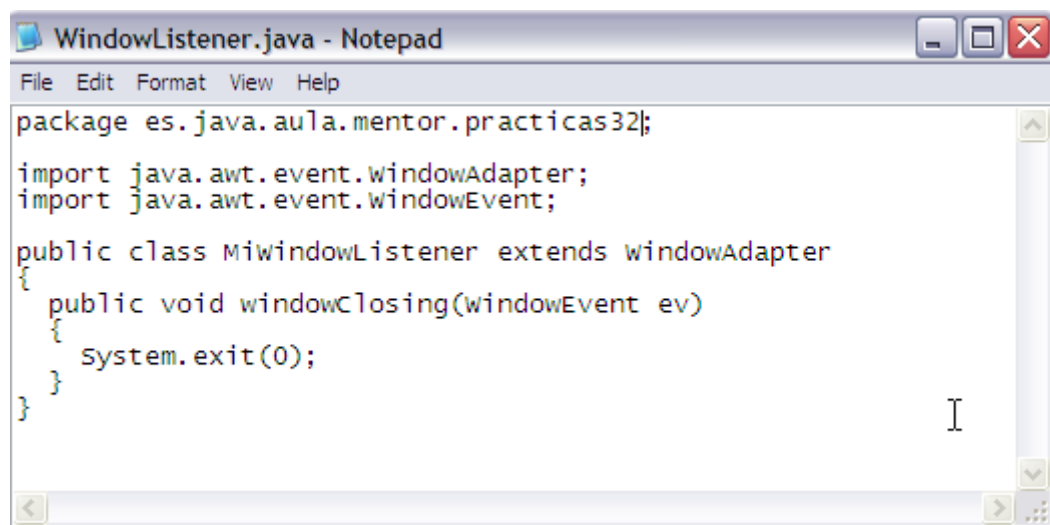
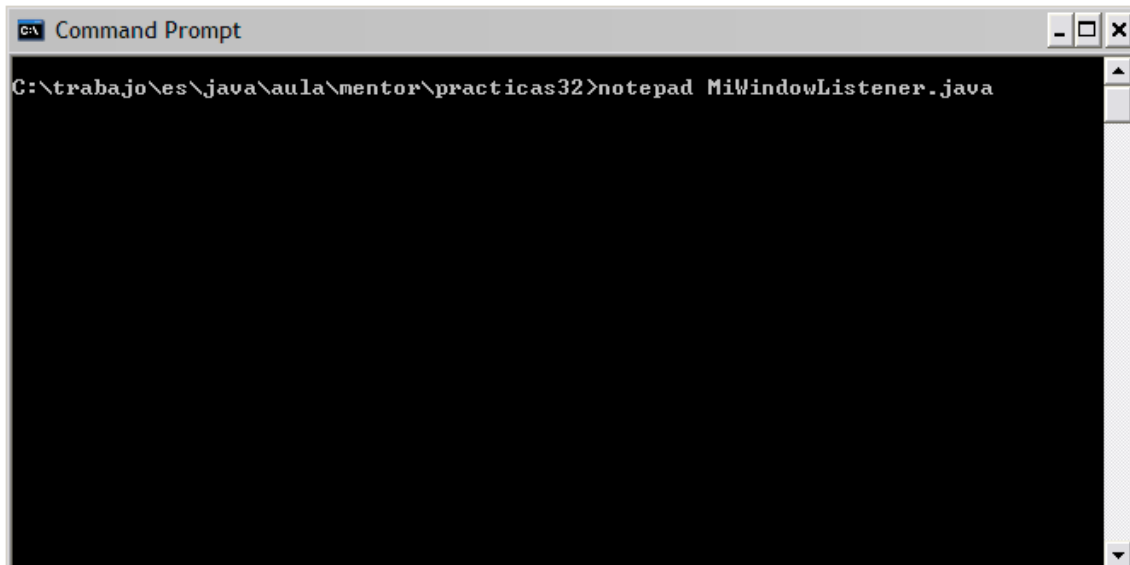


PRÁCTICA B: Implementar las siguientes funcionalidades en la calculadora del ejercicio anterior:

- Operación de suma, resta e igualdad.
- Tecleo de números mediante el ratón.
- Cierre de la aplicación mediante el botón de cerrado del frame.
- Evitar el tecleo de caracteres no numéricos en el text field.

Solución con la JDK de Oracle:

Desde el directorio `es\java\aula\mentor\practicas32`, creamos nuestra clase `MiWindowListener` que va a ser la encargada de cerrar el Frame y salir de la aplicación:



Modificamos nuestra clase `Calculadora` para que tenga en cuenta todos los eventos mencionados en el enunciado:

```

package es.java.aula.mentor.practicas32;

import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Color;
import java.awt.Frame;
import java.awt.GridLayout;
import java.awt.Panel;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class Calculadora extends Frame implements ActionListener, KeyListener
{
    private TextField t = null;
    private int oper1 = 0; // Primer operando.
    private int oper2 = 0; // Segundo operando.
    private String command = null; // Operación.

    private boolean sw = true; // Switch para añadir o borrar dígitos.

    public Calculadora()
    {
        this.setTitle("Calculadora");
        this.addWindowListener(new MiWindowListener());
        this.setLayout(new BorderLayout());

        t = new TextField(0);
        t.setBackground(Color.YELLOW);
        t.addKeyListener(this);
        Panel aux = new Panel();
        aux.setLayout(new GridLayout(3,4));

        Button b = null;
        b = new Button("1");
        b.addActionListener(this);
        aux.add(b);
        b = new Button("2");
        b.addActionListener(this);
        aux.add(b);
        b = new Button("3");
        b.addActionListener(this);
        aux.add(b);
        b = new Button("+");
        b.addActionListener(this);
        aux.add(b);
        b = new Button("4");
        b.addActionListener(this);
        aux.add(b);
        b = new Button("5");
        b.addActionListener(this);
        aux.add(b);
        b = new Button("6");
        b.addActionListener(this);
        aux.add(b);
        b = new Button("-");
        b.addActionListener(this);
        aux.add(b);
        b = new Button("7");
    }

    b.addActionListener(this);
    aux.add(b);
    b = new Button("8");
    b.addActionListener(this);
    aux.add(b);
    b = new Button("9");
    b.addActionListener(this);
    aux.add(b);
    b = new Button("=");
    b.addActionListener(this);
    aux.add(b);

    this.add(t, BorderLayout.NORTH);
    this.add(aux, BorderLayout.CENTER);

    public void actionPerformed(ActionEvent ev) // Se ha producido un Action Event
    {
        String aux = ((Button)ev.getSource()).getLabel();
        if(aux.equals("+") || aux.equals("-")) {
            command = aux;
            oper1 = Integer.parseInt(t.getText());
            sw = true;
        }
        else if(aux.equals("=")) {
            oper2 = Integer.parseInt(t.getText());
            if(command.equals("+")) {
                t.setText(new Integer(oper1 + oper2).toString());
                oper1 = oper1 + oper2; // Permite teclear un numero e "+" (repetiendo el último operador)
            }
            else {
                t.setText(new Integer(oper1 - oper2).toString());
                oper1 = oper1 - oper2; // Permite teclear un numero e "-" (repetiendo el último operador)
            }
            sw = true;
        }
        else {
            if(sw) {
                t.setText(aux);
                sw = false;
            }
            else {
                t.setText(t.getText() + aux);
            }
        }
    }

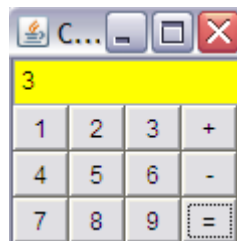
    public void keyPressed(KeyEvent ev) // Se ha producido un Key Event de tipo keyPressed

    public void keyReleased(KeyEvent ev) // Se ha producido un Key Event de tipo keyReleased
    {
        char tmp = ev.getKeyChar();
        if(!Character.isDigit(tmp)) {
            t.setText(t.getText().substring(0, t.getText().length() - 1));
            t.setCaretPosition(t.getText().length()); // Posicionar el al final del texto.
        }
    }

    public void keyTyped(KeyEvent ev) // Se ha producido un Key Event de tipo keyTyped
    }
}

```

Compilamos las dos clases anteriores y ejecutamos la clase PracticaA del ejercicio anterior. En esta practica, si que podemos cerrar la calculadora mediante la X del Frame.

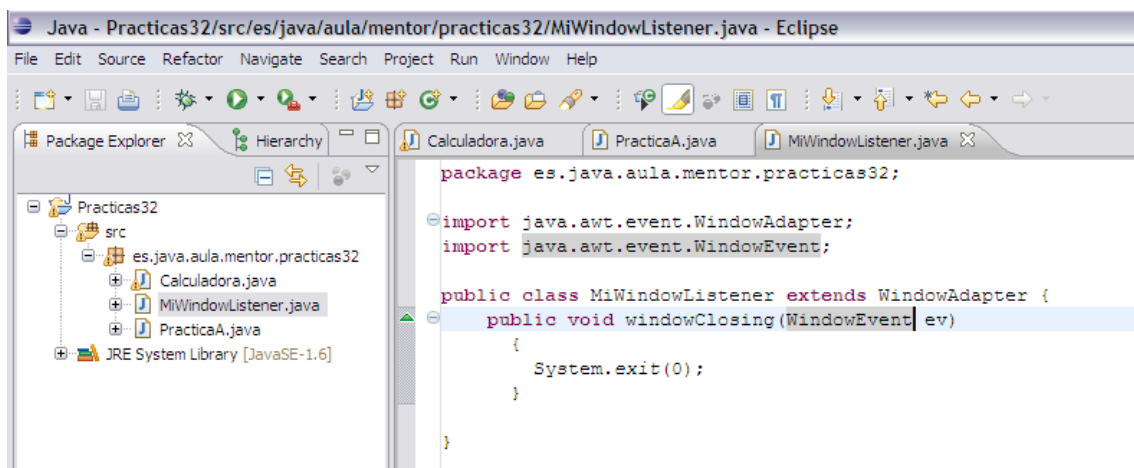
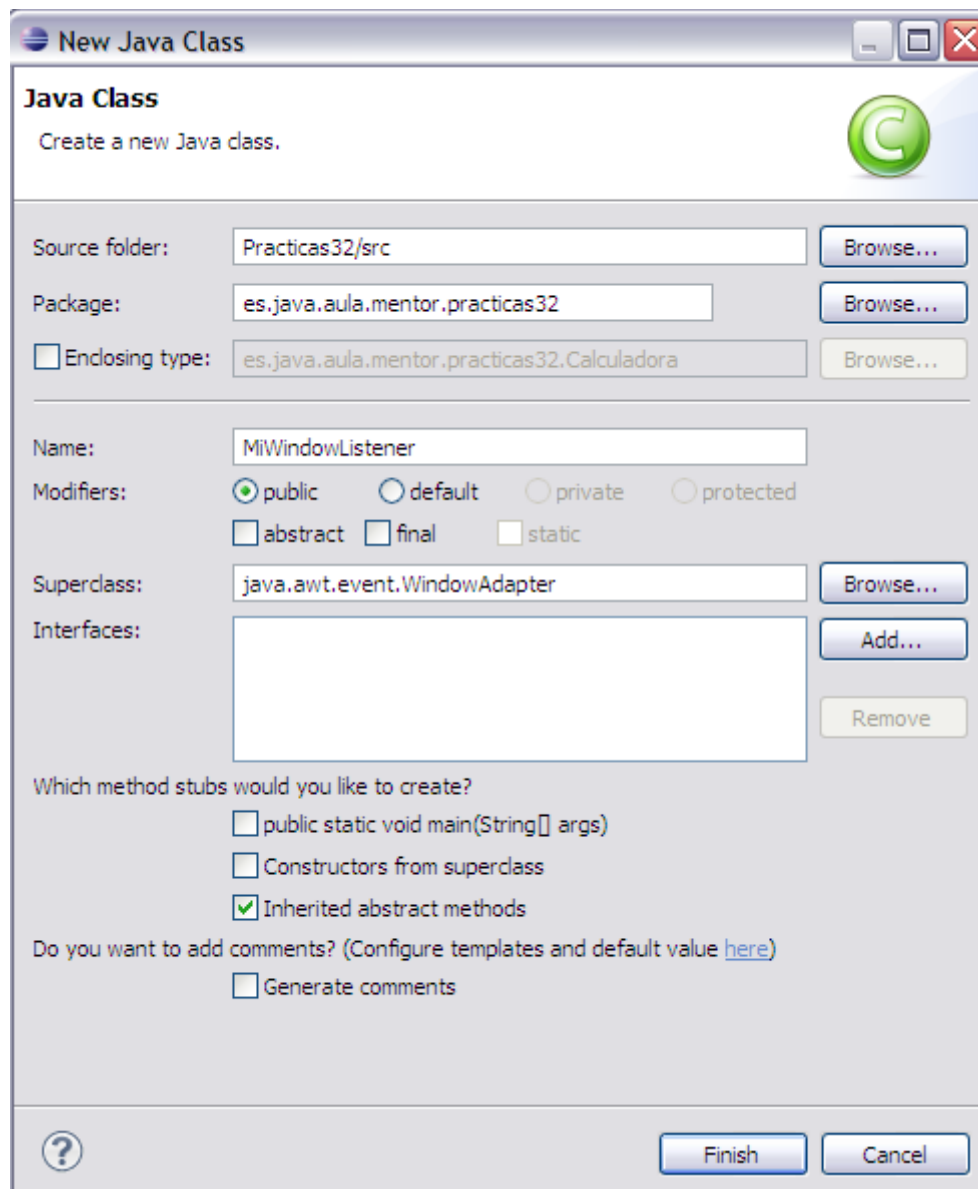


Solución con Eclipse:

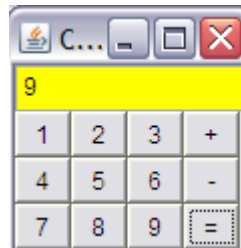
Arrancar Eclipse, ejecutando c:\eclipse3.7.1\eclipse.exe

Seleccionar la ubicación del "workspace" (o área de trabajo).

Ya tenemos creado el proyecto Practicas32, utilizado en la Práctica A. Vamos a crear la clase MiWindowListener que hereda de WindowAdapter, perteneciente al paquete es.java.aula.mentor.practicas32:



Al ejecutar, la aplicación comprobaremos que ya se puede terminar pulsando sobre la X del Frame:





para recordar

AWT, son las siglas de: Abstract Window Toolkit

Es una librería de clases Java para el desarrollo de interfaces de usuario gráficas (GUI).

Los elementos básicos que componen la librería AWT son:

- Los componentes (`java.awt.Component`) como Buttons, Labels, TextFields, etc....
- Los contenedores (`java.awt.Container`) como los Frames, los Panels, etc.... que pueden contener componentes.
- Los gestores de posición (`java.awt.LayoutManager`) que gestionan la disposición de los componentes dentro de los contenedores.
- Los eventos (`java.awt.AWTEvent`) que avisan de las acciones del usuario.

los contenedores sirven para agrupar componentes visuales. Pero, ¿cómo se distribuyen dichos componentes en su interior? Para dicho cometido, se utilizan los Layout Managers, encargados de distribuir los componentes en el interior del contenedor.

La interacción con un interfaz visual se realiza mediante la Gestión de Eventos. Los distintos componentes visuales (notificadores) lanzarán una serie de eventos como consecuencia de alguna acción, y ejecutarán código de otras clases (escuchadoras) que se habrán suscrito a dichos eventos previamente.

Para ser escuchador de una familia concreta de eventos, hay que implementar previamente el interfaz correspondiente, que contendrá las definiciones de los métodos que podrá invocar el notificador de dicha familia de eventos.

Paquete Swing

Tema 1.2

Índice de la unidad:

1. Introducción
2. Swing
3. Componentes visuales Swing
4. Layout Managers
5. Gestión de Eventos
6. Look & Feel

En la Unidad anterior, hemos visto los principios básicos de la programación de interfaces visuales en Java mediante el paquete AWT: componentes, contenedores, Layout Managers y gestión de eventos.

En esta Unidad, veremos un paquete nuevo que apareció unos años después para pulir las restricciones y rigidez de la AWT. No obstante, los fundamentos básicos siguen siendo los mismos por lo que recomendamos al lector estudiar primero la Unidad anterior, ya que en la presente Unidad se dan por sabidos dichos conceptos.

1. Introducción

Java Foundation Classes (JFC) es un conjunto de clases para mejorar el soporte al desarrollo de GUIs.

Se creó como una extensión de la AWT añadiendo las siguientes características:

- Componentes Swing.
- Soporte de Look & Feel.
- API de accesibilidad.
- Java 2D API.
- Soporte de Drag & Drop.

2. Swing

Swing es el conjunto de nuevos componentes visuales de JFC. Aunque habitualmente también se usa como término genérico para referirse a las JFC.

Su diferencia más importante con la AWT es que los componentes son "*lightweight*" (ligeros). Esto significa que a diferencia de lo que ocurriera con los componentes AWT que son "*heavyweight*" (pesados), que se pintaban en la pantalla a través de llamadas a las librerías nativas del Sistema Operativo, en la Swing los componentes se pintan pixel a pixel ellos mismos.

Por eso, los componentes AWT siempre tenían la apariencia que dictaba el Sistema Operativo mientras que ahora en la Swing, los componentes pueden mostrarse por

ejemplo en Windows como si estuvieran en un Linux, etc... Veremos esta característica conocida con el nombre de *"Look & Feel"* más adelante.

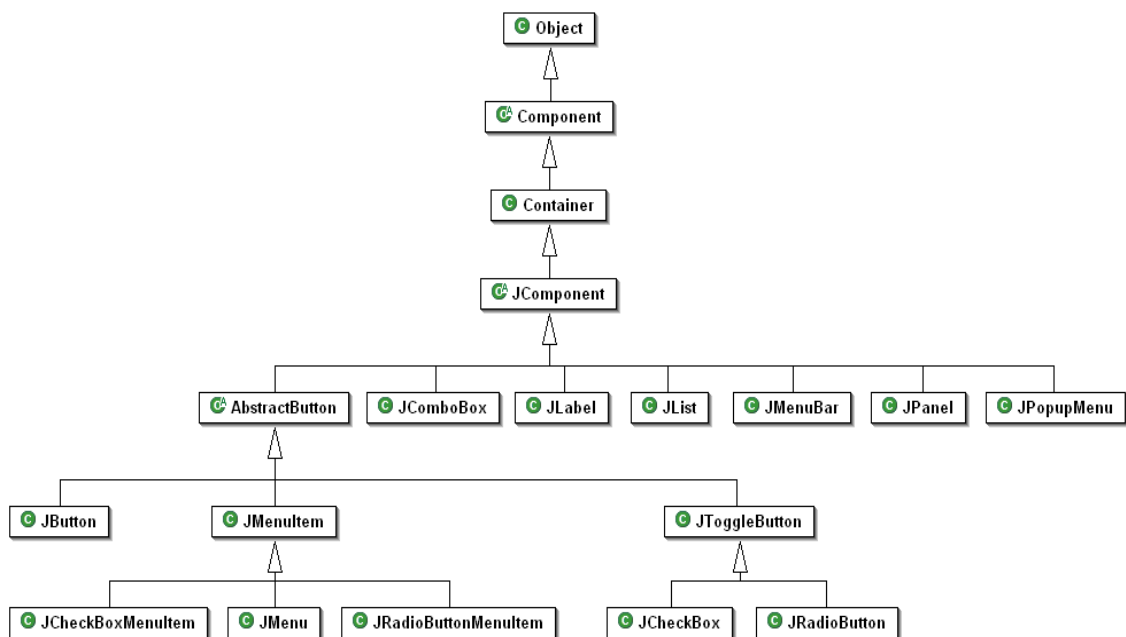
Nunca debemos mezclar componentes Swing con componentes AWT en una misma aplicación: "Lightweight" vs. "Heavyweight". Ya que el Sistema Operativo siempre manda, y aunque tuvieramos un componente Swing sobre uno AWT, siempre se verá el AWT.

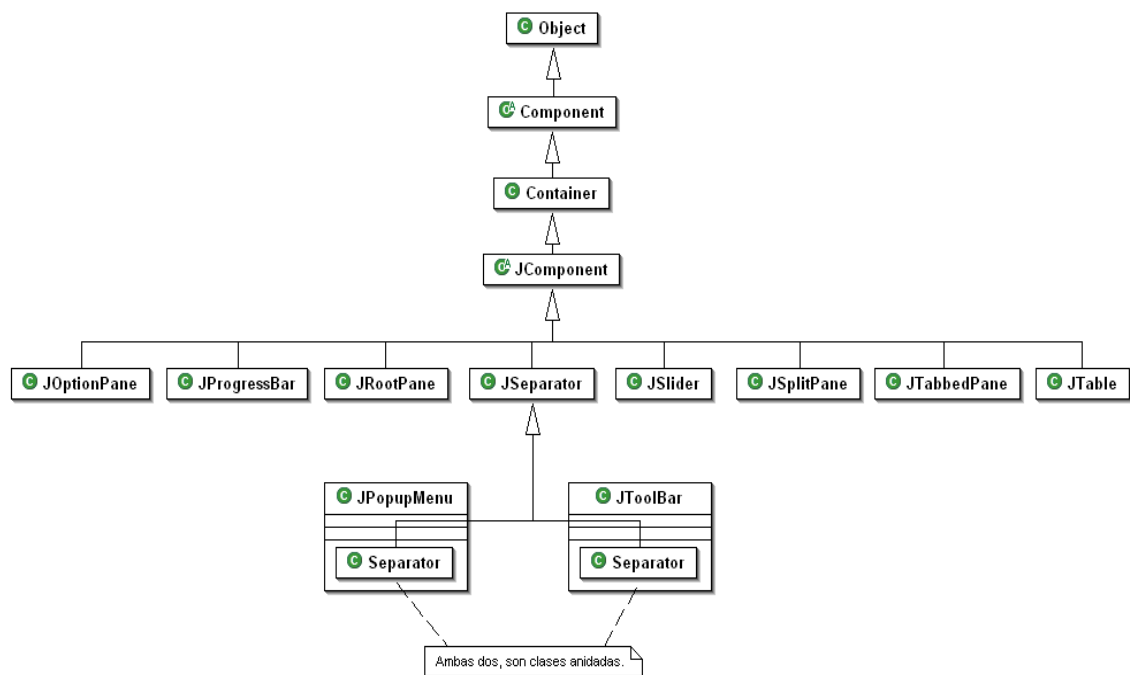
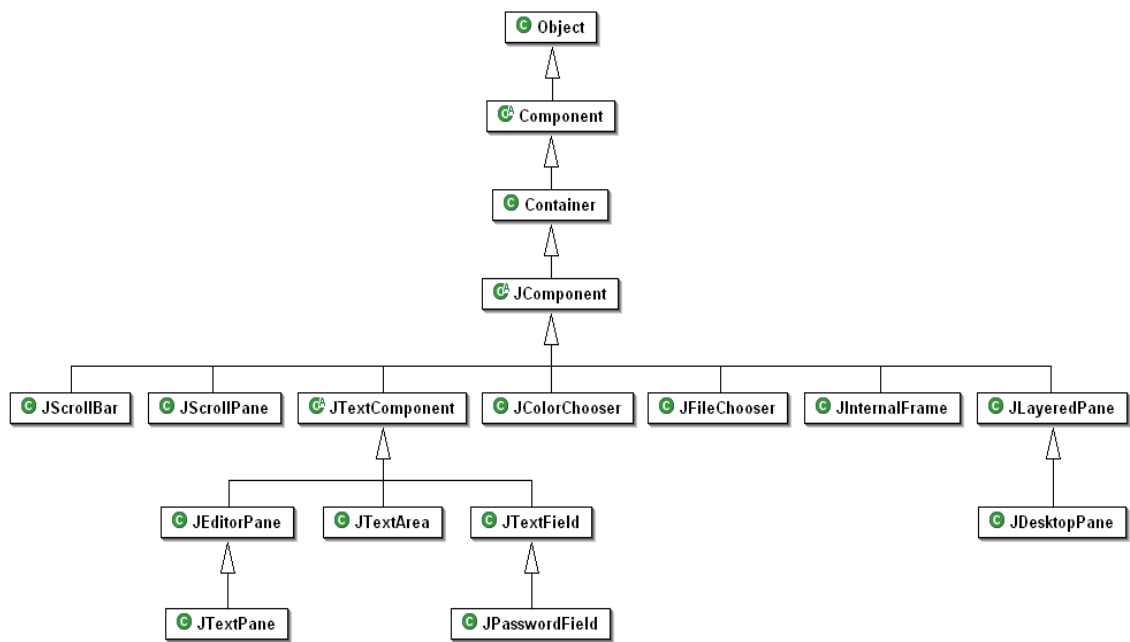
Para diferenciar los componentes Swing de los AWT, sus nombres están precedidos por una 'J'. Y todas las clases se encuentran en el paquete javax.swing.*

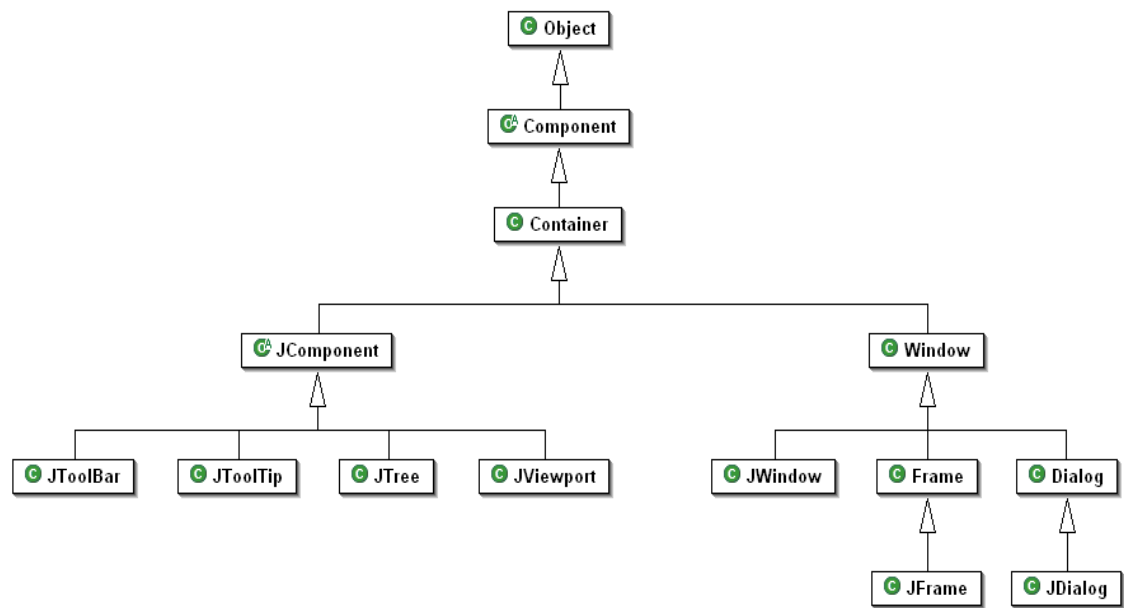
Swing sigue trabajando con los conceptos de la AWT:

- Contenedores.
- Componentes.
- LayoutManagers.
- Eventos.

Y evidentemente, sus clases también están dispuestas en un diseño Orientado a Objetos en una jerarquía de herencia:







javax.swing.JComponent hereda de la clase AWT **java.awt.Container**.

Se trata de una clase abstracta que implementa toda la funcionalidad básica de las clases visuales que la Swing añadió sobre la AWT:

- Ayudas emergentes.
- Bordes.
- Gestión del Look & Feel.
- Gestión de la accesibilidad.
- Gestión de teclas asociadas.
- Soporte de Drag & Drop.

Migrar aplicaciones de AWT a Swing, a priori puede ser una tarea más o menos sencilla aunque siempre es recomendable realizar un buen estudio de migración para evitar sorpresas. Los pasos básicos serían:

- Eliminar todos los import de paquetes `java.awt.*` siempre que sea posible.
- Importar el paquete `javax.swing.*`
- Cambiar cada componente AWT por el Swing más parecido:

Frame -> JFrame, Button -> JButton, etc...

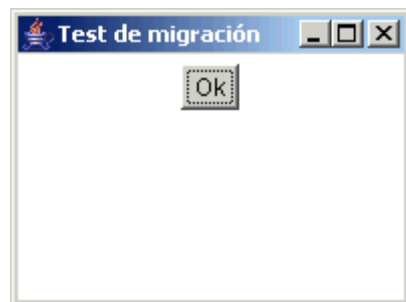
¡Ojo! No se pueden añadir componentes o establecer LayoutManagers directamente sobre JWindow, JFrame, JDialog o JApplet. Hay que hacerlo sobre el Container que devuelve el método:

public Container getContentPane();

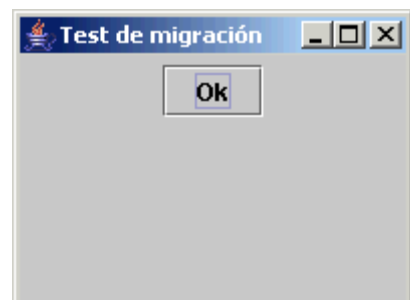
Nota: A partir del Java SE 5.0, el compilador no fuerza al desarrollador a escribir este método, ya se encarga él de hacerlo. Pero hay que conocer esta diferencia con la AWT por si nos encontramos con el mantenimiento de aplicaciones que usan niveles de Java anteriores.

- **Ejemplo:** Migrando de AWT a Swing:

```
import java.awt.Button;
import java.awt.FlowLayout;
import java.awt.Frame;
public class Test
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.setTitle("Test de migración");
        f.setSize(200,150);
        f.setLayout(new FlowLayout());
        Button b = new Button("Ok");
        f.add(b);
        f.setVisible(true);
    }
}
```



```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
public class Test
{
    public static void main(String[] args)
    {
```



```

JFrame f = new JFrame();
f.setTitle("Test de migración");
f.setSize(200,150);
f.getContentPane().setLayout(new FlowLayout());
JButton b = new JButton("Ok");
f.getContentPane().add(b);
f.setVisible(true);
}
}

```

3. Componentes visuales Swing

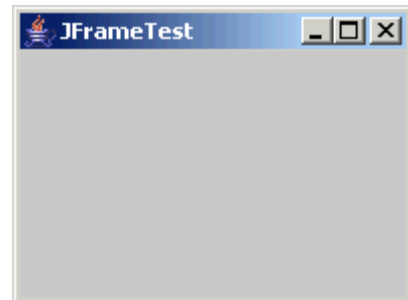
Ahora haremos un repaso mediante ejemplos de código de un gran porcentaje de los componentes visuales existentes en la Swing. Para profundizar en cada uno de ellos, recomendamos examinar en el API.

3.1 javax.swing.JFrame

```

import javax.swing.JFrame;
public class JFrameTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JFrameTest");
        f.setSize(200,150);
        f.setVisible(true);
    }
}

```



3.2 javax.swing.JInternalFrame

```

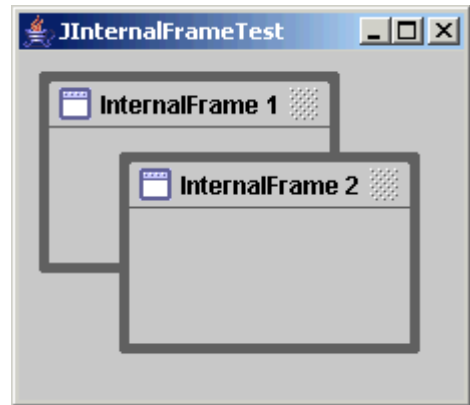
import javax.swing.*;
public class JInternalFrameTest
{
    public static void main(String[] args)

```

```

{
    JFrame f = new JFrame();
    f.setTitle("JInternalFrameTest");
    f.getContentPane().setLayout(null);
    f.setSize(230,200);
    JInternalFrame f1 =
        new JInternalFrame("InternalFrame 1");
    f1.setBounds(10,10,150,100);
    f1.setVisible(true);
    JInternalFrame f2 = new JInternalFrame("InternalFrame 2");
    f2.setBounds(50,50,150,100);
    f2.setVisible(true);
    f.getContentPane().add(f2);
    f.getContentPane().add(f1);
    f.setVisible(true);
}
}

```

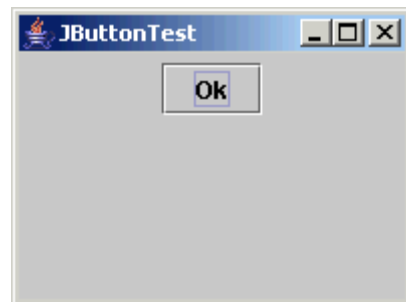


3.3 javax.swing.JButton

```

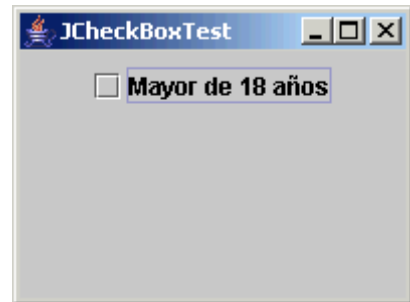
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
public class JButtonTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JButtonTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JButton b = new JButton("Ok");
        f.getContentPane().add(b);
        f.setVisible(true);
    }
}

```



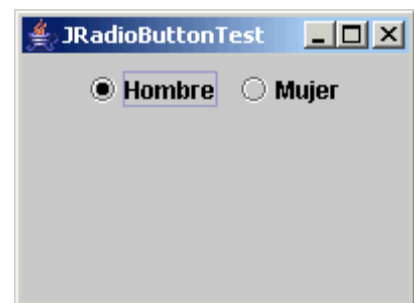
3.4 javax.swing.JCheckBox

```
import java.awt.FlowLayout;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
public class JCheckboxTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JCheckBoxTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JCheckBox c = new JCheckBox("Mayor de 18 años");
        f.getContentPane().add(c);
        f.setVisible(true);
    }
}
```



3.5 javax.swing.JRadioButton

```
import java.awt.FlowLayout;
import javax.swing.*;
import javax.swing.JFrame;
import javax.swing.JRadioButton;
public class JRadioButtonTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JRadioButtonTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        ButtonGroup bg = new ButtonGroup();
```




```

JRadioButton c1 = new JRadioButton("Hombre",true);
bg.add(c1);
JRadioButton c2 = new JRadioButton("Mujer",false);
bg.add(c2);
f.getContentPane().add(c1);
f.getContentPane().add(c2);
f.setVisible(true);
}
}

```

3.6 javax.swing.JToggleButton

```

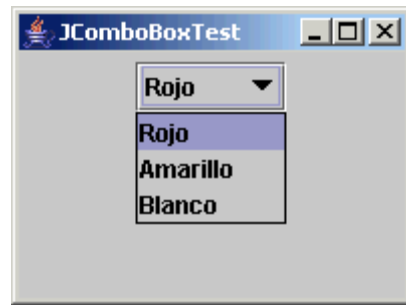
import java.awt.FlowLayout;
import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JToggleButton;
public class JToggleButtonTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JToggleButtonTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        ButtonGroup bg = new ButtonGroup();
        JToggleButton b1 = new JToggleButton("Hombre",true);
        bg.add(b1);
        JToggleButton b2 = new JToggleButton("Mujer",false);
        bg.add(b2);
        f.getContentPane().add(b1);
        f.getContentPane().add(b2);
        f.setVisible(true);
    }
}

```



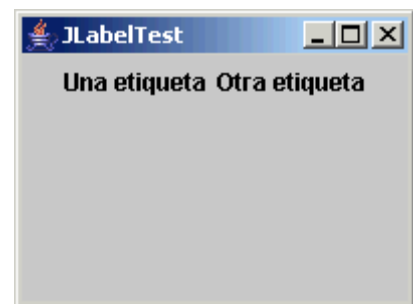
3.7 javax.swing.JComboBox

```
import java.awt.FlowLayout;
import javax.swing.JComboBox;
import javax.swing.JFrame;
public class JComboBoxTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JComboBoxTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        String[] list = { "Rojo","Amarillo","Blanco"};
        JComboBox c = new JComboBox(list);
        f.getContentPane().add(c);
        f.setVisible(true);
    }
}
```



3.8 javax.swing.JLabel

```
import java.awt.FlowLayout;
import javax.swing.*;
public class JLabelTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JLabelTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JLabel l1 = new JLabel("Una etiqueta");
        JLabel l2 = new JLabel();
        l2.setText("Otra etiqueta");
        f.getContentPane().add(l1);
```



```

        f.getContentPane().add(l2);
        f.setVisible(true);
    }
}

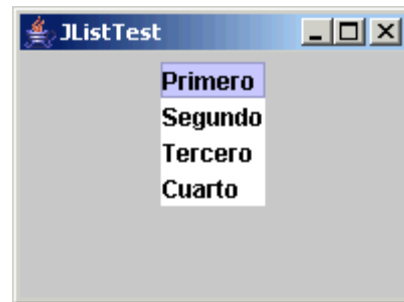
```

3.9 javax.swing.JList

```

import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JList;
public class JListTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JListTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        String[] list = {"Primero","Segundo","Tercero","Cuarto"};
        JList l = new JList(list);
        f.getContentPane().add(l);
        f.setVisible(true);
    }
}

```

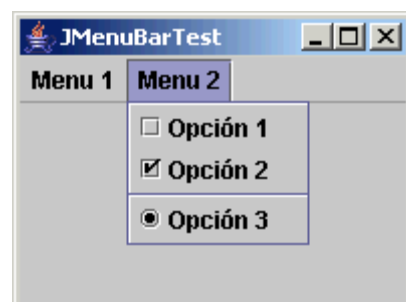


3.10 javax.swing.JMenuBar

```

import javax.swing.*;
public class JMenuBarTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame("JMenuBarTest");
        f.setSize(200,150);
        JMenuBar mb = new JMenuBar();
    }
}

```



```

JMenu m1 = new JMenu("Menu 1");
m1.add(new JMenuItem("Opción 1"));
m1.add(new JMenuItem("Opción 2"));
JMenu m2 = new JMenu("Menu 2");
m2.add(new JCheckBoxMenuItem("Opción 1"));
m2.add(new JCheckBoxMenuItem("Opción 2", true));
m2.addSeparator();
m2.add(new JRadioButtonMenuItem("Opción 3", true));
mb.add(m1);
mb.add(m2);
f.setJMenuBar(mb);
f.setVisible(true);
}
}

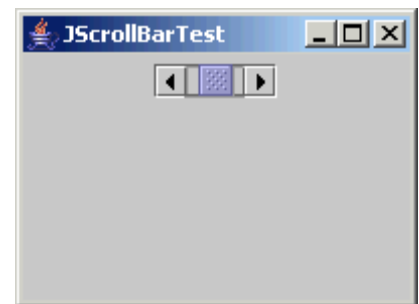
```

3.11 javax.swing.JScrollBar

```

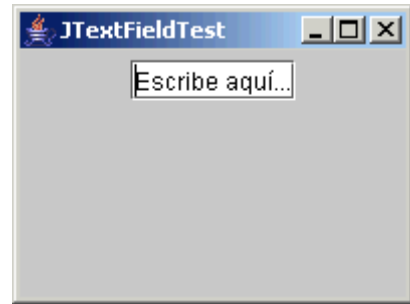
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollBar;
public class JScrollBarTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JScrollBarTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JScrollBar sb = new JScrollBar(JScrollBar.HORIZONTAL,0,5,-100,100);
        f.getContentPane().add(sb);
        f.setVisible(true);
    }
}

```



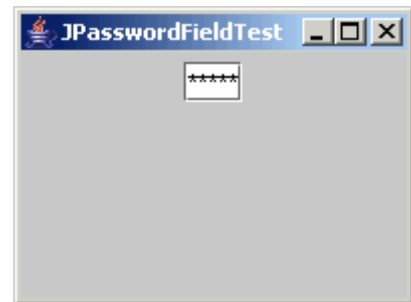
3.12 javax.swing.JTextField

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class JTextFieldTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JTextFieldTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JTextField tf = new JTextField("Escribe aquí...");
        f.getContentPane().add(tf);
        f.setVisible(true);
    }
}
```



3.13 javax.swing.JPasswordField

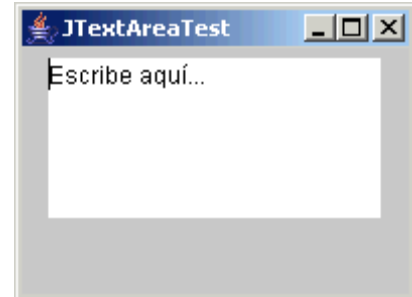
```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JPasswordField;
public class JPasswordFieldTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JPasswordFieldTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JPasswordField pf = new JPasswordField("miPassword");
        f.getContentPane().add(pf);
        f.setVisible(true);
    }
}
```



```
}
```

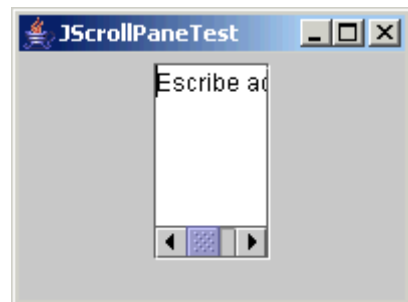
3.14 javax.swing.JTextArea

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JTextArea;
public class JTextAreaTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JTextAreaTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JTextArea ta = new JTextArea("Escribe aquí...",5,15);
        f.getContentPane().add(ta);
        f.setVisible(true);
    }
}
```



3.15 javax.swing.JScrollPane

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
public class JScrollPaneTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JScrollPaneTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
```



```

        JTextArea ta = new JTextArea("Escribe aquí...",5,5);
        JScrollPane p = new JScrollPane(ta);
        f.getContentPane().add(p);
        f.setVisible(true);
    }
}

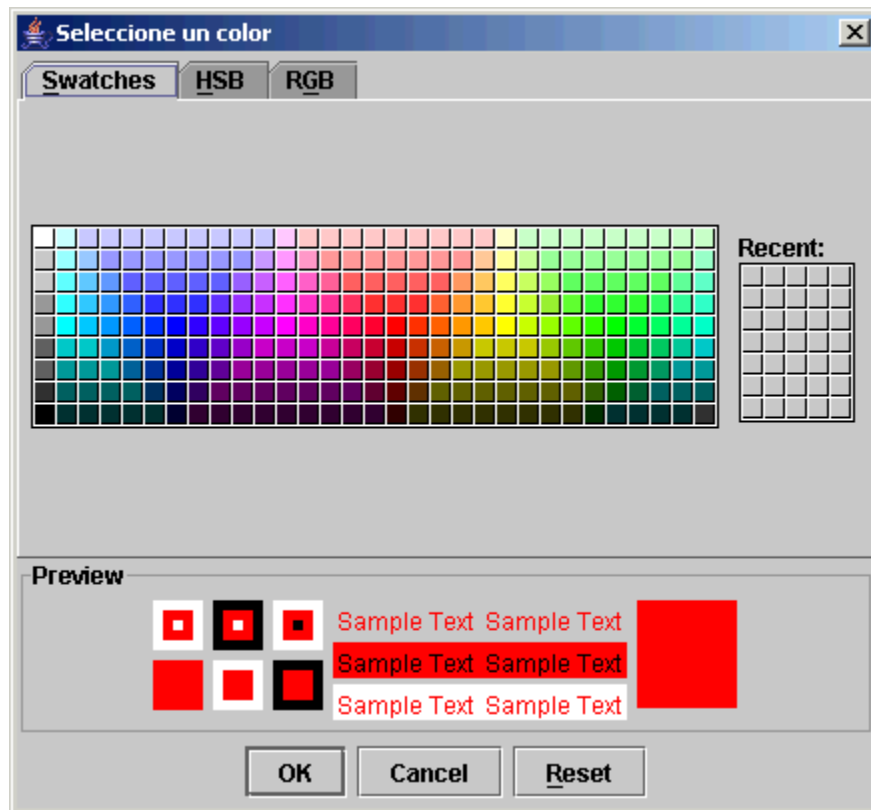
```

3.16 javax.swing.JColorChooser

```

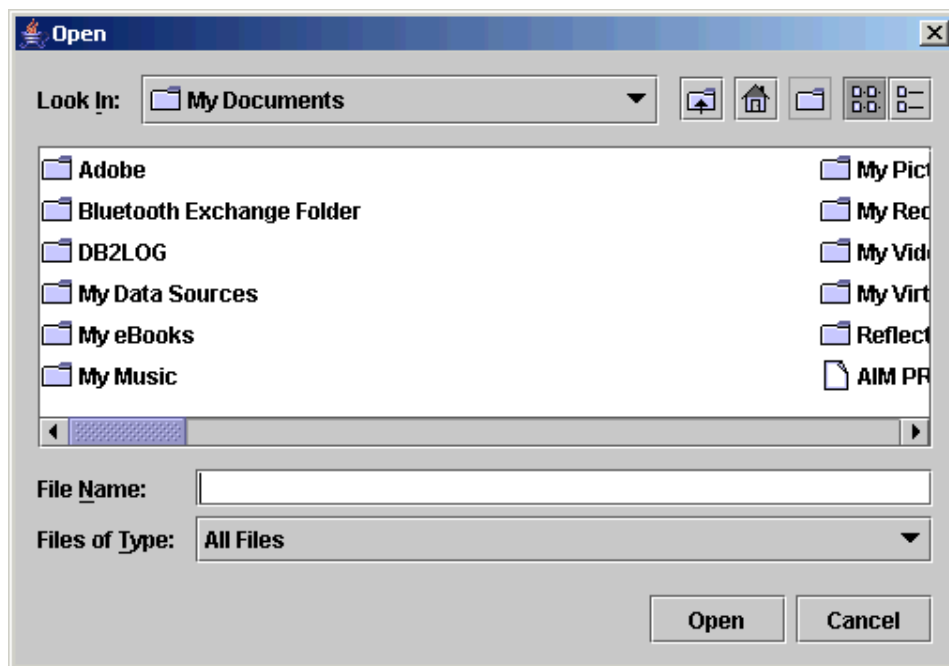
import java.awt.Color;
import javax.swing.JColorChooser;
import javax.swing.JFrame;
public class JColorChooserTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JColorChooserTest");
        f.setSize(200,150);
        f.setVisible(true);
        Color c = JColorChooser.showDialog(f,"Seleccione un color",Color.RED);
        System.out.println("El color seleccionado es: " + c);
    }
}

```



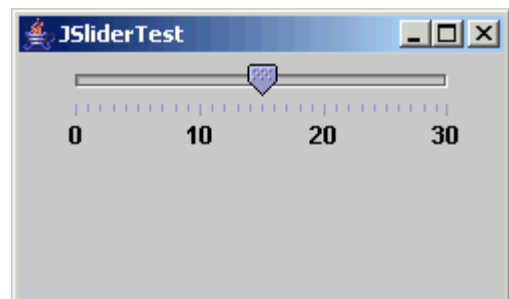
3.17 javax.swing.JFileChooser

```
import javax.swing.JFileChooser;
import javax.swing.JFrame;
public class Test
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JFileChooserTest");
        f.setSize(200,150);
        f.setVisible(true);
        JFileChooser fc = new JFileChooser();
        int op = fc.showOpenDialog(f);
        if(op == JFileChooser.APPROVE_OPTION)
            System.out.println(fc.getSelectedFile());
    }
}
```

3.18 javax.swing.JSlider

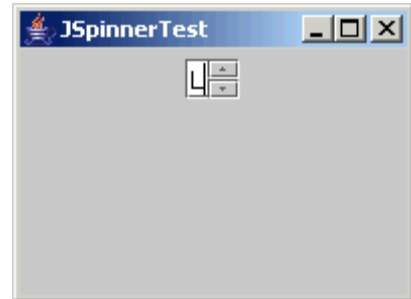
```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JSlider;
public class JSliderTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JSliderTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JSlider s = new JSlider(JSlider.HORIZONTAL,0,30,15);
        s.setMajorTickSpacing(10);
        s.setMinorTickSpacing(1);
        s.setPaintTicks(true);
        s.setPaintLabels(true);
        f.getContentPane().add(s);
        f.setVisible(true);
    }
}
```



```
}
```

3.19 javax.swing.JSpinner

```
import java.awt.FlowLayout;
import javax.swing.*;
public class JSpinnerTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JSpinnerTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        String[] dias = { "L","M","X","J","V","S","D" };
        SpinnerListModel modelo = new SpinnerListModel(dias);
        JSpinner s = new JSpinner(modelo);
        f.getContentPane().add(s);
        f.setVisible(true);
    }
}
```



3.20 javax.swing.JTable

```
import java.awt.FlowLayout;
import javax.swing.*;
public class JTableTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JTableTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        Object[][] datos =
```

```

{
    { "Nombre1", "Apellido1", new Integer(911234567) },
    { "Nombre2", "Apellido2", new Integer(917463527) },
    { "Nombre3", "Apellido3", new Integer(912494735) },
    { "Nombre4", "Apellido4", new Integer(912387448) },
};
String[] columnas = {"Nombre", "Apellidos", "Tfno"};
JTable t = new JTable(datos, columnas);
JScrollPane sp = new JScrollPane(t);
f.getContentPane().add(sp);
f.setVisible(true);
}
}

```



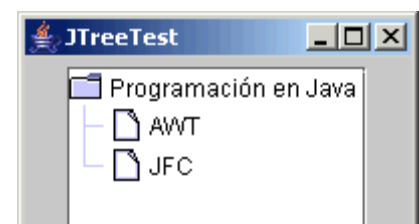
Nombre	Apellidos	Tfno
Nombre1	Apellido1	911234567
Nombre2	Apellido2	917463527
Nombre3	Apellido3	912494735
Nombre4	Apellido4	912387448

3.21 javax.swing.JTree

```

import java.awt.FlowLayout;
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class JTreeTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JTreeTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        DefaultMutableTreeNode titulo =

```



```

        new DefaultMutableTreeNode("Programación en Java");
DefaultMutableTreeNode capitulo = new DefaultMutableTreeNode("AWT");
titulo.add(capitulo);
capitulo = new DefaultMutableTreeNode("JFC");
titulo.add(capitulo);
JTree tree = new JTree(titulo);
JScrollPane sp = new JScrollPane(tree);
f.getContentPane().add(sp);
f.setVisible(true);
    }
}

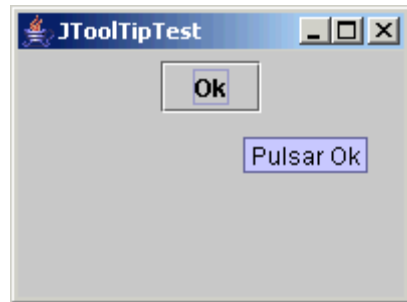
```

3.22 java.swing.JToolTip

```

import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
public class JToolTipTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JToolTipTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JButton b = new JButton("Ok");
        b.setToolTipText("Pulsar Ok");
        f.getContentPane().add(b);
        f.setVisible(true);
    }
}

```



3.23 javax.swing.JDialog

```
import javax.swing.JDialog;
import javax.swing.JFrame;
public class JDialogTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JFrameTest");
        f.setSize(200,150);
        f.setVisible(true);
        JDialog d = new JDialog(f);
        d.setTitle("JDialogTest");
        d.setBounds(50,50,70,50);
        d.setVisible(true);
    }
}
```



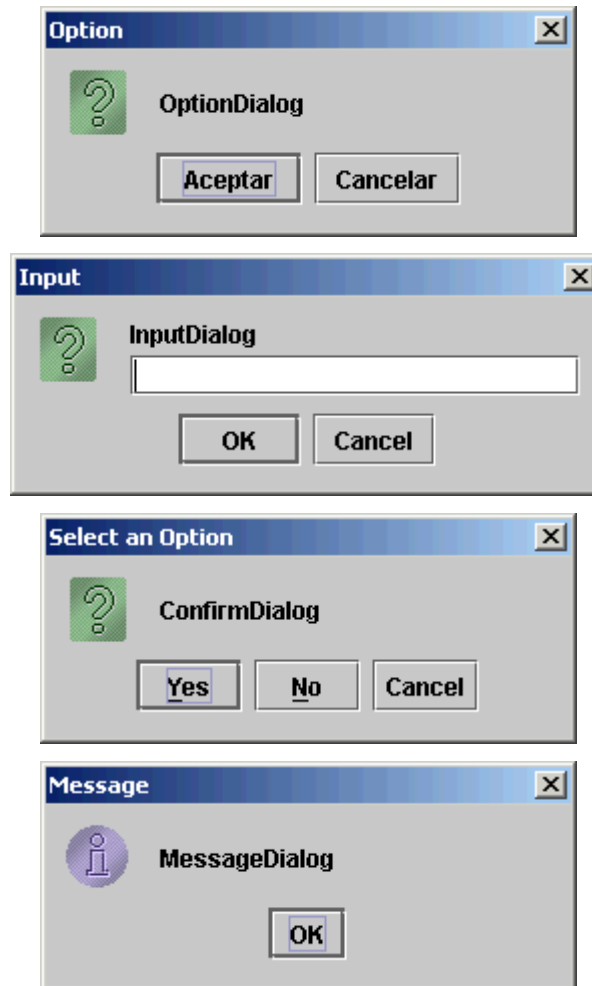
3.24 javax.swing.JOptionPane

```
import javax.swing.*;
public class JOptionPaneTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JOptionPaneTest");
        f.setSize(200,150);
        f.setVisible(true);
        JOptionPane.showMessageDialog(f, "MessageDialog");
        Object[] opciones = { "Aceptar", "Cancelar" };
        int i = JOptionPane.showOptionDialog(f, "OptionDialog",
        "Option", JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE, null,
        opciones, opciones[0]);
    }
}
```

```

i = JOptionPane.showConfirmDialog(f,"ConfirmDialog");
String s = JOptionPane.showInputDialog(f,"InputDialog");
}
}

```

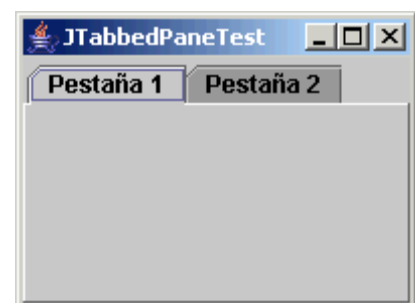


3.25 javax.swing.JTabbedPane

```

import javax.swing.*;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
public class JTabbedPaneTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
    }
}

```



```

f.setTitle("JTabbedPaneTest");
f.setSize(200,150);
JTabbedPane tabbedPane = new JTabbedPane();
JPanel panel1 = new JPanel();
tabbedPane.addTab("Pestaña 1", panel1);
JPanel panel2 = new JPanel();
tabbedPane.addTab("Pestaña 2", panel2);
f.getContentPane().add(tabbedPane);
f.setVisible(true);
}
}

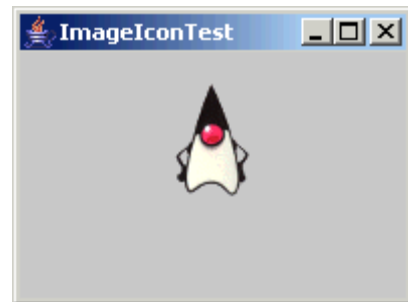
```

3.26 javax.swing.ImageIcon

```

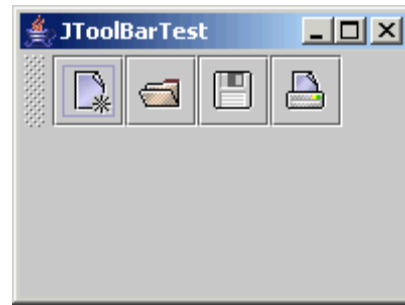
import java.awt.FlowLayout;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class ImageIconTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("ImageIconTest");
        f.setSize(200,150);
        f.getContentPane().setLayout(new FlowLayout());
        JLabel l = new JLabel();
        l.setIcon(new ImageIcon("duke.gif")); //Soporta formatos GIF, JPG y PNG.
        f.getContentPane().add(l);
        f.setVisible(true);
    }
}

```



3.27 javax.swing.JToolBar

```
import java.awt.*;
import javax.swing.*;
public class JToolBarTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("JToolBarTest");
        f.setSize(200,150);
        JToolBar tb = new JToolBar();
        JButton b = new JButton(new ImageIcon("New24.gif"));
        tb.add(b);
        b = new JButton(new ImageIcon("Open24.gif"));
        tb.add(b);
        b = new JButton(new ImageIcon("Save24.gif"));
        tb.add(b);
        b = new JButton(new ImageIcon("Print24.gif"));
        tb.add(b);
        f.getContentPane().add(tb,BorderLayout.NORTH);
        f.setVisible(true);
    }
}
```



Nota: Galería de iconos para el Java Look & Feel "Metal" en <http://java.sun.com/developer/techDocs/hi/repository/>

3.28 javax.swing.JSplitPane

```
import java.awt.Dimension;
import javax.swing.*;
public class JSplitPaneTest
{
    public static void main(String[] args)
    {
```



```

JFrame f = new JFrame();
f.setTitle("JSplitPaneTest");
f.setSize(275,252);
JLabel l1 = new JLabel(new ImageIcon("argh.jpg"));
l1.setMinimumSize(new Dimension(20, 20));
JLabel l2 = new JLabel(new ImageIcon("comic.jpg"));
l2.setMinimumSize(new Dimension(20, 20));
JSplitPane sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,l1,l2);
sp.setContinuousLayout(true);
sp.setOneTouchExpandable(true);
sp.setDividerLocation(100);
f.getContentPane().add(sp);
f.setVisible(true);
}
}

```

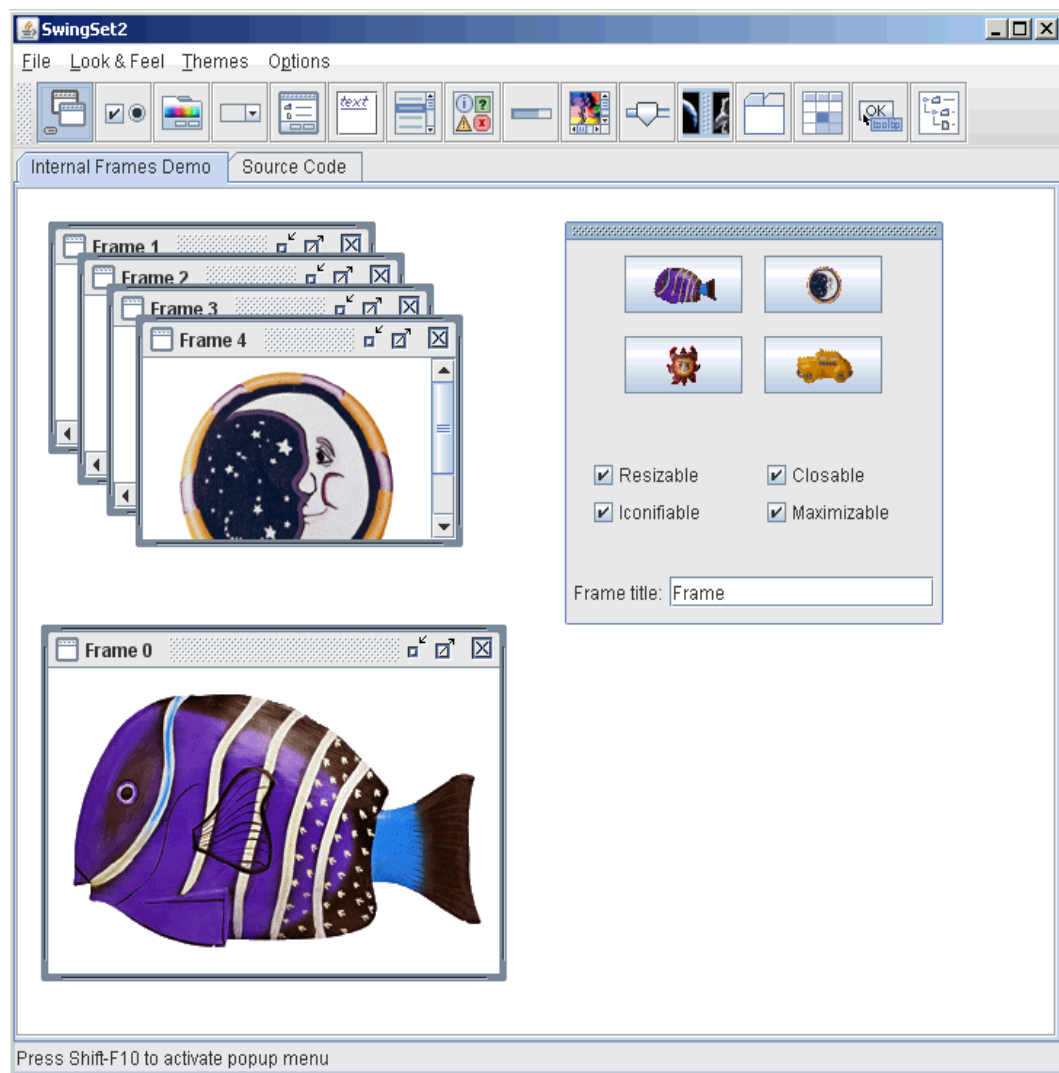


3.29 Aplicación de ejemplo

La JDK incluye una aplicación de ejemplo de la Swing donde se pueden ver y probar directamente todas sus capacidades, además de tener acceso al código fuente que implementa lo que se está probando. Se llama SwingSet2.

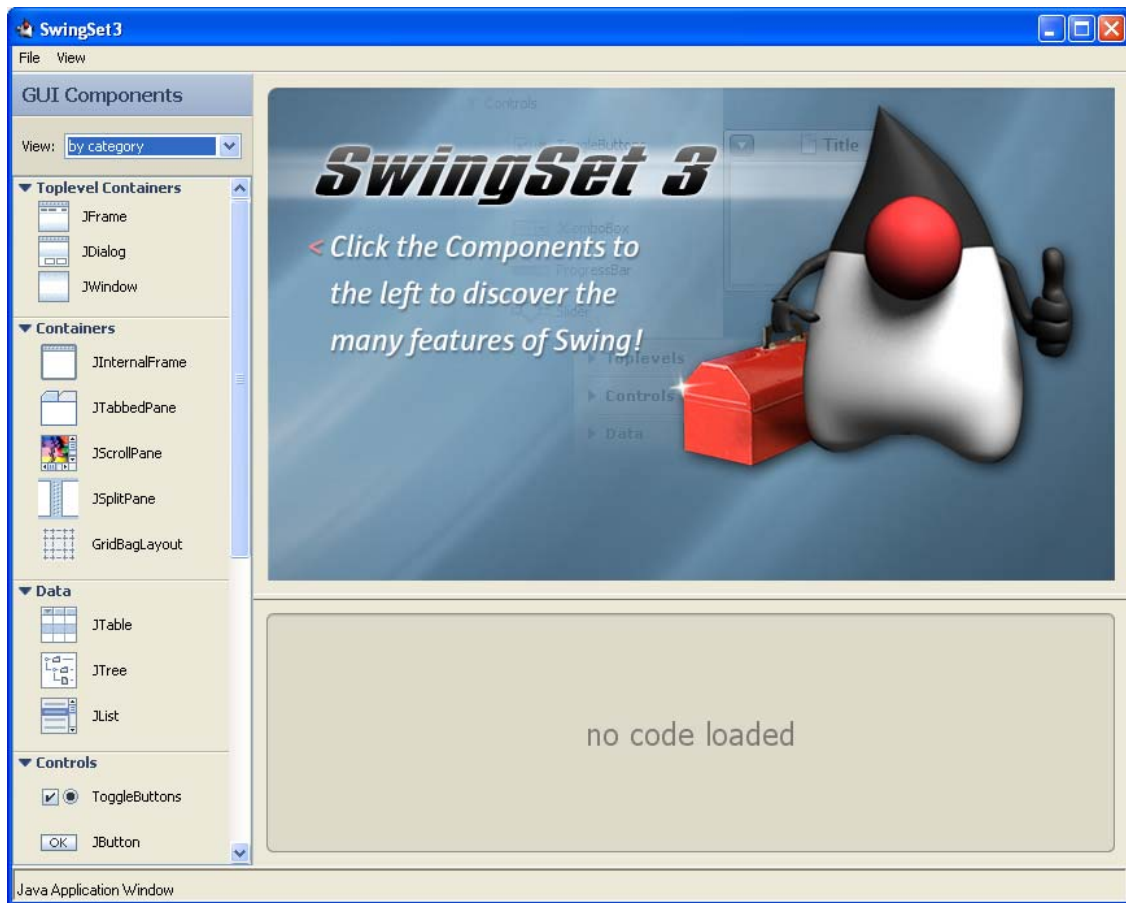
Se puede arrancar mediante el siguiente comando:

```
java -jar SwingSet2.jar
```



Con la aparición de la JDK 6.0 Update 10, se desarrolló una nueva versión de esta aplicación denominada SwingSet3 que a diferencia de SwingSet2 no está incluida con el instalable de la JDK (al menos de momento). Está accesible en la web en la siguiente URL:

<http://download.java.net/javadesktop/swingset3/SwingSet3.jnlp>



4. Layout Managers

Todos los contenedores Swing tienen asociado un `LayoutManager` para coordinar el tamaño y la situación de sus componentes. Por ejemplo:

- `JPanel` tiene asociado un `FlowLayout`
- `JFrame` tiene asociado un `BorderLayout`

Cada Layout se caracteriza por el estilo que emplea para situar los componentes en su interior:

- Alineación de izquierda a derecha.
- Alineación en rejilla.
- Alineación del frente a atrás.

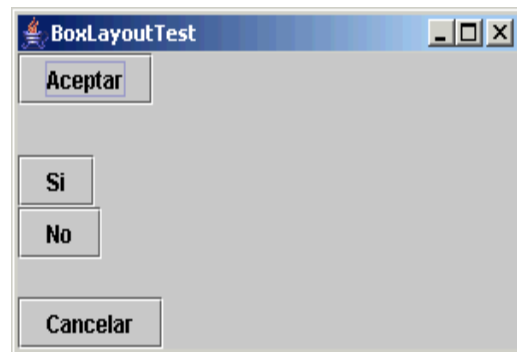
En la Swing siguen teniendo vigencia todos los Layout Managers que vimos en la AWT. Pero se añade alguno nuevo:

4.1 javax.swing.BoxLayout

Sitúa los componentes en línea vertical u horizontal. Respeta sus tamaños.

La clase `javax.swing.Box` tiene métodos para crear zonas con espacio como `createVerticalStrut(int)` y zonas que absorban los espacios como `createVerticalGlue(int)`.

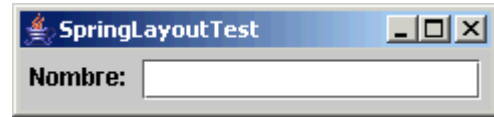
```
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
public class BoxLayoutTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("BoxLayoutTest");
        f.setSize(300,150);
        f.getContentPane().setLayout(
            new BoxLayout(f.getContentPane(),BoxLayout.Y_AXIS));
        f.getContentPane().add(new JButton("Aceptar"));
        f.getContentPane().add(Box.createVerticalStrut(25));
        f.getContentPane().add(new JButton("Si"));
        f.getContentPane().add(new JButton("No"));
        f.getContentPane().add(Box.createVerticalGlue());
        f.getContentPane().add(new JButton("Cancelar"));
        f.setVisible(true);
    }
}
```



4.2 javax.swing.SpringLayout

Permite definir la relación (distancia) entre los límites de los distintos controles.

```
import java.awt.Container;
import javax.swing.*.*;
public class SpringLayoutTest
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame();
        f.setTitle("SpringLayoutTest");
        Container container = f.getContentPane();
        SpringLayout layout = new SpringLayout();
        container.setLayout(layout);
        JLabel label = new JLabel("Nombre: ");
        JTextField text = new JTextField(15);
        f.getContentPane().add(label);
        f.getContentPane().add(text);
        layout.putConstraint(SpringLayout.WEST, label, 5, SpringLayout.WEST,
        container);
        layout.putConstraint(SpringLayout.NORTH, label, 5, SpringLayout.NORTH,
        container);
        layout.putConstraint(SpringLayout.WEST, text, 5, SpringLayout.EAST,
        label);
        layout.putConstraint(SpringLayout.NORTH, text, 5, SpringLayout.NORTH,
        container);
        layout.putConstraint(SpringLayout.EAST, container, 5,
        SpringLayout.EAST, text);
        layout.putConstraint(SpringLayout.SOUTH, container, 5,
        SpringLayout.SOUTH, text);
        f.pack();
        f.setVisible(true);
    }
}
```



4.3 ScrollPaneLayout y ViewportLayout

Utilizados internamente por Swing para algunos de los componentes como el ScrollPane.

5. Gestión de eventos

La Gestión de Eventos en Swing es exactamente igual que en AWT. La única diferencia es que existen familias de eventos nuevas, y por tanto, interfaces que implementar nuevos.

Los nuevos eventos están enumerados en la siguiente tabla:

AncestorEvent	Un padre se ha añadido, movido o eliminado.
CaretEvent	El cursor en un texto ha cambiado.
ChangeEvent	El estado ha cambiado.
DocumentEvent	Los atributos de un Document han cambiado, se ha insertado o se ha eliminado contenido.
HyperlinkEvent	Se ha activado, entrado o salido de un hyperlink.
InternalFrameEvent	Se ha activado, cerrado, desactivado, minimizado, maximizado o abierto un internal frame.
ListDataEvent	Ha cambiado el contenido.
ListSelectionEvent	Ha cambiado la selección en una lista.
MenuDragMouseEvent	El ratón se ha arrastrado, entrado, salido, soltado en un menú.
MenuEvent	Se ha seleccionado o deseleccionado un menú.
MenuKeyEvent	Se ha pulsado, soltado o tecleado sobre un menú.
PopupMenuEvent	Se ha mostrado, ocultado o seleccionado un menú emergente.
TableColumnModelEvent	Se ha añadido, eliminado, movido, redimensionada o seleccionada una columna.
TableModelEvent	El modelo de la tabla ha cambiado.
TreeExpansionEvent	Se ha abierto o cerrado el árbol.
TreeModelEvent	Se ha cambiado, añadido o eliminado un elemento del

	árbol.
TreeSelectionEvent	Ha cambiado la selección en el árbol.
UndoableEditEvent	Se ha realizado una operación que no se puede deshacer.

Y ¿qué componente puede notificar de qué evento? Para contestar a esta pregunta, añadimos la siguiente tabla:

Componente Swing	Listener
AbstractButton DefaultButtonModel JComboBox JFileChooser JTextField Timer	ActionListener
JScrollBar	AdjustmentListener
JComponent	AncestorListener
JTextComponent	CaretListener
DefaultCellEditor DefaultTreeCellEditor	CellEditorListener
AbstractButton DefaultBoundedRangeModel DefaultButtonModel DefaultCaret DefaultColorSelectionModel DefaultSingleSelectionModel JProgressBar JSlider JTabbedPane JViewport MenuSelectionManager StyleContext StyleContext.NamedStyle	ChangeListener
DefaultTableColumnModel	ColumnModelListener
AbstractDocument DefaultStyledDocument	DocumentListener
JEditorPane	HyperlinkListener
JInternalFrame	InternalFrameListener
AbstractButton	ItemListener

DefaultButtonModel JComboBox	
AbstractListModel	ListDataListener
DefaultListSelectionModel JList	ListSelectionListener
JMenuItem	MenuDragMouseListener
JMenuItem	MenuKeyListener
JMenu	MenuListener
JPopupMenu	PopupMenuListener
AbstractAction DefaultTreeSelectionModel JComponent SwingPropertyChangeSupport TableColumn UIDefaults UIManager	PropertyChangeListener
AbstractTableModel	TableModelListener
JTree	TreeExpansionListener
DefaultTreeModel	TreeModelListener
DefaultTreeSelectionModel JTree	TreeSelectionListener
AbstractDocument UndoableEditSupport	UndoableEditListener
JComponent	VetoableChangeListener

En la siguiente tabla se enumeran todos los métodos (su nombre es auto explicativo) de los distintos interfaces:

Listener interface	Métodos
AncestorListener	ancestorAdded ancestorMoved ancestorRemoved
CaretListener	caretUpdate
CellEditorListener	editingCanceled editingStopped

ChangeListener	stateChanged
DocumentListener	changedUpdate insertUpdate removeUpdate
HyperlinkListener	hyperlinkUpdate
InternalFrameListener	internalFrameActivated internalFrameClosed internalFrameClosing internalFrameDeactivated internalFrameDeiconified internalFrameIconified internalFrameOpened
ListDataListener	contentsChanged intervalAdded intervalRemoved
ListSelectionListener	valueChanged
MenuDragMouseListener	menuDragMouseDragged menuDragMouseEntered menuDragMouseExited menuDragMouseReleased
MenuKeyListener	menuKeyPressed menuKeyReleased menuKeyTyped
MenuListener	menuCanceled menuDeselected menuSelected
MouseListener	mouseClicked mouseDragged mouseEntered mouseExited mouseMoved mousePressed mouseReleased
PopupMenuListener	popupmenuCanceled popupMenuWillBecomeInvisible

	popupMenuWillBecomeVisible
TableColumnModelListener	columnAdded columnMarginChanged columnMoved columnRemoved columnSelectionChanged
TableModelListener	tableChanged
TreeExpansionListener	treeCollapsed treeExpanded
TreeModelListener	treeNodesChanged treeNodesInserted treeNodesRemoved treeStructureChanged
TreeSelectionListener	valueChanged
UndoableEditListener	undoableEditHappened

6. Look & Feel

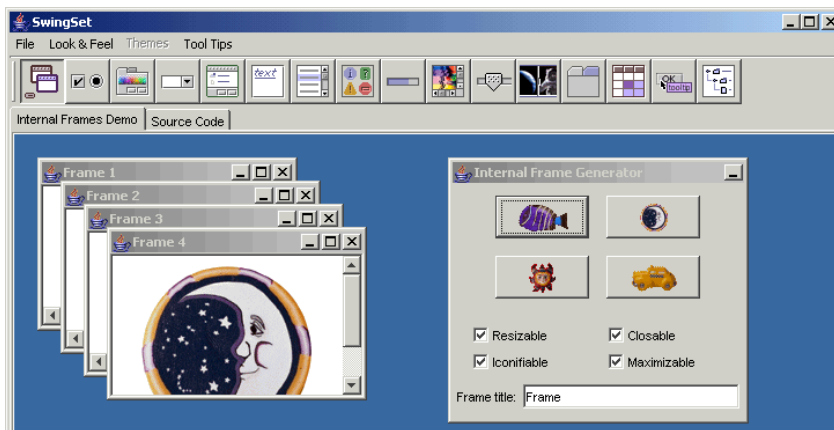
Recordemos que en la introducción de esta unidad, comentamos que una de las diferencias entre la AWT y la Swing era que los componentes visuales de esta última eran "*lightweight*" (ligeros), es decir, que se pintaban ellos mismos pixel a pixel en la pantalla.

Esto permite la existencia del concepto de "*Look & Feel*", es decir, la apariencia visual de un mismo componente en pantalla.

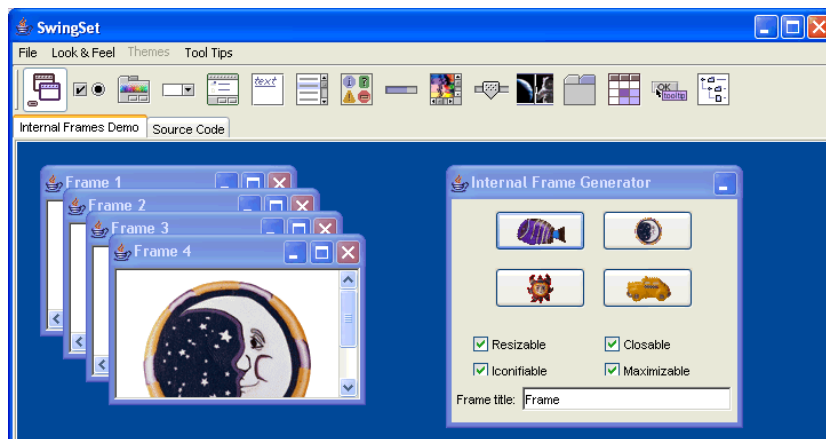
En la propia Swing, existen los siguientes Look & Feel (aunque en Internet se pueden encontrar muchos mas):

6.1 Windows

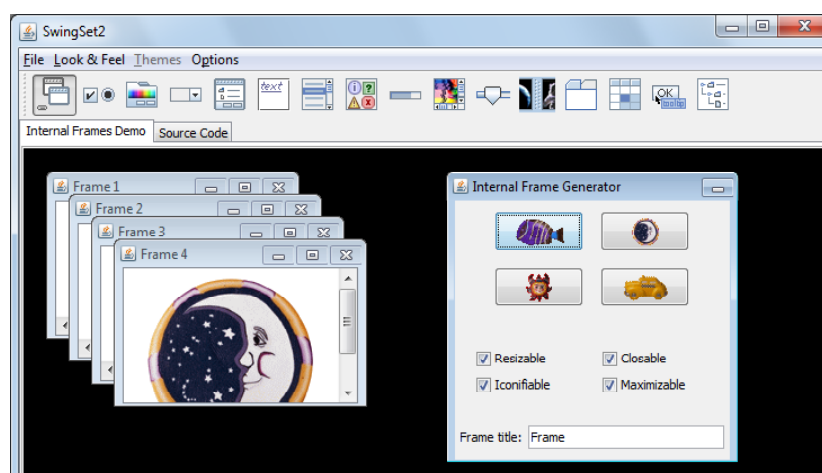
A la mayoría os será muy conocido. Dependiendo de la versión de Windows será uno u otro. Por ejemplo en Windows NT:



O en Windows XP:



O en Windows 7:



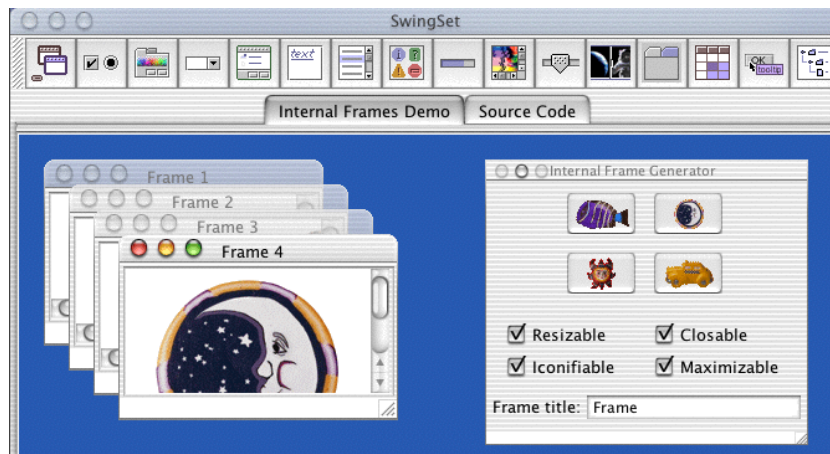
La clase raíz de esta implementación es:

`com.sun.java.swing.plaf.windows.WindowsLookAndFeel`

Por temas legales, la JVM solo permite el uso de este Look & Feel en plataformas de Microsoft.

6.2 Macintosh

Este Look & Feel conocido con el nombre de Aqua, es el de MacOS.



La clase raíz de esta implementación es:

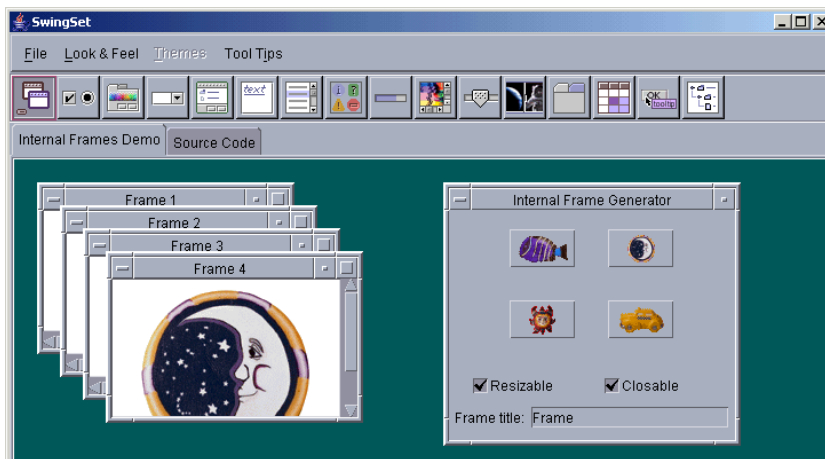
`com.sun.java.swing.plaf.mac.MacLookAndFeel`

Por temas legales, la JVM solo permite el uso de este Look & Feel en plataformas de Macintosh.

6.3 Unix

Para todos los sabores de Unix: Linux, Solaris, AIX, HP-UX, etc... existen dos versiones distintas.

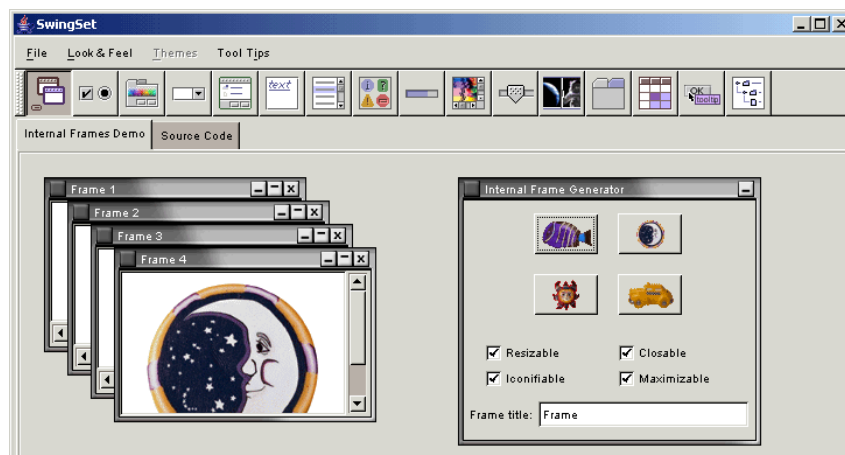
Motif:



La clase raíz de esta implementación es:

`com.sun.java.swing.plaf.motif.MotifLookAndFeel`

GTK:



La clase raíz de esta implementación es:

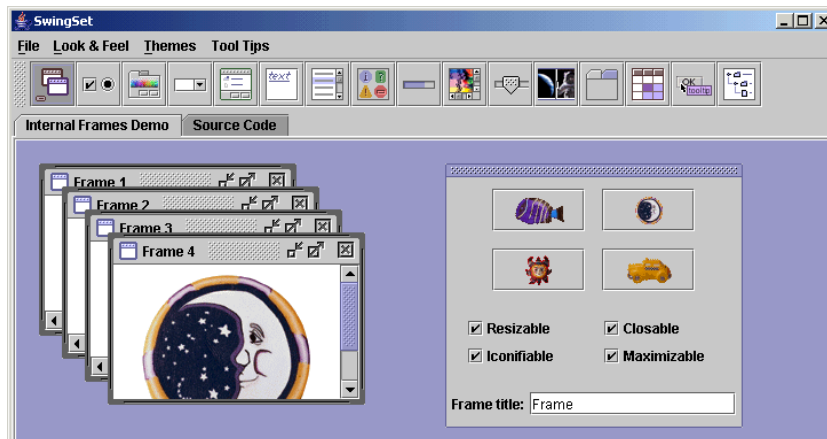
`com.sun.java.swing.plaf.gtk.GTKLookAndFeel`

A diferencia de lo que ocurría con los de Windows o Macintosh, en este caso no hay ninguna restricción de uso. Se pueden usar en cualquier plataforma aunque no sean Unix.

6.4 Java

Conocido también con el nombre de Metal. Es independiente de plataforma, es decir, su apariencia visual no tiene nada que ver con ninguna de los sistemas operativos conocidos.

Es el Look & Feel utilizado por defecto.

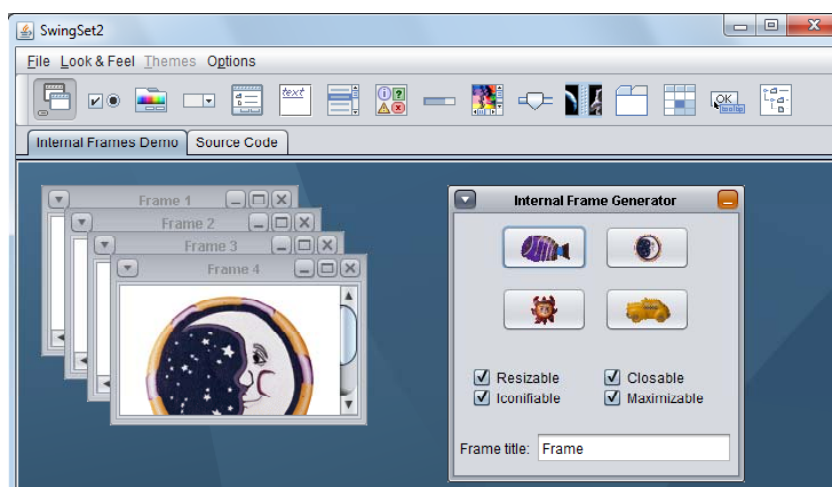


La clase raíz de esta implementación es:

`javax.swing.plaf.metal.MetalLookAndFeel`

6.5 Nimbus

Con la JDK 6.0 Update 10, se introdujo un Look & Feel nuevo como modernización del de Java. Al igual que este, también es independiente de plataforma, es decir, su apariencia visual no tiene nada que ver con ninguna de los sistemas operativos conocidos.



La clase raíz de esta implementación es:

com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel

Existen distintas alternativas para seleccionar el Look & Feel de una aplicación Java:

- Por línea de comando:

```
java -Dswing.defaultlaf=com.sun.java.swing.plaf.motif.MotifLookAndFeel
MiApp
```

- Mediante el fichero `swing.properties` (localizado en el directorio `\lib` del JRE):

```
# Swing properties
swing.defaultlaf=com.sun.java.swing.plaf.gtk.GTKLookAndFeel
```

- Por código de forma estática (al inicio del programa):

```
try
{
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
}
catch (Exception ex) { }
```

- Por código de forma dinámica:

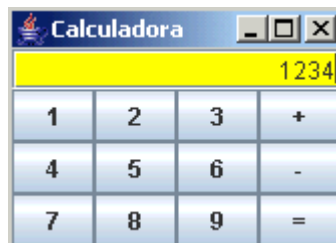
```
try
{
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
    SwingUtilities.updateComponentTreeUI(frame);
    frame.pack();
}
catch (Exception ex)
{
}
```

Nota: siendo `frame`, el contenedor raíz de la aplicación.



para practicar

PRÁCTICA A: Desarrollar, tanto con la JDK de Oracle como con Eclipse, una calculadora mediante el framework Swing. El resultado debe ser algo parecido a:



Implementar las siguientes funcionalidades:


- Operación de suma, resta e igualdad.
- Tecleo de números mediante el ratón.
- Cierre de la aplicación mediante el botón de cerrado del frame.
- Evitar el tecleo de caracteres no numéricos en el text field.
- Los números deben estar alineados a la derecha.

Nota 1: en esta ocasión se ha utilizado una clase anónima para implementar el listener de eventos de ventana.

Nota 2: la mayoría de constantes de Swing se encuentran en la clase `javax.swing.SwingConstants`.

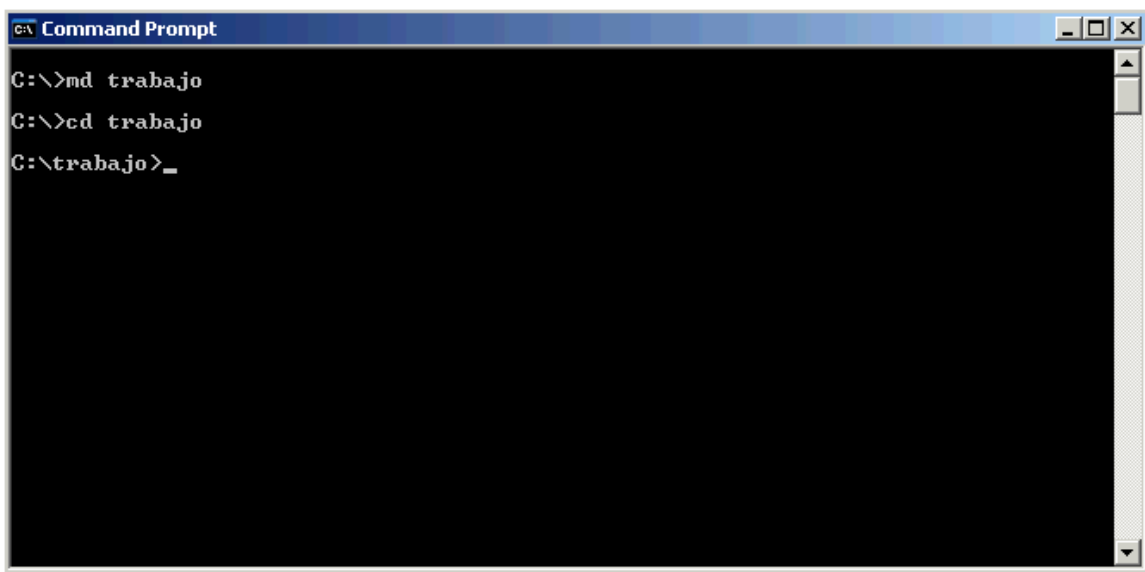
Solución con la JDK de Oracle:

En el caso de que no esté puesta la variable de entorno PATH, abrir una sesión DOS y ajustar la variable de entorno PATH para que el Sistema Operativo sepa encontrar las herramientas del JDK. Para respetar el valor que ya tuviese la variable PATH le añadimos %PATH%.



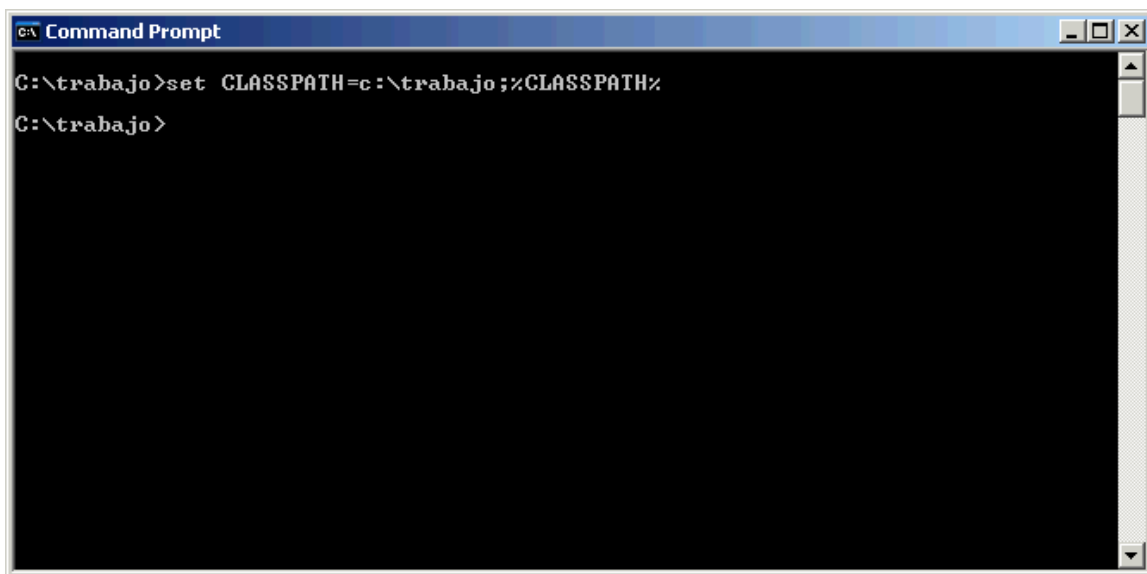
```
C:\>set PATH=c:\jdk1.6.0\bin;%PATH%
C:\>_
```

Creamos un directorio de trabajo donde guardar el programa Java.



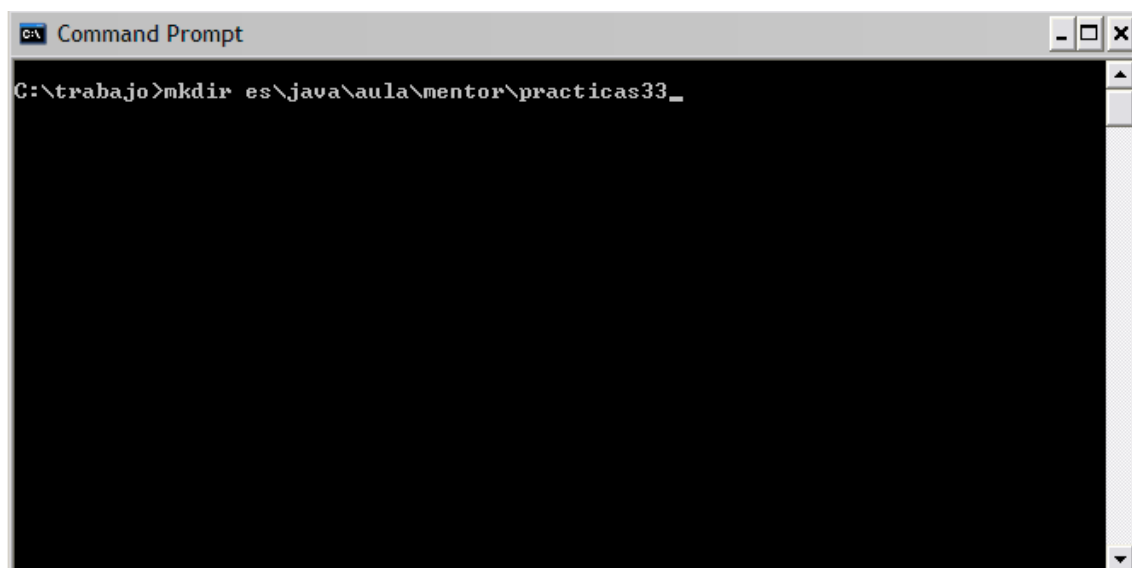
```
C:\>md trabajo
C:\>cd trabajo
C:\trabajo>_
```

Ajustar la variable de entorno CLASSPATH para que las herramientas del JDK sepan encontrar nuestras clases Java. Tenemos dos opciones, o añadir el . (punto) y siempre ejecutar las herramientas en el directorio donde se encuentre el código, o añadir el directorio de trabajo y ejecutar las herramientas donde queramos. Para respetar el valor que ya tuviese la variable CLASSPATH le añadimos %CLASSPATH%



```
C:\> Command Prompt
C:\trabajo>set CLASSPATH=c:\trabajo;%CLASSPATH%
C:\trabajo>
```

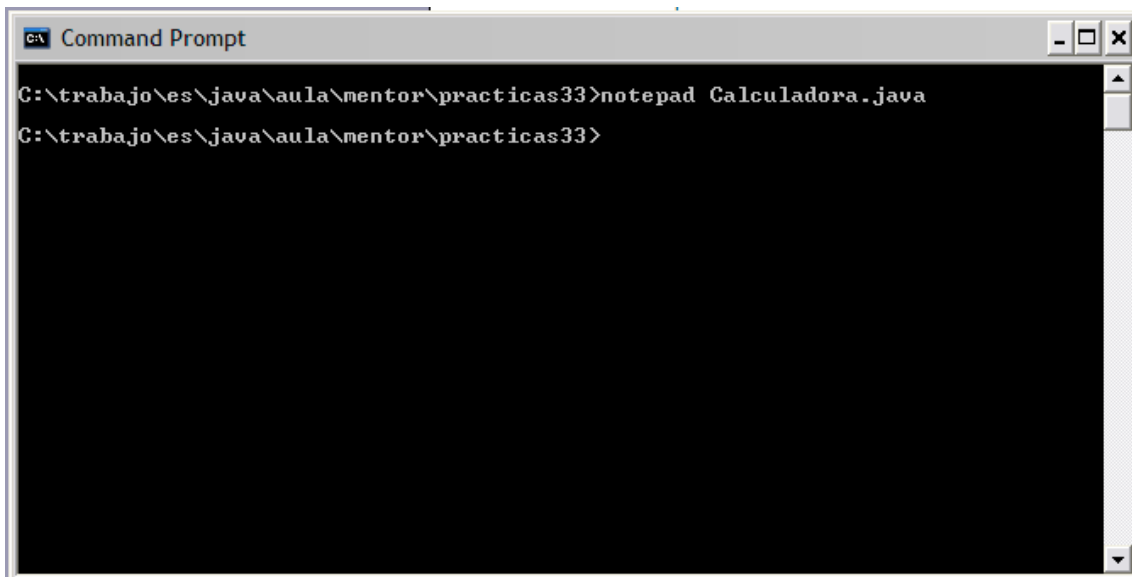
Creamos el directorio donde va a estar nuestra clase
es\java\aula\mentor\practicas33



```
C:\> Command Prompt
C:\trabajo>mkdir es\java\aula\mentor\practicas33_
```

Desde el directorio es\java\aula\mentor\practicas33, con un editor de texto (por ejemplo Notepad) vamos a escribir el código fuente de nuestra clase java; el nombre del fichero de la clase debe ser exactamente igual (incluyendo mayúsculas y minúsculas) al de la clase Java que vamos a desarrollar.

Creamos la clase Calculadora.java



Quedaría como sigue:

```

Calculadora.java - Notepad
File Edit Format View Help
package es.java.aula.mentor.practicass3;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;

public class Calculadora extends JFrame implements ActionListener, KeyListener
{
    private JTextField t = null;
    private int oper1 = 0; // Primer operando.
    private int oper2 = 0; // Segundo operando.
    private String command = null; // Operación.
    private boolean sw = true; // Switch para añadir o borrar dígitos.

    public Calculadora()
    {
        this.setTitle("calculadora");
        this.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent ev)
            {
                System.exit(0);
            }
        });
        this.getContentPane().setLayout(new BorderLayout());

        t = new JTextField(0);
        t.setBackground(Color.YELLOW);
        t.setHorizontalAlignment(SwingConstants.RIGHT);
        t.addKeyListener(this);
        JPanel aux = new JPanel();
        aux.setLayout(new GridLayout(3,4));

        JButton b = null;
        b = new JButton("1");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("2");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("3");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("+");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("4");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("5");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("6");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("-");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("7");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("8");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("9");
        b.addActionListener(this);
        aux.add(b);
        b = new JButton("=");
        b.addActionListener(this);
        aux.add(b);

        this.getContentPane().add(t, BorderLayout.NORTH);
        this.getContentPane().add(aux, BorderLayout.CENTER);
    }

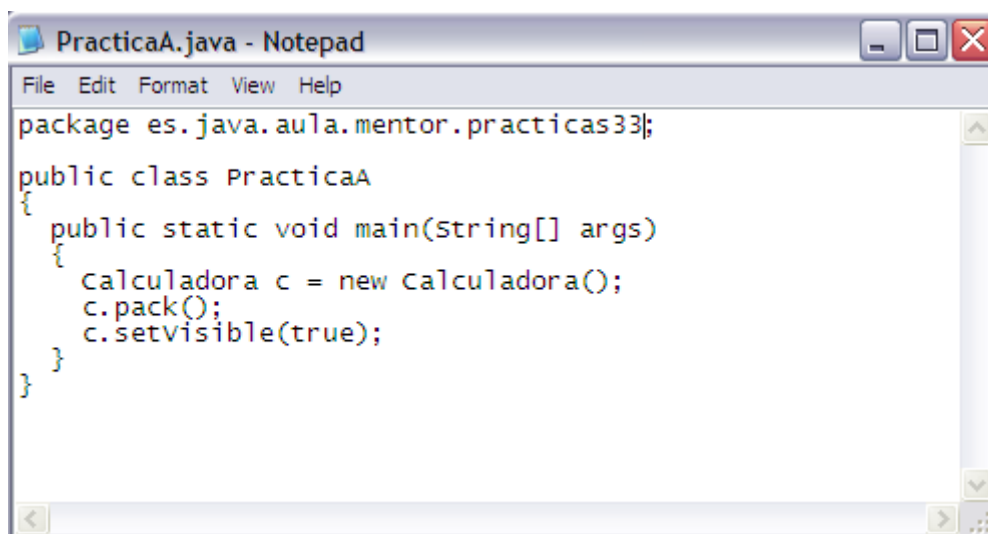
    public void actionPerformed(ActionEvent ev) // Se ha producido un Action Event
    {
        String aux = ((JButton)ev.getSource()).getText();
        if(aux.equals("+") || aux.equals("-")) {
            command = aux;
            oper1 = Integer.parseInt(t.getText());
            sw = true;
        } else if(aux.equals("=")) {
            oper2 = Integer.parseInt(t.getText());
            if(command.equals("+")) {
                t.setText(new Integer(oper1 + oper2).toString());
                oper1 = oper1 + oper2; // Permite teclear un número y "+" (repetir)
            } else {
                t.setText(new Integer(oper1 - oper2).toString());
                oper1 = oper1 - oper2; // Permite teclear un número y "-" (repetir)
            }
            sw = true;
        } else {
            if(sw) {
                t.setText(aux);
                sw = false;
            } else {
                t.setText(t.getText() + aux);
            }
        }
    }

    public void keyPressed(KeyEvent ev) { // Se ha producido un Key Event de
    public void keyReleased(KeyEvent ev) // Se ha producido un Key Event de
    {
        char tmp = ev.getKeyChar();
        if(!Character.isDigit(tmp)) {
            t.setText(t.getText().substring(0, t.getText().length() - 1));
            t.setCaretPosition(t.getText().length()); // Posicionar el cursor al final del texto
        }
    }

    public void keyTyped(KeyEvent ev) { // Se ha producido un Key Event de

```

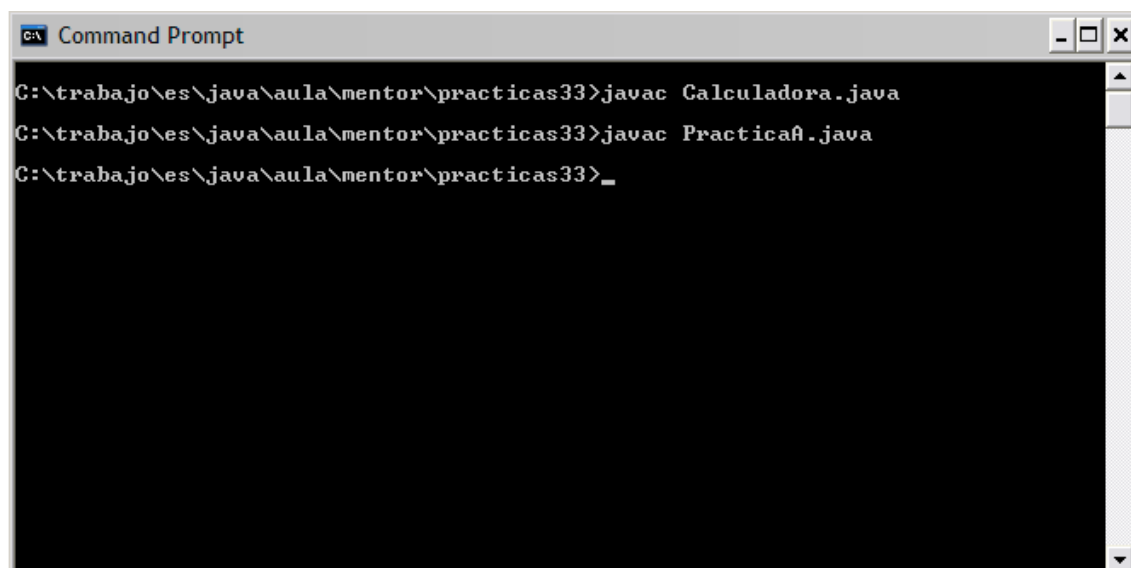
Nos creamos la clase PracticaA que va a contener nuestro main



```
PracticaA.java - Notepad
File Edit Format View Help
package es.java.aula.mentor.practicas33;

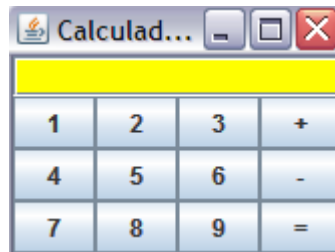
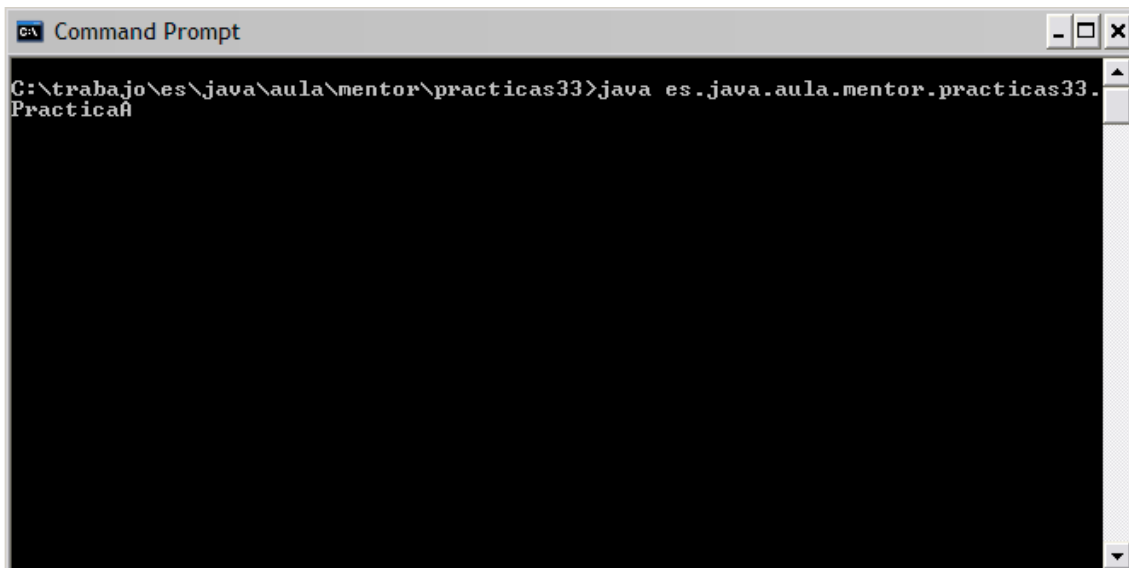
public class PracticaA
{
    public static void main(String[] args)
    {
        calculadora c = new Calculadora();
        c.pack();
        c.setVisible(true);
    }
}
```

Compilamos nuestras clases



```
Command Prompt
C:\trabajo\es\java\aula\mentor\practicas33>javac Calculadora.java
C:\trabajo\es\java\aula\mentor\practicas33>javac PracticaA.java
C:\trabajo\es\java\aula\mentor\practicas33>_
```

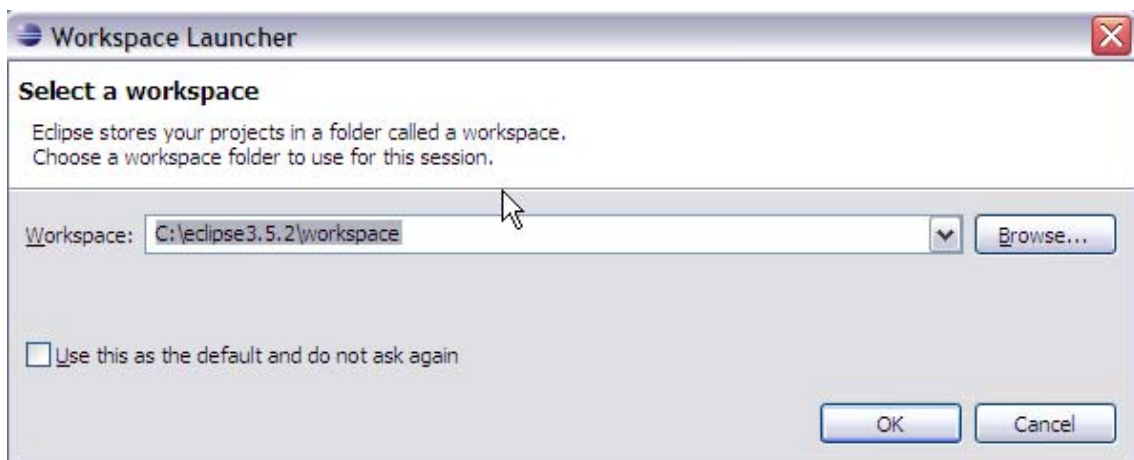
Ejecutamos nuestra práctica



Solución con Eclipse:

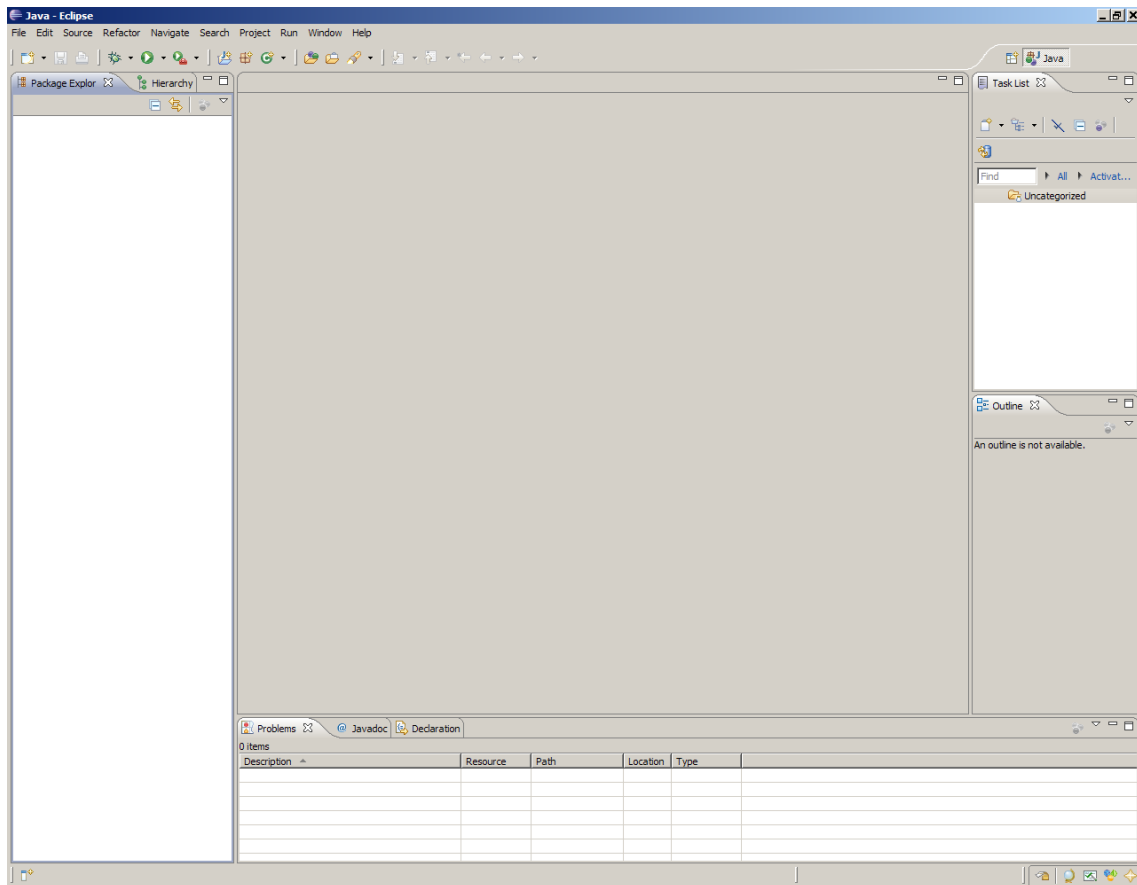
Arrancar Eclipse, ejecutando C:\eclipse3.7.1\eclipse.exe

Seleccionar la ubicación del "workspace" (o área de trabajo).

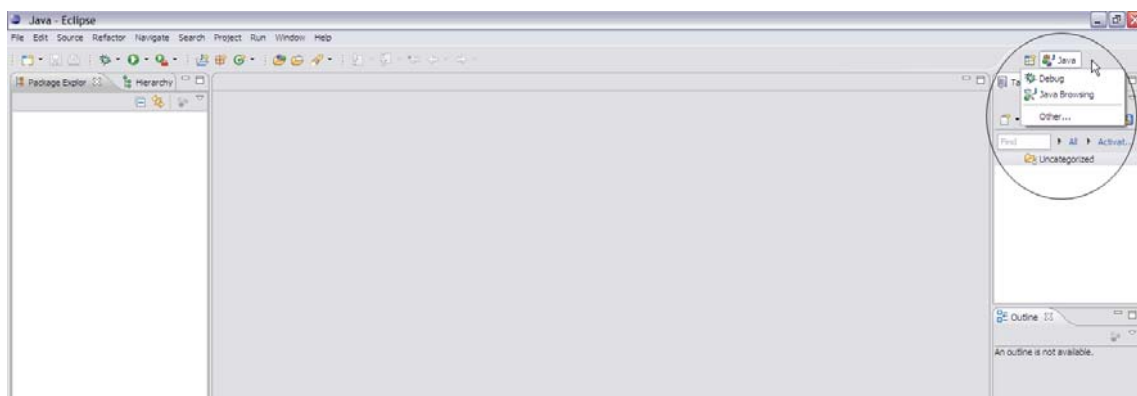


Aparecerá la pantalla para empezar a trabajar.

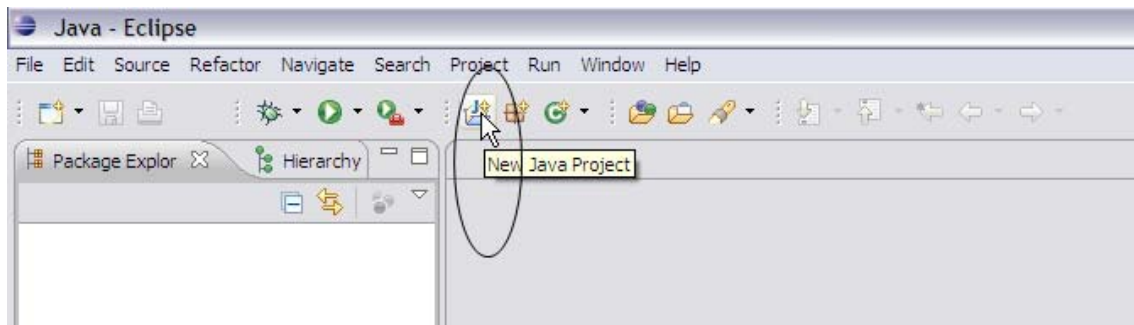
MODULO 1 Interfaces Visuales



Verificar que la perspectiva Java está abierta, y sino cambiar a ella

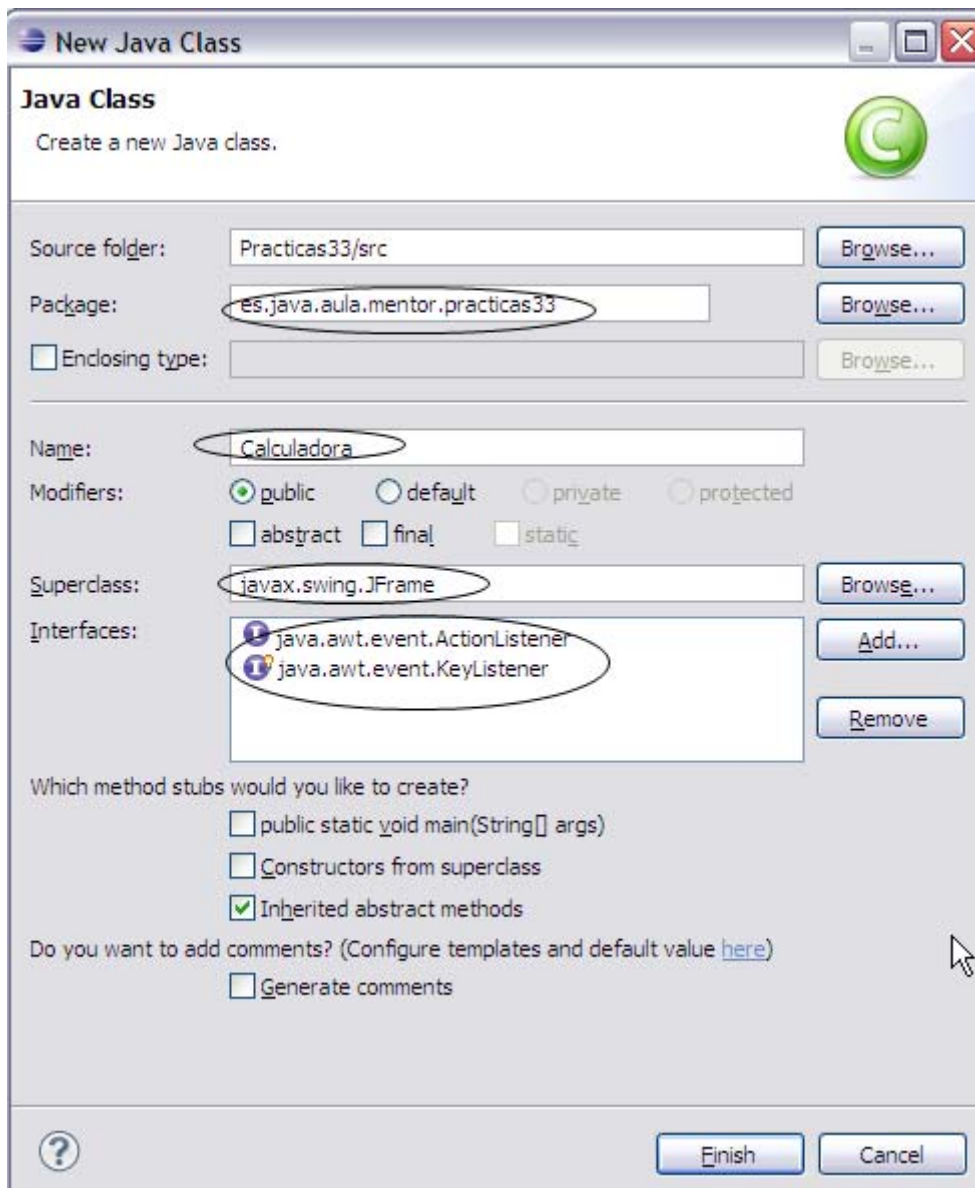


Crear un proyecto nuevo de nombre Practicas33

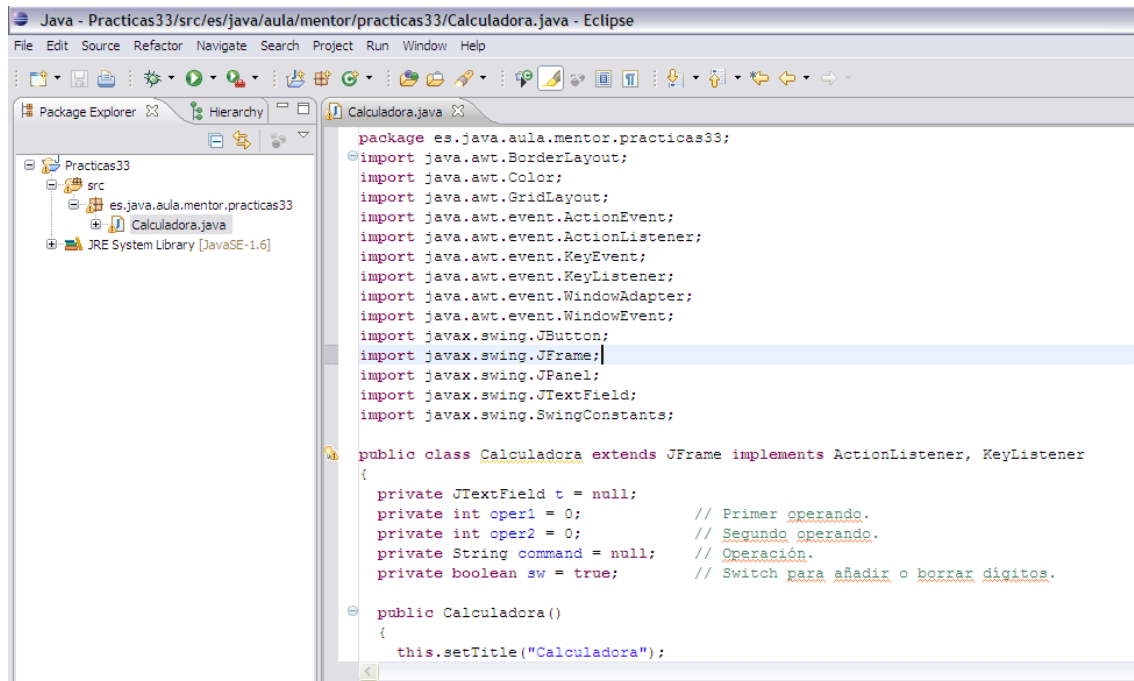


Darle el nombre y seleccionar Finish.

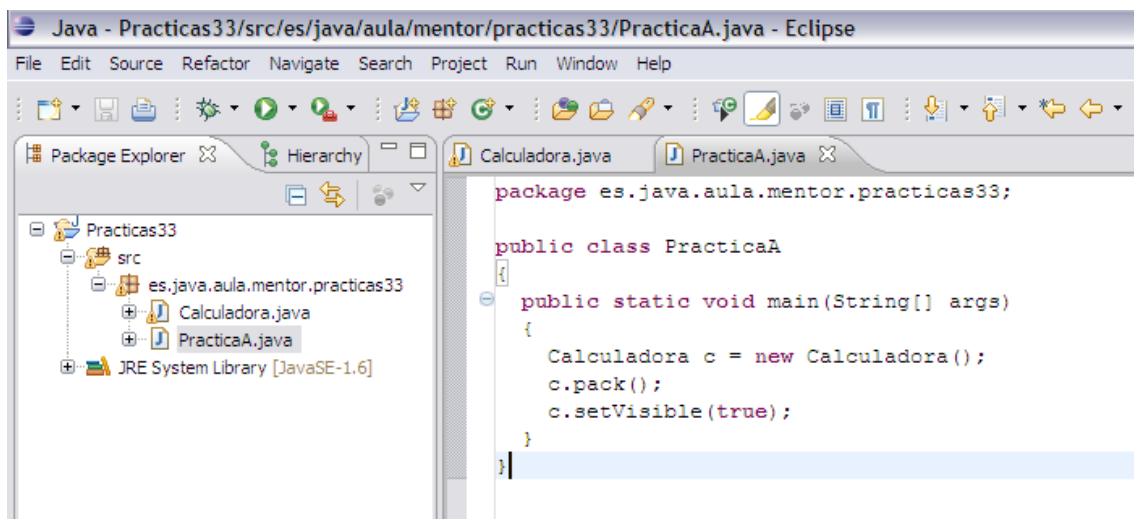
Creamos la clase Calculadora, en el paquete `es.java.aula.mentor.practicas33`, indicando que heredamos de la clase `javax.swing.JFrame` y que implemente los interfaces `java.awt.event.ActionListener` y `java.awt.event.KeyListener`



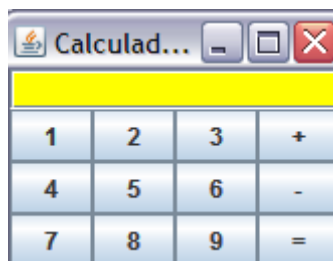
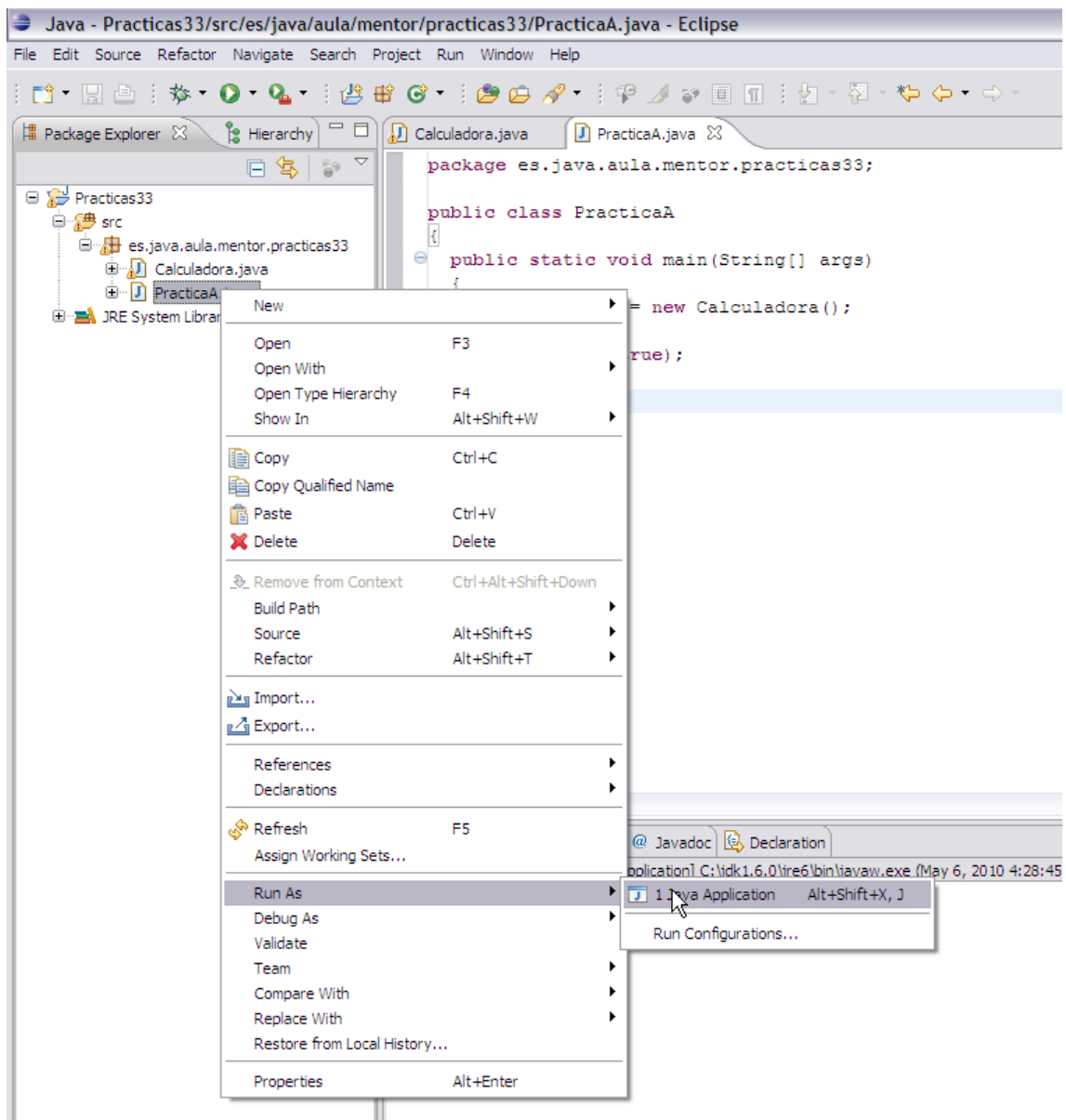
El contenido de la clase es el mismo que el desarrollado con la JDK:



Nos creamos la clase PracticaA, que contendrá el método main



Para ejecutar la aplicación, situarnos encima de la clase PracticaA y con el botón derecho del ratón seleccionar Run As -> Java Application





para recordar

Java Foundation Classes (JFC) es un conjunto de clases para mejorar el soporte al desarrollo de GUIs.

Se creó como una extensión de la AWT añadiendo las siguientes características:

- Componentes Swing.
- Soporte de Look & Feel.
- API de accesibilidad.
- Java 2D API.
- Soporte de Drag & Drop.

Para diferenciar los componentes Swing de los AWT, sus nombres están precedidos por una 'J'. Y todas las clases se encuentran en el paquete `javax.swing`.*

Swing sigue trabajando con los conceptos de la AWT:

- Contenedores.
- Componentes.
- LayoutManagers.
- Eventos.

Aunque evidentemente, lo que si hace es añadir nuevos componentes visuales, nuevos Layout Managers y nuevas familias de eventos.

EL Look & Feel de un componente es su apariencia visual. Swing, al ser "*light-weight*" implementa este concepto soportando seleccionar el Look & Feel de una aplicación.

Applets

Tema 1.3

Índice de la unidad:

1. Introducción
2. Las clases Applet y JApplet
3. Html
4. Ejecutando Applets Java
5. Seguridad

En la siguiente Unidad, vamos a estudiar los Applets Java, qué son, cómo se implementan, cómo se utilizan y una serie de conceptos y características relacionadas con la seguridad como son la firma y los certificados digitales utilizados en Internet.

1. Introducción

Un Applet Java es un tipo de aplicación Java que se inserta dentro de las páginas HTML. Cuando estas páginas se descargan, los Applets se ejecutan en el navegador.

Los Applets Java se cargan de la siguiente forma:

- Se escribe una URL a la página HTML en el navegador.
- El navegador carga la página HTML.
- El navegador interpreta el contenido de la página HTML, y al encontrar la existencia de un Applet Java, carga sus clases Java.
- Se ejecuta el Applet Java.

Los Applets Java, a diferencia de las aplicaciones convencionales Java vistas hasta ahora, no se ejecutan siempre con el conocimiento del usuario. Podemos estar navegando tranquilamente por Internet, y cargar un Applet Java sin saberlo. Por ello, los Applets Java sufren una serie de restricciones de seguridad inexistentes en las aplicaciones Java convencionales.

Por tanto, los navegadores implementan un entorno de ejecución seguro habitualmente conocido con el nombre de SandBox. No obstante, existen técnicas de firma digital para poder evitar estas restricciones.

Algunas restricciones son:

- Realizar llamadas a programas externos.
- Realizar operaciones de entrada/salida.
- Realizar llamadas a métodos nativos (JNI).

- Abrir conexiones con servidores remotos distintos al origen del propio Applet Java.

De esta manera, un Applet Java jamás podrá dañar o robar información del sistema.

Aunque más adelante, en esta misma unidad trataremos un poco más el tema de la firma digital en los Applets Java, en la siguiente URL el lector podrá encontrar mucha más información si quiere profundizar en el tema de la seguridad y el lenguaje Java:

<http://download.oracle.com/javase/tutorial/security/index.html>

2. Las clases Applet y JApplet

Las clases Applets y JApplet son un contenedor visual (`java.awt.Container`) del estilo de los Frame y Panel o los JFrame y JPanel.

Como el lector ya habrá intuido, existen dos implementaciones distintas:

- AWT
- Swing

La implementación AWT se llama Applet, y se encuentra en el paquete `java.applet`.*

Hereda de la clase `java.awt.Panel`, con lo que se le aplican los `LayoutManagers` y se le añaden componentes visuales como ya hiciéramos con los contenedores estándar AWT. De igual forma, la gestión de eventos es exactamente igual.

La implementación Swing se llama JApplet, y se encuentra en el paquete `javax.swing`.*

Hereda de su clase homónima en AWT, `java.applet.Applet` así que por tanto, una vez más se le aplican los `LayoutManagers` y se le añaden componentes visuales como ya hiciéramos con los contenedores estándar AWT. De igual forma, la gestión de eventos es exactamente igual.

2.1 El ciclo de vida de un Applet Java

A diferencia de las aplicaciones Java convencionales, los Applets Java no tienen un método `main`: `public static void main(String[] args)` que como ya sabemos sirve para arrancar la aplicación.

Por el contrario tienen estos cinco métodos:

- `public void init();`
- `public void start();`
- `public void stop();`
- `public void destroy();`
- `public void paint(Graphics g);`

El método `init()` es el primer método que se ejecuta una vez que se ha llamado al constructor del Applet. Solo se ejecuta una vez en la vida del Applet y suele utilizarse para la inicialización del mismo.

El método `start()` se ejecuta una vez haya terminado la ejecución del método `init()`. A diferencia del método `init()`, `start()` se ejecuta cada vez que haya que arrancar el Applet (recarga de la página HTML, maximizar el navegador después de haberlo minimizado,...) y suele utilizarse para arrancar procesos (por ejemplo, Threads que se verán con más detalle en la Unidad 3.7).

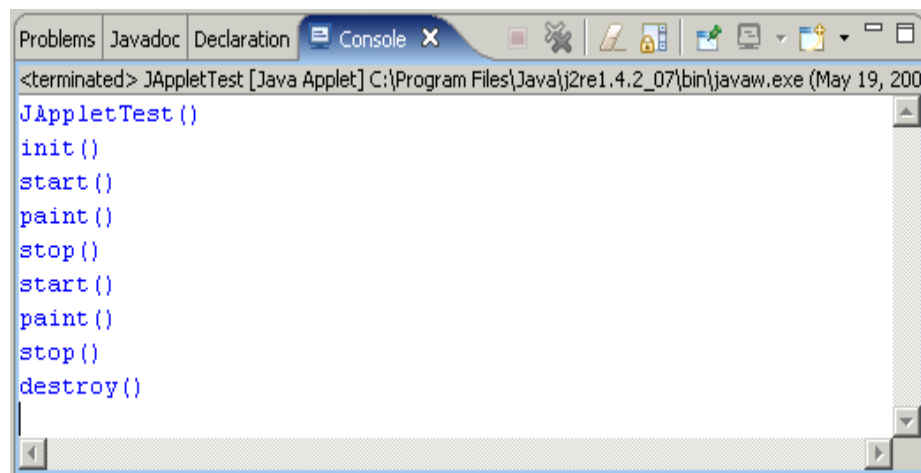
El método `stop()` se ejecuta cada vez que haya que parar el Applet (recarga de la página HTML, minimizar el navegador, cargar otra página,...).

El método `destroy()` se ejecuta al cerrar el navegador o cuando este decide eliminarle (por ejemplo, en el purgado de caché). Al igual que ocurría con el método `init()`, también se ejecuta solo una vez en la vida del Applet y suele utilizarse para liberar cualquier recurso utilizado.

El método `paint(Graphics g)` recibe una instancia de la clase `java.awt.Graphics` como ya ocurriese con los componentes visuales AWT y Swing, que puede ser utilizada para escribir o pintar en el Applet directamente. Este método se ejecuta cada vez que el navegador necesita refrescar el Applet (por ejemplo, recarga de la página HTML, maximizar el navegador después de haberlo minimizado,...).

- **Ejemplo:** Cuando se carga un Applet Java en el navegador se ejecutan los siguientes métodos en este orden: se ejecuta el constructor por defecto (constructor sin parámetros) del Applet, se ejecuta el método `init()`, se ejecuta el método `start()` y por último se ejecuta el método `paint(Graphics g)`.
- **Ejemplo:** Cuando con el navegador vamos a otra página HTML nueva y volvemos a la anterior que tenía el Applet Java cargado se ejecutan los siguientes métodos en este orden: se ejecuta el método `stop()` al salir, y se ejecutan los métodos `start()` y `paint(Graphics g)` al volver.
- **Ejemplo:** Cuando cerramos el navegador, se ejecutan los siguientes métodos en este orden: se ejecuta el método `stop()` y acto seguido el método `destroy()`.
- **Ejemplo:** A continuación, vemos un ejemplo en ejecución:

```
import java.awt.Graphics;
import javax.swing.JApplet;
public class JAppletTest extends JApplet
{
    public JAppletTest() { System.out.println("JAppletTest()"); }
    public void init() { System.out.println("init()"); }
    public void start() { System.out.println("start()"); }
    public void paint(Graphics g) { System.out.println("paint()"); }
    public void stop() { System.out.println("stop()"); }
    public void destroy() { System.out.println("destroy()"); }
}
```

2.2 Otros métodos de Applet

Los Applets Java tienen otros métodos específicos para leer los parámetros que se les pase desde la página HTML (se verá como hacer esto en el siguiente apartado), para cargar recursos (por ejemplo, música e imágenes) desde Internet, etc...

- **public String getParameter(String name);** Se utiliza para acceder a un parámetro especificado en la página HTML por su nombre. Como se puede observar, el valor de dicho parámetro es devuelto como String con lo que habrá que hacer las conversiones necesarias si realmente es un parámetro de tipo número, una fecha, etc...
- **public String[][] getParameterInfo();** Se utiliza para acceder a todos los parámetros especificados en la página HTML. Devuelve un array bidimensional donde la primera dimensión contiene el nombre del parámetro y la segunda dimensión su valor.
- **public URL getDocumentBase();** Se utiliza para acceder a la URL desde donde se cargó la página HTML que contiene al Applet Java.
- **public AudioClip getAudioClip(URL url, String clip);** Se utiliza para acceder a un archivo de música desde Internet. Soporta los formatos Sun Audio (*.au), Windows Wave (*.wav), Macintosh AIFF (*.aif o *.aiff extensions) y Musical Instrument Digital Interface - MIDI (*.mid o *.rmi).

- **public Image getImage(URL url, String img);** Se utiliza para acceder a una imagen desde Internet. Soporta los formatos GIF, JPEG y PNG.
- **Ejemplo:** Applet Java que accede a un archivo de música en Internet y lo ejecuta (hace sonar):

```
import java.applet.Applet;
import java.applet.AudioClip;
public class AudioClipTest extends Applet
{
    AudioClip clip = null;
    public void init()
    {
        clip = this.getAudioClip(this.getDocumentBase(),"audio/test.au");
    }
    public void start() { clip.play(); }
    public void stop() { clip.stop(); }
}
```

- **Ejemplo:** Applet Java que accede a una imagen en Internet y la muestra:

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;
public class ImageTest extends Applet
{
    Image img = null;
    public void init()
    {
        img = this.getImage(this.getDocumentBase(),"images/test.gif");
    }
    public void paint(Graphics g)
    {
        g.drawImage(img, 10,20, this);
    }
}
```

3. HTML

Las siglas HTML significan en inglés HiperText Markup Language.

Se trata de un lenguaje muy simple, utilizado para crear documentos Web de extensión *.html o *.htm que se limita a describir la estructura y el contenido de un documento.

Actualmente se encuentra en su versión 4.01

Es un lenguaje basado en etiquetas, donde cada etiqueta se escribe entre los caracteres: < > (menor que y mayor que). Estas etiquetas pueden contener atributos que modifican o complementan el significado de la etiqueta.

Las etiquetas suelen ir formando bloques: <> y </>. El carácter / se utiliza para indicar que la etiqueta en este caso es de cierre de bloque. En el caso de utilizar una etiqueta que no va a ser un bloque sino aislada, se sigue utilizando el carácter / al final (por ejemplo,
).

El lenguaje HTML no es case-sensitive, es decir, ignora el uso de minúsculas o mayúsculas.

- **Ejemplo:** El esqueleto básico de una página HTML sería algo así:

```
<HTML>
  <HEAD>
    <TITLE>Mi primer HTML</TITLE>
  </HEAD>
  <BODY>

  </BODY>
</HTML>
```

Como se puede observar en el código de ejemplo unos bloques van conteniendo a otros bloques.

3.1 La etiqueta HTML <APPLET>

La etiqueta HTML para definir e incluir un Applet Java es <APPLET>.

Su sintaxis es la siguiente:

```
<APPLET [name=?] [codebase=?] code=? [archive=?] width=?
height=?
```

```

    [align=?] [vspace=?] [hspace=?] [alt=?]>
    [<PARAM name=? value=?/>]
    .....
</APPLET>

```

donde cada etiqueta y atributo significa lo siguiente:

- **name:** especifica un nombre al Applet.
- **codebase:** especifica la URL por defecto del Applet en caso de que sea distinta a la de la página HTML.
- **code:** especifica el nombre de la clase principal del Applet, incluyendo el paquete y la extensión (por ejemplo, eud.test.MiApplet.class).
- **archive:** especifica los archivos JAR o ZIP que contengan clases necesarias y que deban ser precargados por el navegador.
- **width:** especifica el ancho del Applet en píxeles.
- **height:** especifica el alto del Applet en píxeles.
- **align:** especifica el alineamiento del Applet respecto del texto. Algunos valores válidos son: left, right, top, middle...
- **vspace:** especifica el espacio a dejar por encima y por debajo del Applet en píxeles.
- **hspace:** especifica el espacio a dejar por derecha e izquierda del Applet en píxeles.
- **alt:** especifica el texto a mostrar en caso de que el navegador no sea capaz de mostrar el Applet (por ejemplo porque en la máquina no haya instalada una JVM, o porque el navegador se haya configurado para inhabilitar la ejecución de código Java).
- **<PARAM>:** es la manera de pasar parámetros al Applet desde el exterior. Esta etiqueta tiene dos atributos: *name* para el nombre del parámetro y *value* para su valor.

- **Ejemplo:** Uso de la etiqueta <APPLET>:

```
<HTML>
<HEAD>
  <TITLE>Mi primer HTML</TITLE>
</HEAD>
<BODY>
  <APPLET name="Test1" codebase="."
    code="edu.test.MiApplet.class"
    archive="miApplet.jar" width="200" height="300" align="middle"
    vspace="5" hspace="5" alt="No tiene soporte de Applets Java">
    <PARAM name="param1" value="value1">
    <PARAM name="parma2" value="value2">
  </APPLET>
  <APPLET code= "edu.test.OtroApplet.class" width="200" height="300">
</APPLET>
</BODY>
</HTML>
```

3.2 El fichero JAR

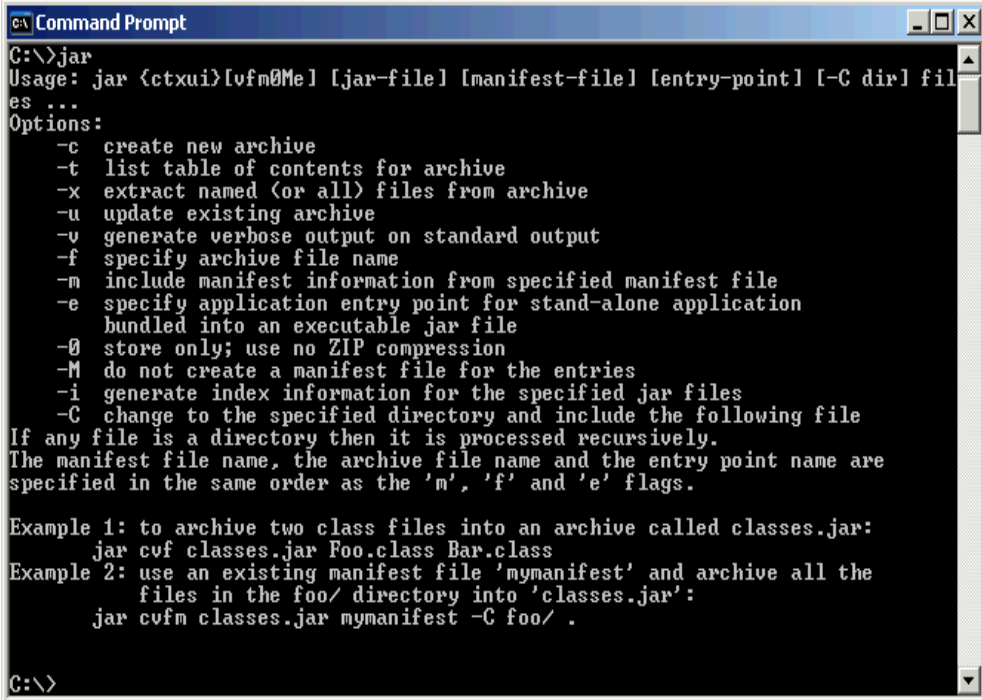
Los ficheros JAR (Java ARchive) sirven para empaquetar (y opcionalmente comprimir) clases Java y otros recursos utilizados por el programa Java (imágenes, archivos de sonido, etc...).

La JDK (Java Development Kit) contiene una herramienta para crear, listar o expandir archivos JAR: jar.exe

Repasemos su sintaxis:

```
jar.exe {ctxu} [vfmOM] [fichero-jar] [fichero-manifest] [-C dir]
ficheros....
```

- **Ejemplo:**



```

C:\>jar
Usage: jar {ctxui}[vfm0Me] [jar-file] [manifest-file] [entry-point] [-C dir] file
es ...
Options:
  -c create new archive
  -t list table of contents for archive
  -x extract named (or all) files from archive
  -u update existing archive
  -v generate verbose output on standard output
  -f specify archive file name
  -m include manifest information from specified manifest file
  -e specify application entry point for stand-alone application
    bundled into an executable jar file
  -0 store only; use no ZIP compression
  -M do not create a manifest file for the entries
  -i generate index information for the specified jar files
  -C change to the specified directory and include the following file
If any file is a directory then it is processed recursively.
The manifest file name, the archive file name and the entry point name are
specified in the same order as the 'm', 'f' and 'e' flags.

Example 1: to archive two class files into an archive called classes.jar:
    jar cvf classes.jar Foo.class Bar.class
Example 2: use an existing manifest file 'mymanifest' and archive all the
    files in the foo/ directory into 'classes.jar':
    jar cvfm classes.jar mymanifest -C foo/ .

C:\>

```

4. Ejecutando Applets Java

Existen distintas alternativas para probar o ejecutar un Applet Java:

- Utilidad Applet Viewer de la JDK (Java Development Kit).
- Navegador Web.

4.1 Applet Viewer

La JDK contiene una herramienta para probar los Applets Java sin necesidad de utilizar un navegador. Se llama Applet Viewer (appletviewer.exe).

Esta herramienta necesita un fichero HTML como argumento:

appletviewer.exe HelloWorld.html

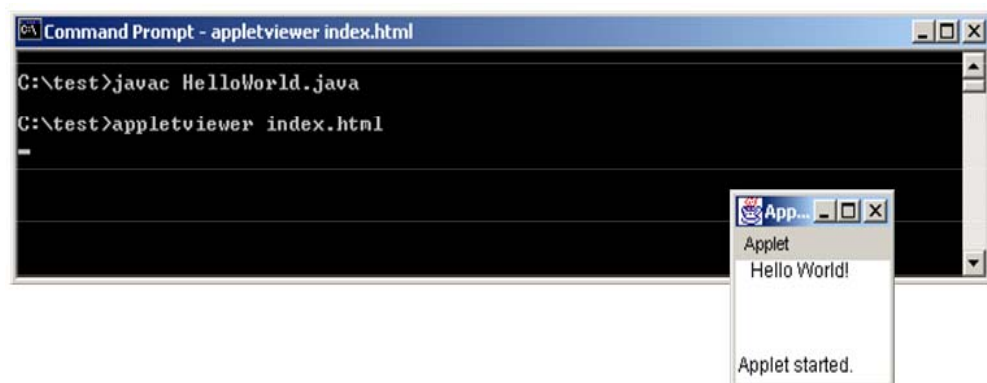
Dicho fichero no tiene porque ser una página HTML completa, de hecho, Applet Viewer ignora cualquier etiqueta distinta de <APPLET>. Eso sí, requiere un mínimo imprescindible que es:

```
<HTML>
  <APPLET code= "HelloWorld.class" width="200" height="300">
  </APPLET>
</HTML>
```

- **Ejemplo:** Desarrollo y prueba de un Applet Java con la JDK:

```
import java.applet.Applet;
import java.awt.Graphics;
public class HelloWorld extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Hello World!", 10, 10);
    }
}
```

```
<HTML>
  <APPLET code="HelloWorld.class" width="100" height="50">
  </APPLET>
</HTML>
```



4.2 Navegador Web

Adicionalmente, como ya hemos comentado, también se puede utilizar un Navegador Web para probar nuestros Applets Java.

Al instalar la JDK o el JRE en nuestras máquinas, automáticamente se detecta qué navegadores hay instalados añadiendo el soporte Java. Están soportados los navegadores más habituales:

- Firefox
- Mozilla
- Internet Explorer
- Chrome
- Opera
- etc...

Basta con cargar la página HTML que contiene el Applet Java en el navegador, bien desde el sistema de ficheros, por ejemplo:

```
file:///c:/test/index.html
```

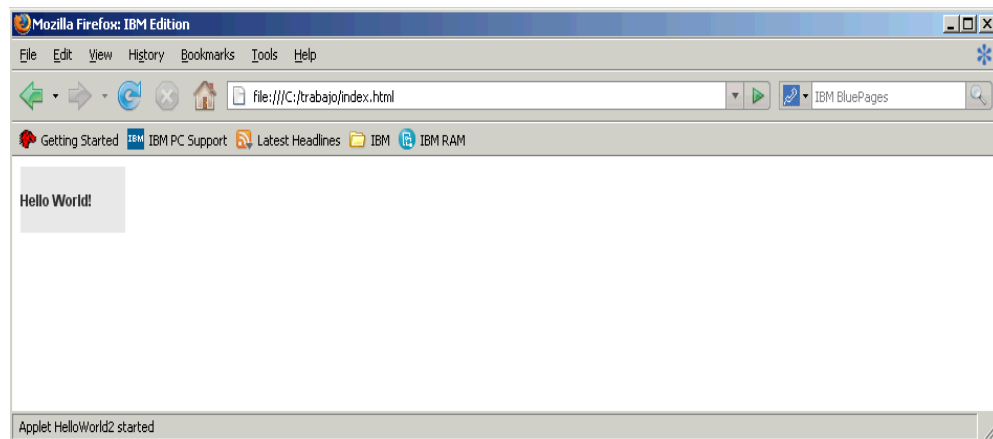
o bien desde una URL de Internet, por ejemplo:

```
http://centrovirtual.educacion.es/test/index.html
```

- **Ejemplo:** Desarrollo y prueba de un Applet Java con un Navegador Web:

```
import javax.swing.JApplet;
import javax.swing.JLabel;
public class HelloWorld2 extends JApplet
{
    public void init()
    {
        JLabel l = new JLabel("Hello World!");
        this.getContentPane().add(l);
    }
}
```

```
<HTML>
  <APPLET code="HelloWorld2.class" width="100" height="50">
  </APPLET>
</HTML>
```

5. Seguridad

Aunque no es el objetivo de este curso profundizar en temas de seguridad, si hemos creído conveniente al menos dar unas primeras pinceladas sobre el tema para que el lector pueda decidir si profundizar o no en un futuro, y al menos tener unas nociones básicas del funcionamiento de la seguridad y algunos de sus mecanismos como la firma digital, los certificados digitales, etc... en el mundo de los Applets Java.

Ya hemos comentado, que un Applet Java se ejecuta en un entorno restringido, de forma que no tiene acceso a los recursos de la máquina donde se ejecuta. Esto es debido a que la ejecución de los Applets Java se realiza sin el conocimiento y aprobación de los usuarios.

Evidentemente, estas restricciones hacen que el uso de los Applets Java se reduzca a pequeñas aplicaciones multimedia que enriquezcan de alguna manera el contenido de una página HTML.

Para evitar esto, existen los conceptos de certificados y firmas digitales.

- **Ejemplo:** Un Applet Java que intenta leer el archivo hosts del sistema para poder conocer otras máquinas de nuestra red.

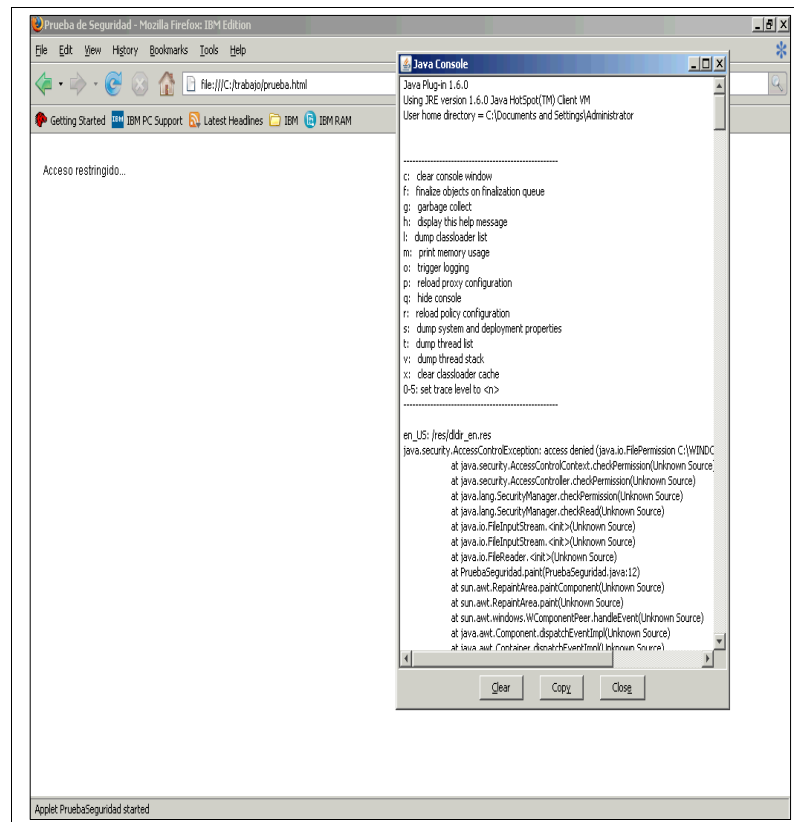
Nota: La unidad donde se explicará entrada/salida está en la Unidad 3.5. No preocuparse por no entender el siguiente código puesto que se trata de un ejemplo para mostrar temas relacionados con la seguridad y el código del ejemplo no es lo importante.

```

import java.applet.Applet;
import java.awt.Graphics;
import java.io.*;
public class PruebaSeguridad extends Applet
{
    public void paint(Graphics g)
    {
        BufferedReader br = null;
        try
        {
            br = new BufferedReader(new FileReader
                ("C:\\WINDOWS\\system32\\drivers\\etc\\hosts"));
            StringBuffer buffer = new StringBuffer();
            String temp = null;
            while((temp = br.readLine()) != null)
                buffer.append(temp);
            g.drawString("Lectura realizada...",10,30);
            br.close();
        }
        catch(IOException ex)
        {
            g.drawString("Problemas de I/O...",10,30);
        }
        catch(SecurityException ex)
        {
            ex.printStackTrace();
            g.drawString("Acceso restringido...",10,30);
        }
    }
}

<HTML>
<HEAD>
    <TITLE>Prueba de Seguridad</TITLE>
</HEAD>
<BODY>
    <APPLET code="PruebaSeguridad.class" archive="prueba.jar" width="300"
        height="50">
    </APPLET>
</BODY>
</HTML>

```



Evidentemente, el Applet Java no puede acceder.

5.1 La firma digital

La firma digital es un mecanismo por el que el autor de un contenido puede decir y asegurar de alguna manera que es suyo y que él es el autor. De esta forma, quien recibe dicho contenido, puede ver el autor y decidir si confía o no en él, dando luz verde a la ejecución de dicho contenido (en el caso de un Applet Java).

La firma digital se basa en algoritmos de cifrado de clave asimétrica, es decir, existe una pareja de claves: pública y privada. A diferencia de los algoritmos de cifrado de clave simétrica, estos tienen una clave para cifrar y otra clave para descifrar.

El mecanismo de la firma digital, consiste en:

- Firmar el contenido (en el caso de los Applets Java cuando decimos contenido realmente nos referimos a uno o varios ficheros JAR con el código

Java a ejecutar) con la clave privada del autor. La clave privada solo la conoce el autor. Firmar significa aplicar un algoritmo hash sobre el código y el resultado, cifrarlo con la clave privada, obteniendo así la firma digital.

- Enviamos el código firmado junto con la clave pública.
- En la recepción, se comprueba la firma digital. Comprobar una firma digital, significa aplicar el mismo algoritmo hash sobre el código y comparar el resultado con la firma digital (después de haberla descifrado, esta vez, con la clave pública).

Normalmente, la clave pública se envía en un certificado digital para validar la identidad del firmante. Trataremos el tema de los certificados digitales un poco más adelante.

Desde un punto de vista práctico, para firmar digitalmente código Java, hay que seguir los siguientes pasos:

- Crear ficheros JAR con todo el código compilado.
- Generar un par de claves (pública y privada) así como un certificado digital con la clave pública si no lo hemos hecho ya.
- Firmar digitalmente los ficheros JAR con la clave privada.

En la JDK tenemos todo lo necesario para llevar a cabo estos pasos.

La herramienta para generar la pareja de claves pública y privada se llama: `keytool.exe`

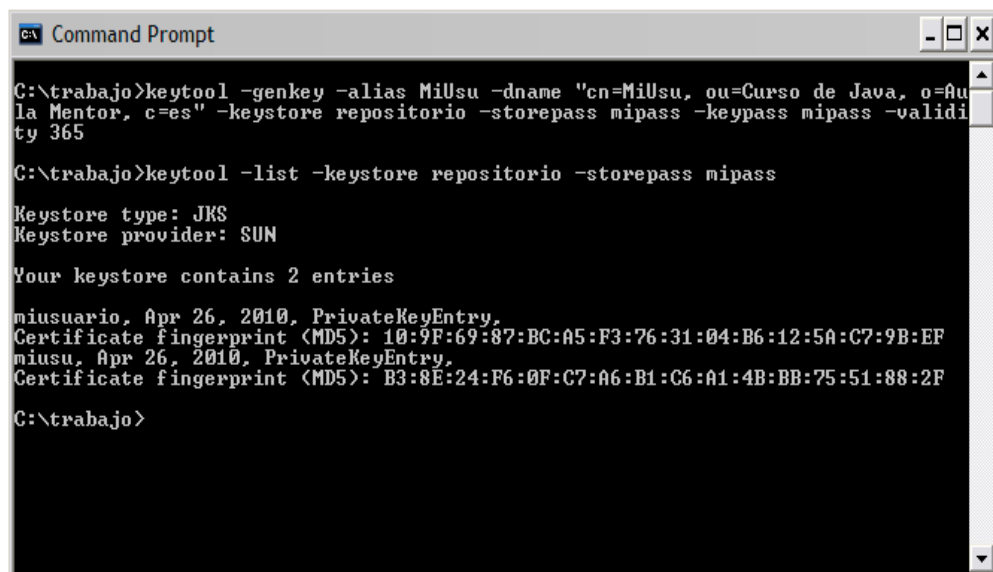
Esta herramienta utiliza distintos parámetros. En nuestro caso usaremos:

- `-genkey`: comando para generar las claves.
- `-dname`: con el *distinguished name* del propietario de las claves.
- `-alias`: con el identificador a usar para referirse a las claves.

- -keystore: con el repositorio (es un fichero) donde se va a guardar toda la información generada.
- -storepass: con la contraseña de acceso al repositorio.
- -keypass: con la contraseña para utilizar las claves.
- -validity: con el número de días de validez del certificado.

No obstante también puede utilizar estos otros:

- -list: lista el contenido de un repositorio.
- -delete: borra una entrada de un repositorio.
- -export: exporta un certificado.
- -import: importa un certificado.
- -help: muestra la ayuda (listando todos los comandos).
- **Ejemplo:** Creamos un fichero llamado repositorio con toda la información necesaria para llevar a cabo la firma digital: un par de claves pública y privada, y un certificado digital.



```

C:\trabajo>keytool -genkey -alias MiUsu -dname "cn=MiUsu, ou=Curso de Java, o=Aula Mentor, c=es" -keystore repositorio -storepass mipass -keypass mipass -validity 365

C:\trabajo>keytool -list -keystore repositorio -storepass mipass

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 2 entries

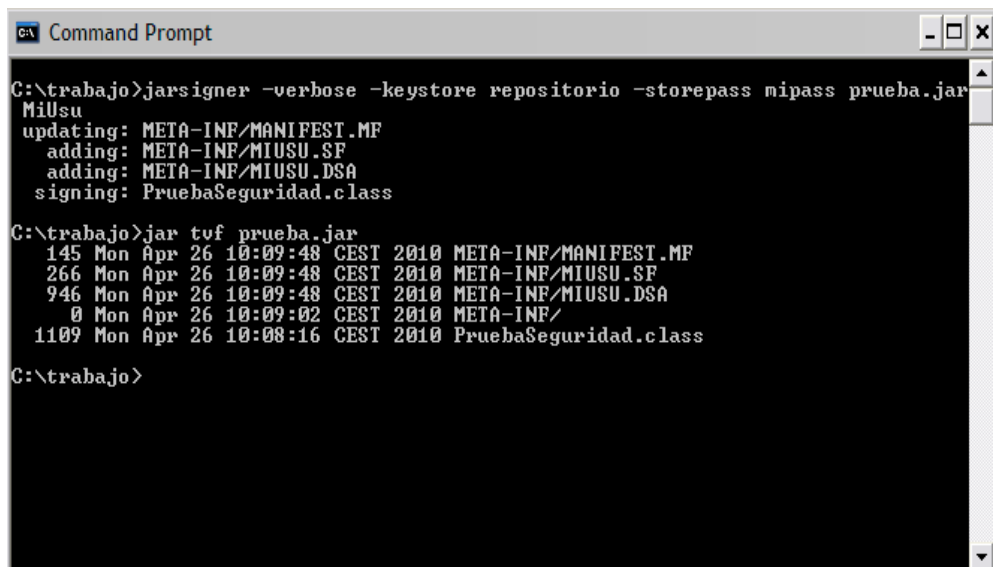
miusuario, Apr 26, 2010, PrivateKeyEntry,
Certificate fingerprint (MD5): 10:9F:69:87:BC:A5:F3:76:31:04:B6:12:5A:C7:9B:EF
miusu, Apr 26, 2010, PrivateKeyEntry,
Certificate fingerprint (MD5): B3:8E:24:F6:0F:C7:A6:B1:C6:A1:4B:BB:75:51:88:2F

C:\trabajo>
  
```

De igual forma que en la JDK tenemos una herramienta para la generación de las claves y el certificado, también tenemos una herramienta para utilizar dicha información y realizar firmas digitales: `jarsigner.exe`

Esta herramienta utiliza distintos parámetros. En nuestro caso usaremos:

- `-verbose`: para que muestre información de lo que va haciendo por pantalla.
- `-keystore`: con el repositorio donde está la clave privada.
- `-storepass`: con la contraseña de acceso al repositorio.
- `%fichero%`: fichero a firmar.
- `%alias%`: identificador de la clave a utilizar.
- **Ejemplo**: Firmamos digitalmente con la información generada anteriormente nuestro código.



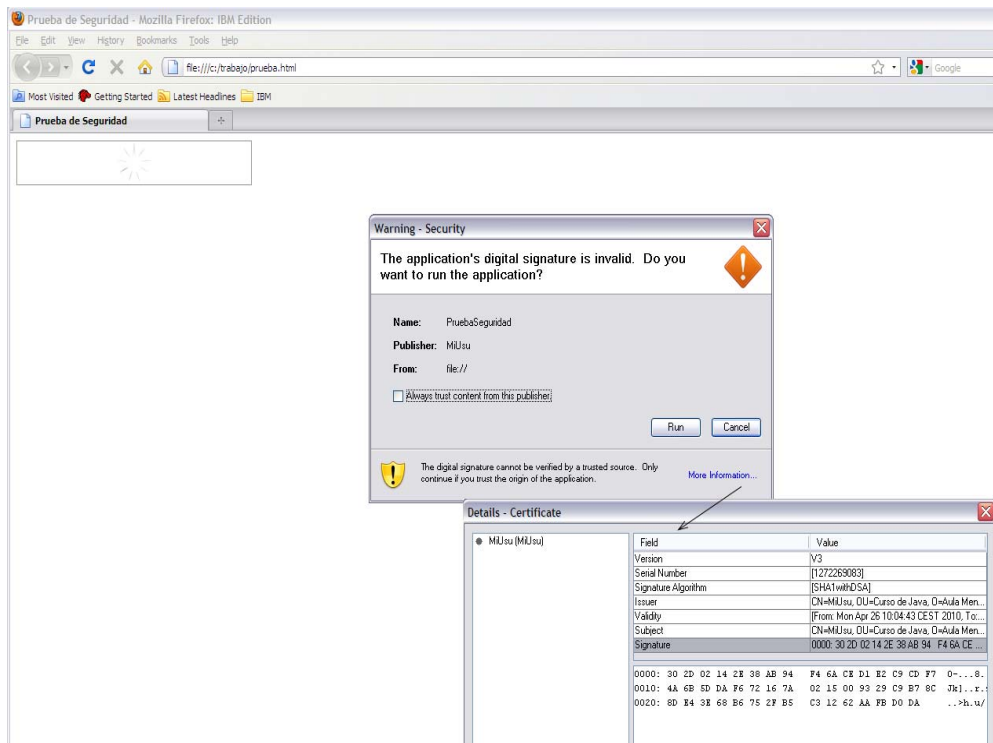
```

C:\trabajo>jarsigner -verbose -keystore repositorio -storepass mipass prueba.jar
MiUsu
updating: META-INF/MANIFEST.MF
adding: META-INF/MIUSU.SF
adding: META-INF/MIUSU.DSA
signing: PruebaSeguridad.class

C:\trabajo>jar tvf prueba.jar
145 Mon Apr 26 10:09:48 CEST 2010 META-INF/MANIFEST.MF
266 Mon Apr 26 10:09:48 CEST 2010 META-INF/MIUSU.SF
946 Mon Apr 26 10:09:48 CEST 2010 META-INF/MIUSU.DSA
0 Mon Apr 26 10:09:02 CEST 2010 META-INF/
1109 Mon Apr 26 10:08:16 CEST 2010 PruebaSeguridad.class

C:\trabajo>
  
```

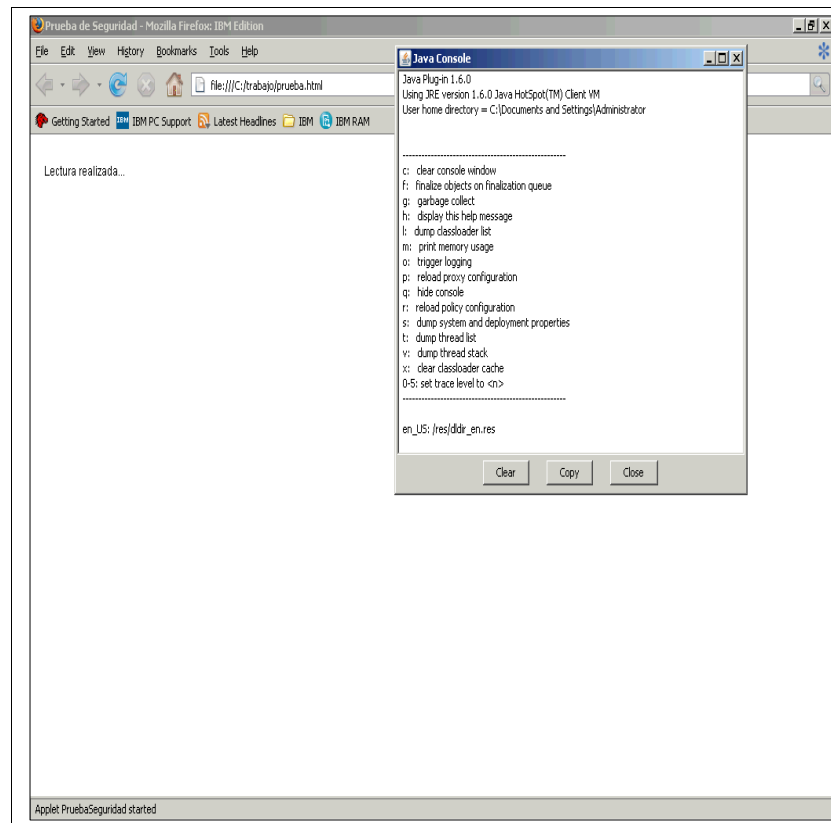
- **Ejemplo**: De nuevo, ejecutamos el Applet Java que intenta leer el archivo `hosts` del sistema para poder conocer otras máquinas de nuestra red. La diferencia es que esta vez está firmado digitalmente.



Esta vez, el navegador detecta que el Applet Java que intenta acceder a los recursos del sistema está firmado y nos muestra información sobre el firmante:

- El nombre del firmante.
- La Autoridad Certificadora que verifica la identidad del firmante.
- Si se confía en dicha Autoridad Certificadora o no.
- Si el certificado es válido y no ha caducado.

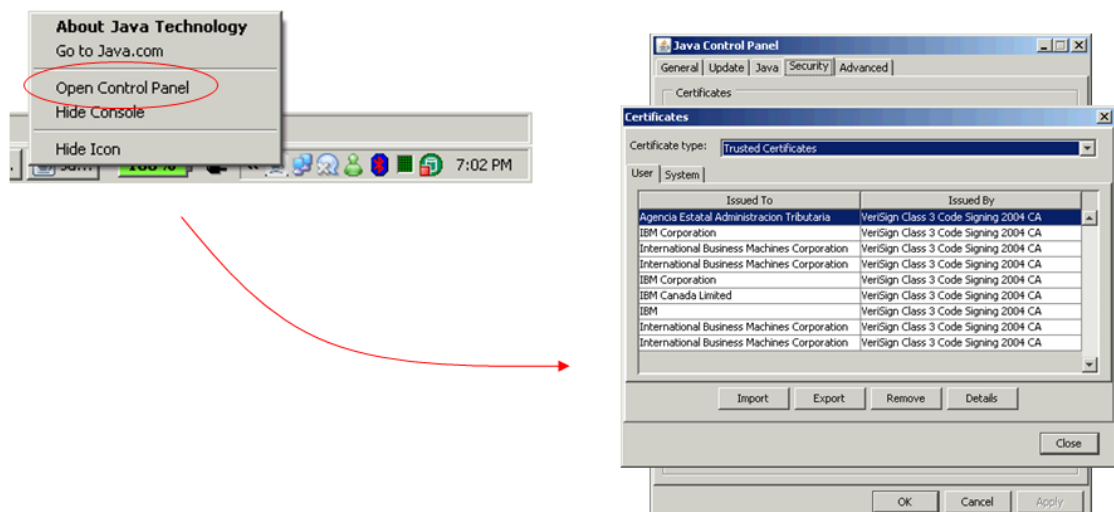
Y nos pregunta si confiamos en el firmante y por tanto le damos permiso de acceso. En caso afirmativo, el Applet Java se ejecutará satisfactoriamente accediendo sin problema al recurso local (el fichero hosts en nuestro ejemplo).



5.2 El certificado digital

Seguramente, el lector se estará preguntando, ¿podría evitar que el navegador nos pregunte y le de permiso de acceso por defecto?

La respuesta es si, pero para ello, tenemos que añadir el certificado del firmante a la lista de certificados en los que confiamos por defecto. El listado de estos certificados se puede ver a través del Panel de Control.



Un certificado digital contiene la siguiente información:

- La clave pública del firmante.
- El distinguished name del firmante: nombre, unidad organizativa, organización, ciudad, provincia y país.
- La firma digital del emisor del certificado (denominado Autoridad Certificadora).
- El distinguished name del emisor del certificado.

Una Autoridad Certificadora, es una entidad que certifica la identidad de un firmante. Algunos ejemplos de Autoridades Certificadoras son: Equifax, FNMT, VeriSign, Thawte, etc...

Normalmente, conseguir un certificado digital cuesta dinero. Es por ello, que normalmente para realizar pruebas creamos nuestros propios certificados, denominados, certificados auto-firmados. Es decir, el propio firmante certifica que es quien dice ser.

En España, la Fabrica Nacional de Moneda y Timbre (FNMT) emite gratuitamente certificados a todos los españoles con DNI:

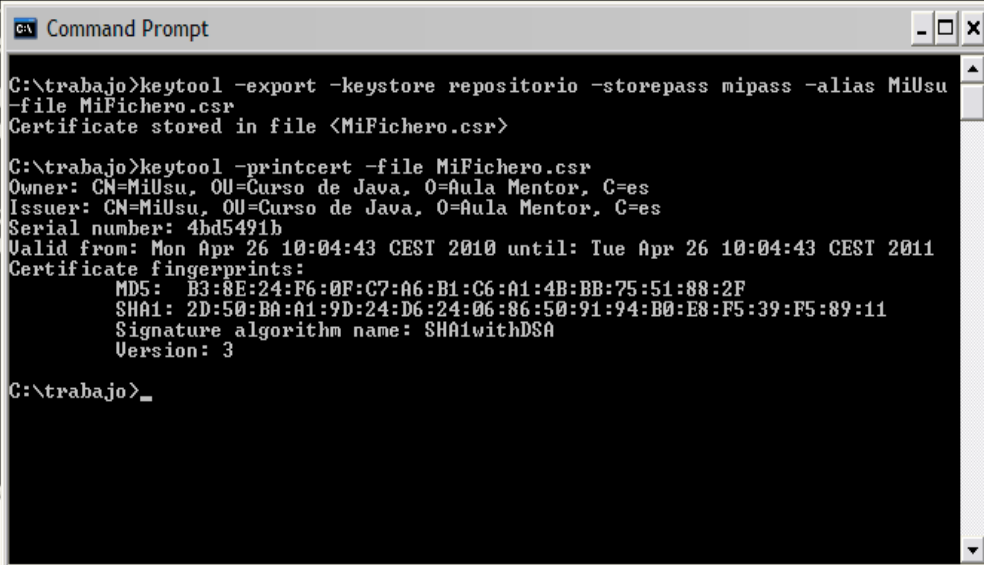
http://www.cert.fnmt.es/index.php?cha=cit&sec=obtain_cert

Nota: La firma digital tiene validez legal, por lo que su uso debe ser cuidadoso y responsable.

Para exportar un certificado digital, debemos utilizar de nuevo la herramienta keytool.exe incluida en la JDK como vimos anteriormente. Pero en esta ocasión, utilizaremos los siguientes parámetros:

- -export: comando para exportar el certificado.
- -keystore: con el nombre del repositorio donde está el certificado.

- -storepass: con la contraseña de acceso al repositorio.
- -alias: con el identificador de las claves/certificado.
- -file: con el nombre del fichero donde guardar el certificado exportado.
- **Ejemplo:** Exportamos nuestro certificado digital creado anteriormente. Con el comando -printcert podemos ver los detalles del certificado.



```

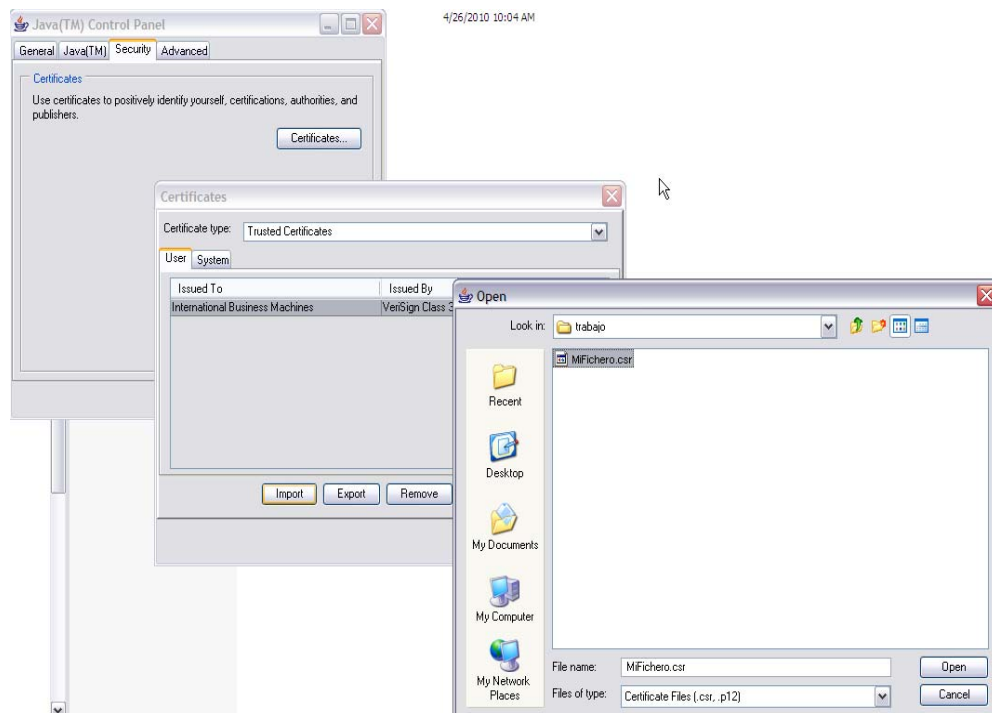
C:\trabajo>keytool -export -keystore repositorio -storepass mipass -alias MiUsu
-file MiFichero.csr
Certificate stored in file <MiFichero.csr>

C:\trabajo>keytool -printcert -file MiFichero.csr
Owner: CN=MiUsu, OU=Curso de Java, O=Aula Mentor, C=es
Issuer: CN=MiUsu, OU=Curso de Java, O=Aula Mentor, C=es
Serial number: 4bd5491b
Valid from: Mon Apr 26 10:04:43 CEST 2010 until: Tue Apr 26 10:04:43 CEST 2011
Certificate fingerprints:
    MD5: B3:8E:24:F6:0F:C7:A6:B1:C6:A1:4B:BB:75:51:88:2F
    SHA1: 2D:50:BA:A1:9D:24:D6:24:06:86:50:91:94:B0:E8:F5:39:F5:89:11
Signature algorithm name: SHA1withDSA
Version: 3

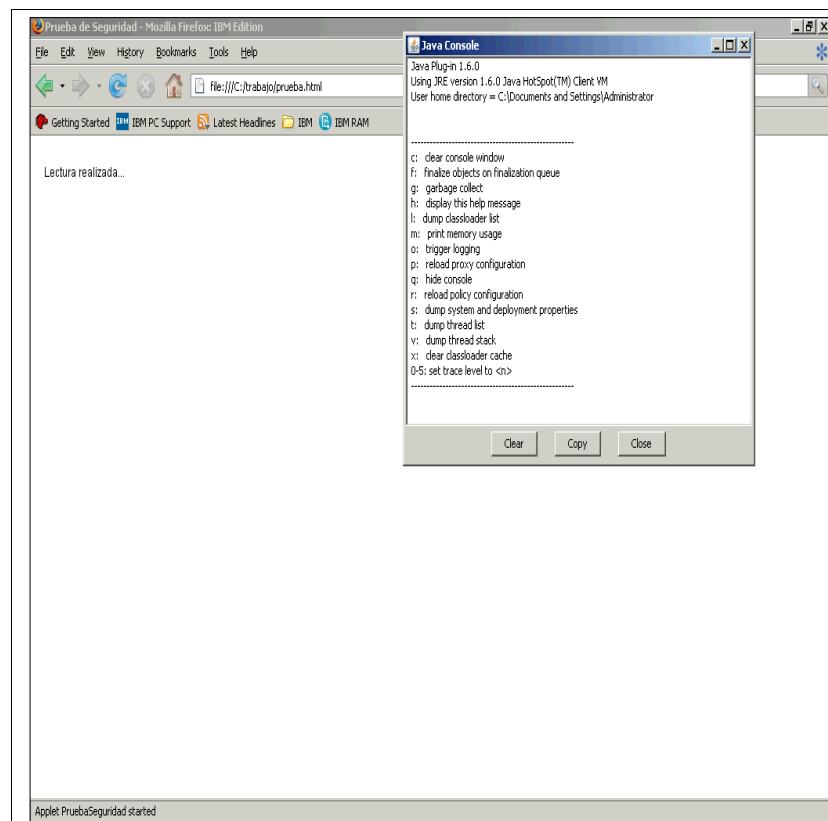
C:\trabajo>_
  
```

Por último, utilizando la consola del Java Plug-in importamos el certificado.

- **Ejemplo:** Importación de un certificado digital a la lista de certificados en los que confiamos.

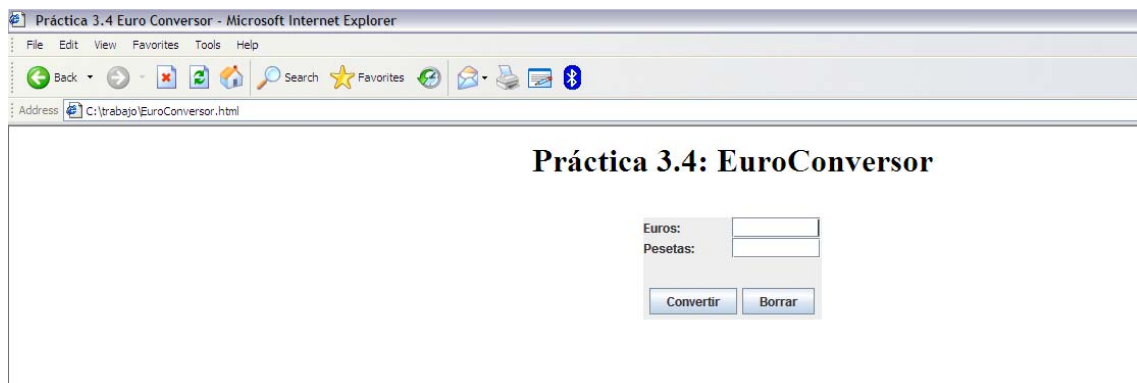


- **Ejemplo:** De nuevo, ejecutamos el Applet Java que intenta leer el archivo hosts del sistema para poder conocer otras máquinas de nuestra red. La diferencia es que esta vez está firmado digitalmente con un certificado digital en el que confiamos con lo que ejecutará el Applet Java sin preguntarnos nada.






PRÁCTICA A: Desarrollar, tanto con la JDK de Oracle como con Eclipse, un applet que convierta de euros a pesetas. El resultado debe ser algo parecido a:



Nota: todos los textos del applet deberán ser parámetros que se pasen por medio de la página HTML

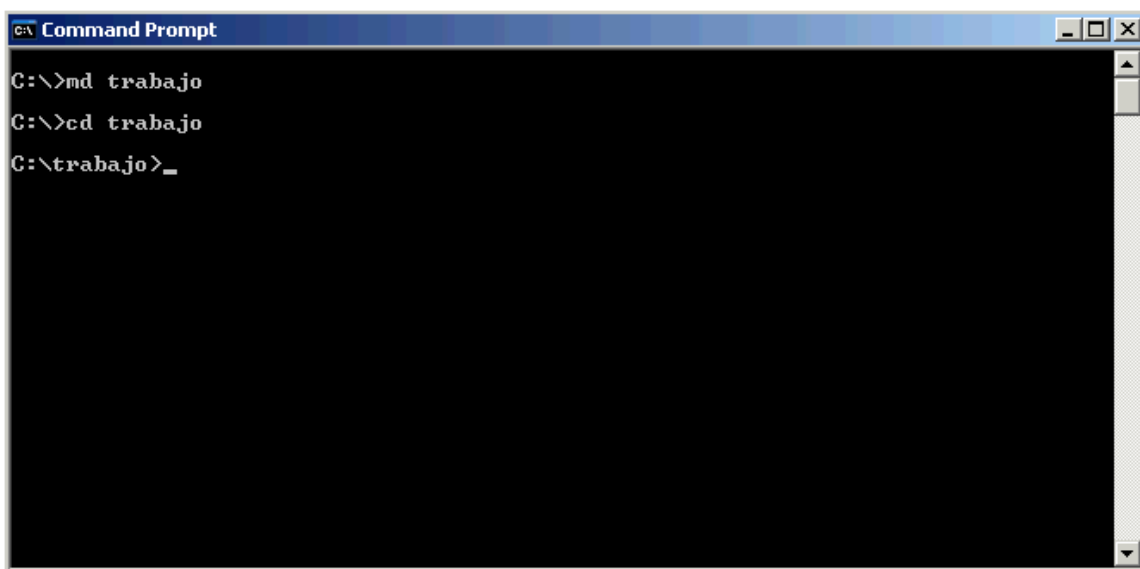
Solución con la JDK de Oracle:

En el caso de que no esté puesta la variable de entorno PATH, abrir una sesión DOS y ajustar la variable de entorno PATH para que el Sistema Operativo sepa encontrar las herramientas del JDK. Para respetar el valor que ya tuviese la variable PATH le añadimos %PATH%.



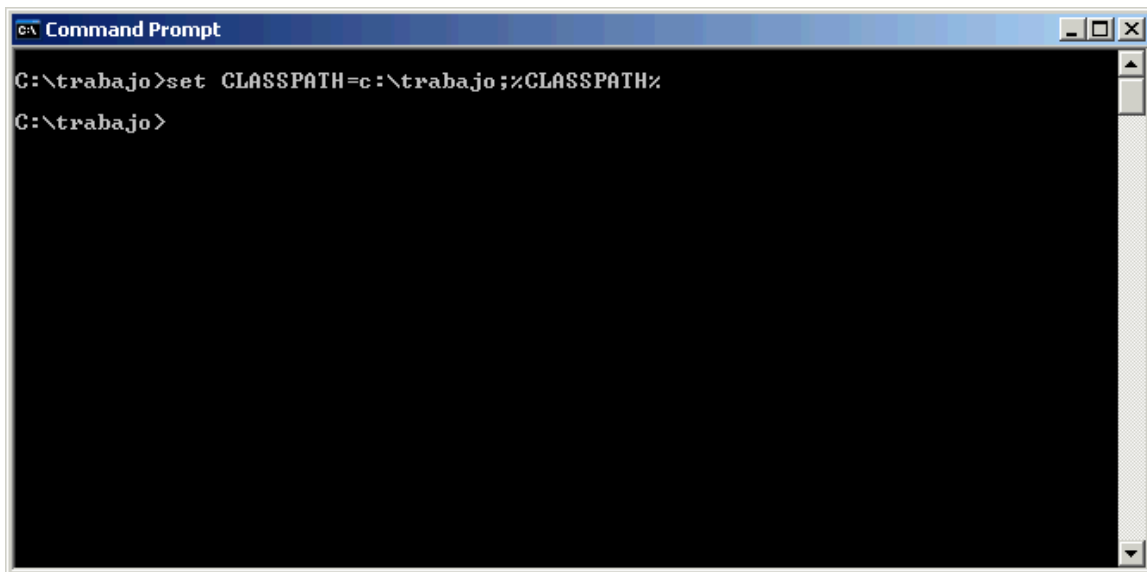
```
C:\>set PATH=c:\jdk1.6.0\bin;%PATH%
C:\>_
```

Creamos un directorio de trabajo donde guardar el programa Java.



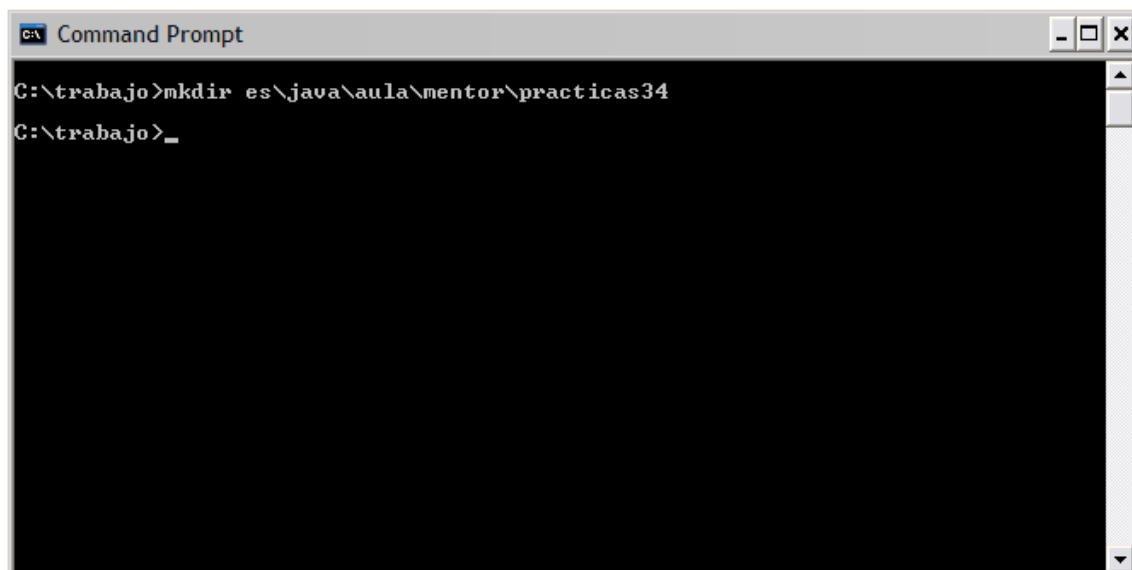
```
C:\>md trabajo
C:\>cd trabajo
C:\trabajo>_
```

Ajustar la variable de entorno CLASSPATH para que las herramientas del JDK sepan encontrar nuestras clases Java. Tenemos dos opciones, o añadir el . (punto) y siempre ejecutar las herramientas en el directorio donde se encuentre el código, o añadir el directorio de trabajo y ejecutar las herramientas donde queramos. Para respetar el valor que ya tuviese la variable CLASSPATH le añadimos %CLASSPATH%



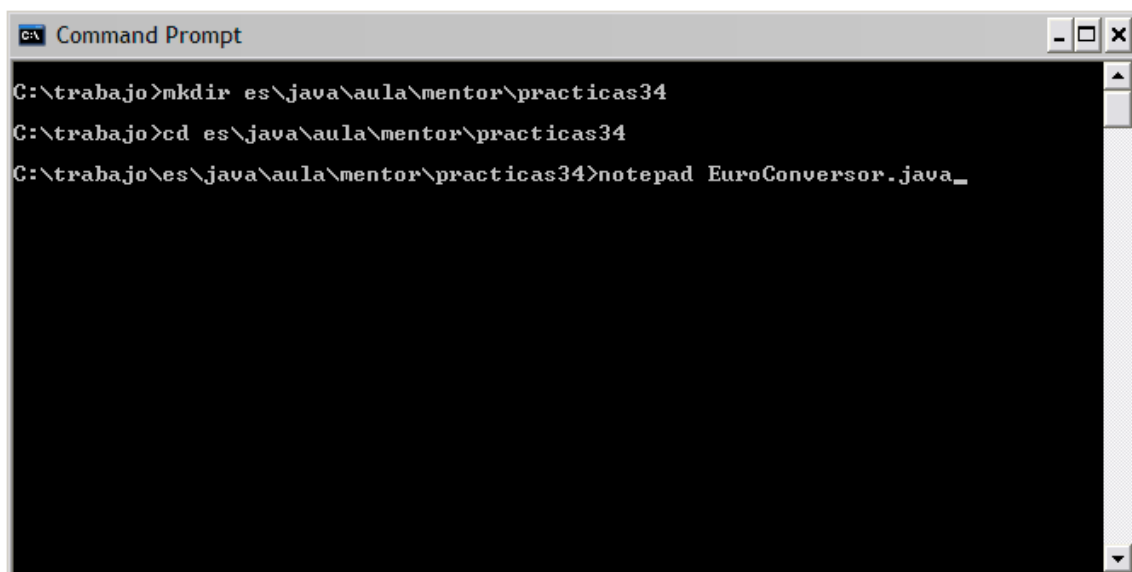
```
C:\> Command Prompt
C:\trabajo>set CLASSPATH=c:\trabajo;%CLASSPATH%
C:\trabajo>
```

Creamos el directorio donde va a estar nuestra clase
es\java\aula\mentor\practicas34



```
C:\> Command Prompt
C:\trabajo>mkdir es\java\aula\mentor\practicas34
C:\trabajo>_
```

Desde el directorio es\java\aula\mentor\practicas34, con un editor de texto (por ejemplo Notepad) vamos a escribir el código fuente de nuestra clase java; el nombre del fichero EuroConversor.java debe ser exactamente igual (incluyendo mayúsculas y minúsculas) al de la clase Java que vamos a desarrollar.



```
C:\trabajo>mkdir es\java\aula\mentor\practicass34
C:\trabajo>cd es\java\aula\mentor\practicass34
C:\trabajo\es\java\aula\mentor\practicass34>notepad EuroConversor.java_
```

Y aceptamos la creación de un fichero nuevo.

El código de Applet quedaría como sigue:

```

EuroConversor.java - Notepad
File Edit Format View Help

package es.java.aula.mentor.practicas34;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class EuroConversor extends JApplet implements ActionListener
{
    private JTextField euros = null;
    private JTextField pesetas = null;

    public void init()
    {
        this.getContentPane().setLayout(new BorderLayout());

        euros = new JTextField();
        euros.setHorizontalAlignment(SwingConstants.RIGHT);
        pesetas = new JTextField();
        pesetas.setHorizontalAlignment(SwingConstants.RIGHT);

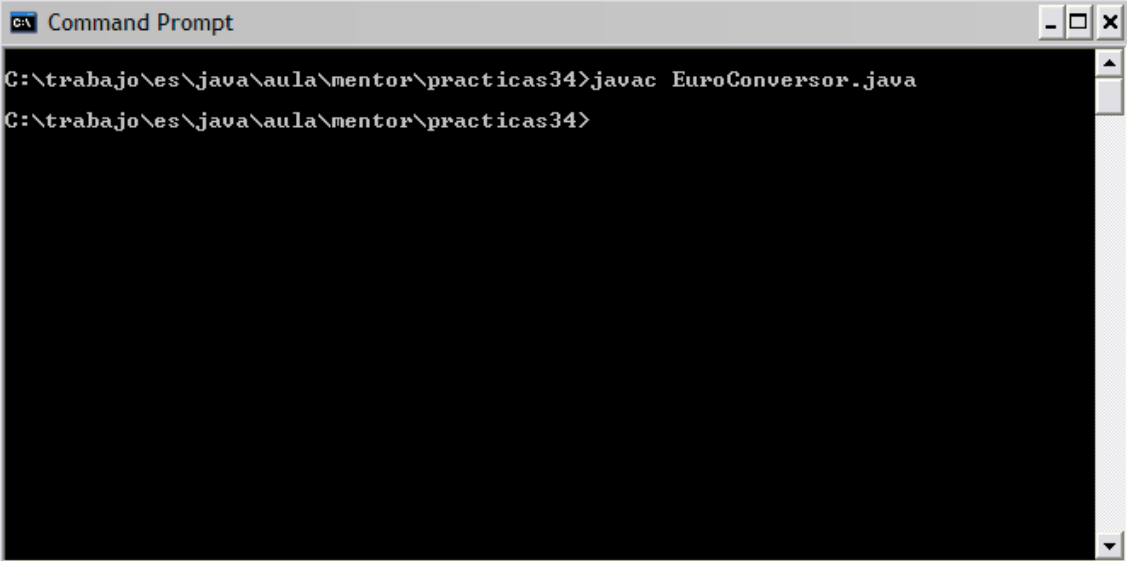
        JPanel aux = new JPanel();
        aux.setLayout(new GridLayout(2,2));
        aux.add(new JLabel(this.getParameter("TEXT01")));
        aux.add(euros);
        aux.add(new JLabel(this.getParameter("TEXT02")));
        aux.add(pesetas);
        this.getContentPane().add(aux, BorderLayout.NORTH);

        aux = new JPanel();
        JButton convertir = new JButton(this.getParameter("TEXT03"));
        convertir.setName("convertir");
        convertir.addActionListener(this);
        aux.add(convertir);
        JButton limpiar = new JButton(this.getParameter("TEXT04"));
        aux.add(limpiar);
        limpiar.setName("limpiar");
        limpiar.addActionListener(this);
        this.getContentPane().add(aux, BorderLayout.SOUTH);
    }

    // Se ha producido un Action Event
    public void actionPerformed(ActionEvent ev)
    {
        String aux = ((JComponent)ev.getSource()).getName();
        if(aux.equals("convertir")) {
            float tmp = Float.parseFloat(euros.getText());
            tmp = tmp * 166.386F;
            pesetas.setText(Float.toString(tmp));
        }
        else {
            euros.setText("");
            pesetas.setText("");
        }
    }
}

```

Compilamos la clase:



```

C:\trabajo\es\java\aula\mentor\practicass34>javac EuroConversor.java
C:\trabajo\es\java\aula\mentor\practicass34>

```

Ahora, creamos la página HTML que va a contener al Applet. Con el notepad, nos creamos el fichero EuroConversor.html en el directorio c:\trabajo

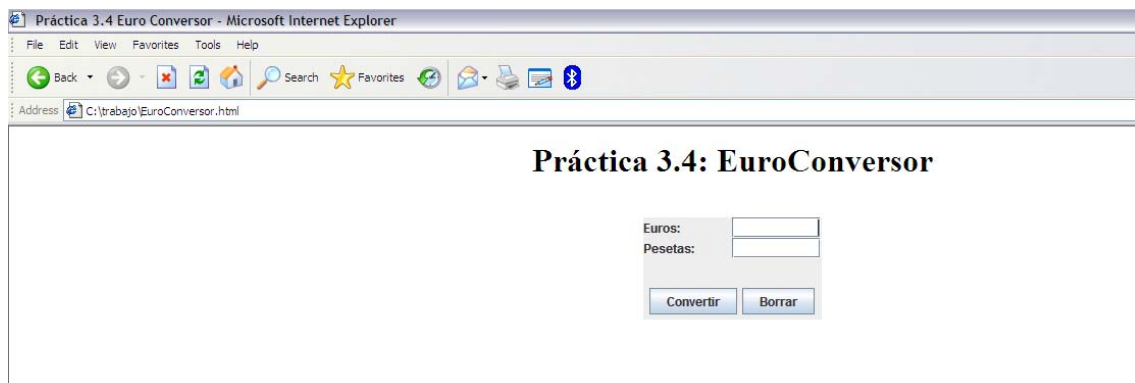


```

<HTML>
<HEAD>
<TITLE>
Práctica 16d
</TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>Práctica 3.4: EuroConversor</H1>
<BR/>
<APPLET code="es.java.aula.mentor.practicass34.EuroConversor"
height="100" width="175"
<PARAM name="TEXT01" value="Euros:"/>
<PARAM name="TEXT02" value="Pesetas:"/>
<PARAM name="TEXT03" value="Convertir"/>
<PARAM name="TEXT04" value="Borrar"/>
</APPLET>
</CENTER>
</BODY>
</HTML>

```

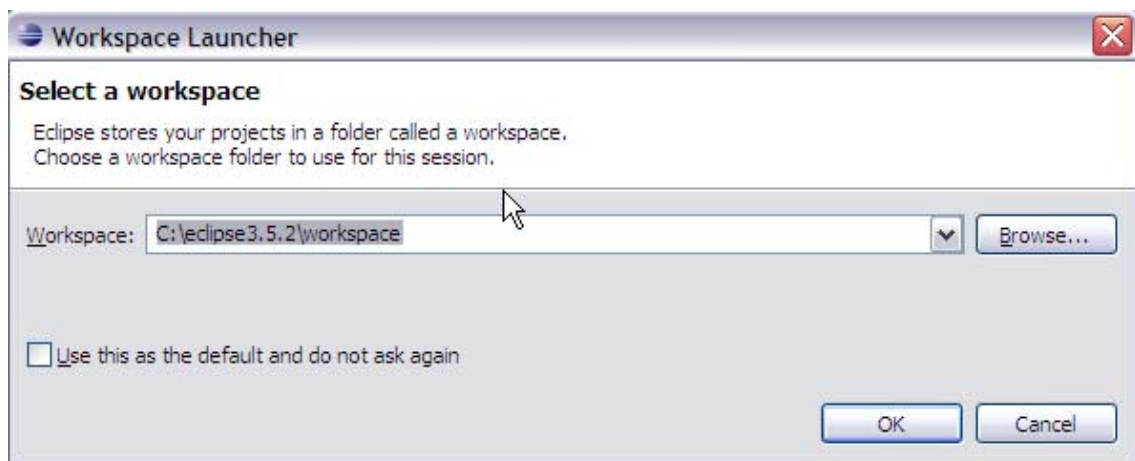
Ejecutamos el applet en el navegador, indicando fi-
le:///c:/trabajo/EuroConversor.html



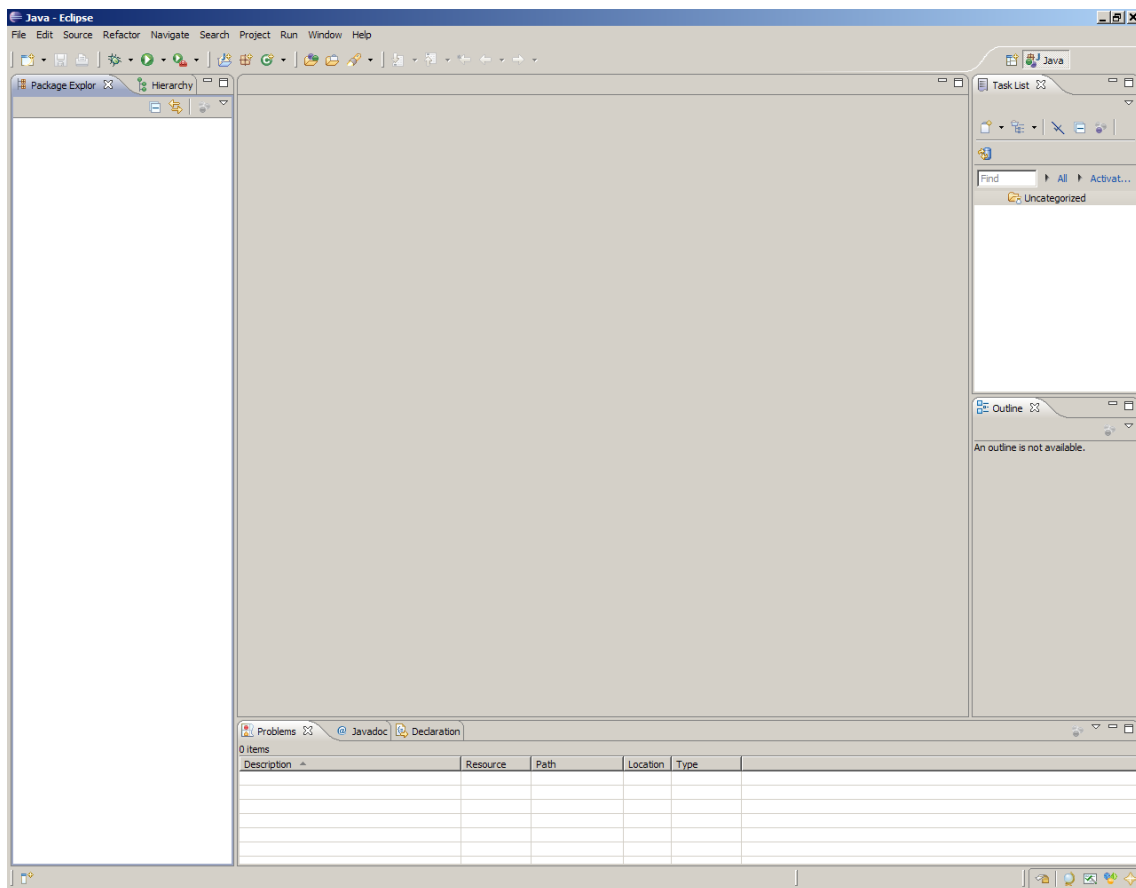
Solución con Eclipse:

Arrancar Eclipse, ejecutando `c:\eclipse3.7.1\eclipse.exe`

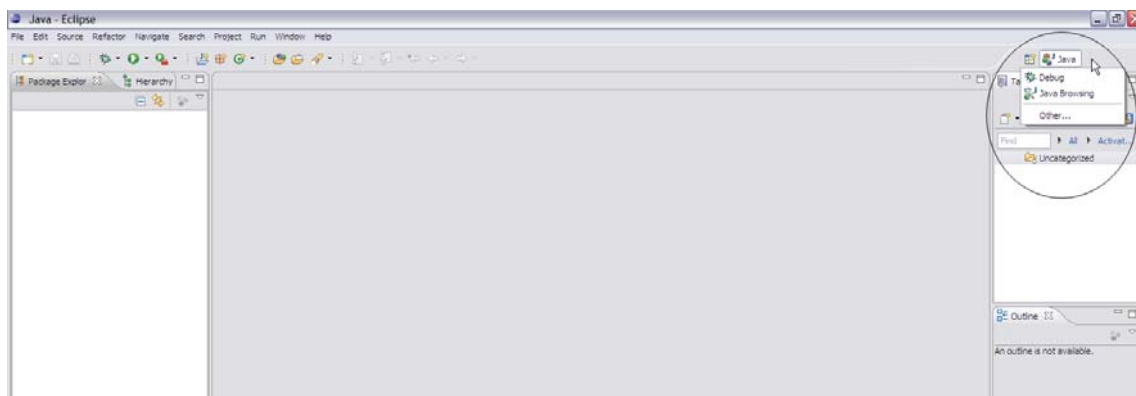
Seleccionar la ubicación del "workspace" (o área de trabajo).



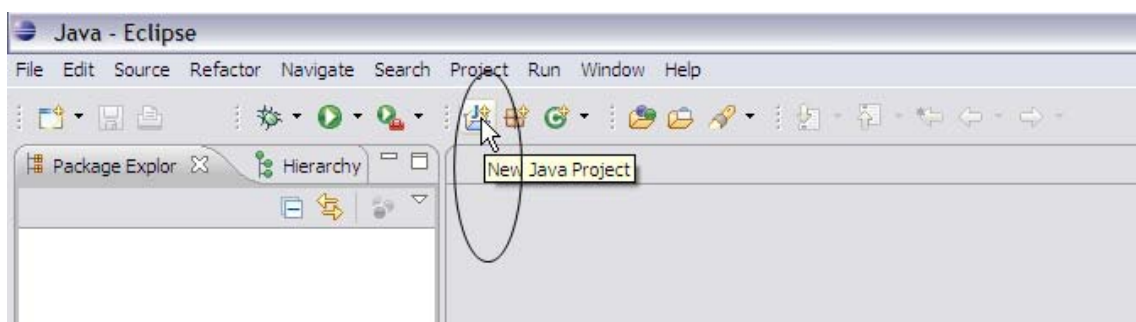
Aparecerá la pantalla para empezar a trabajar.



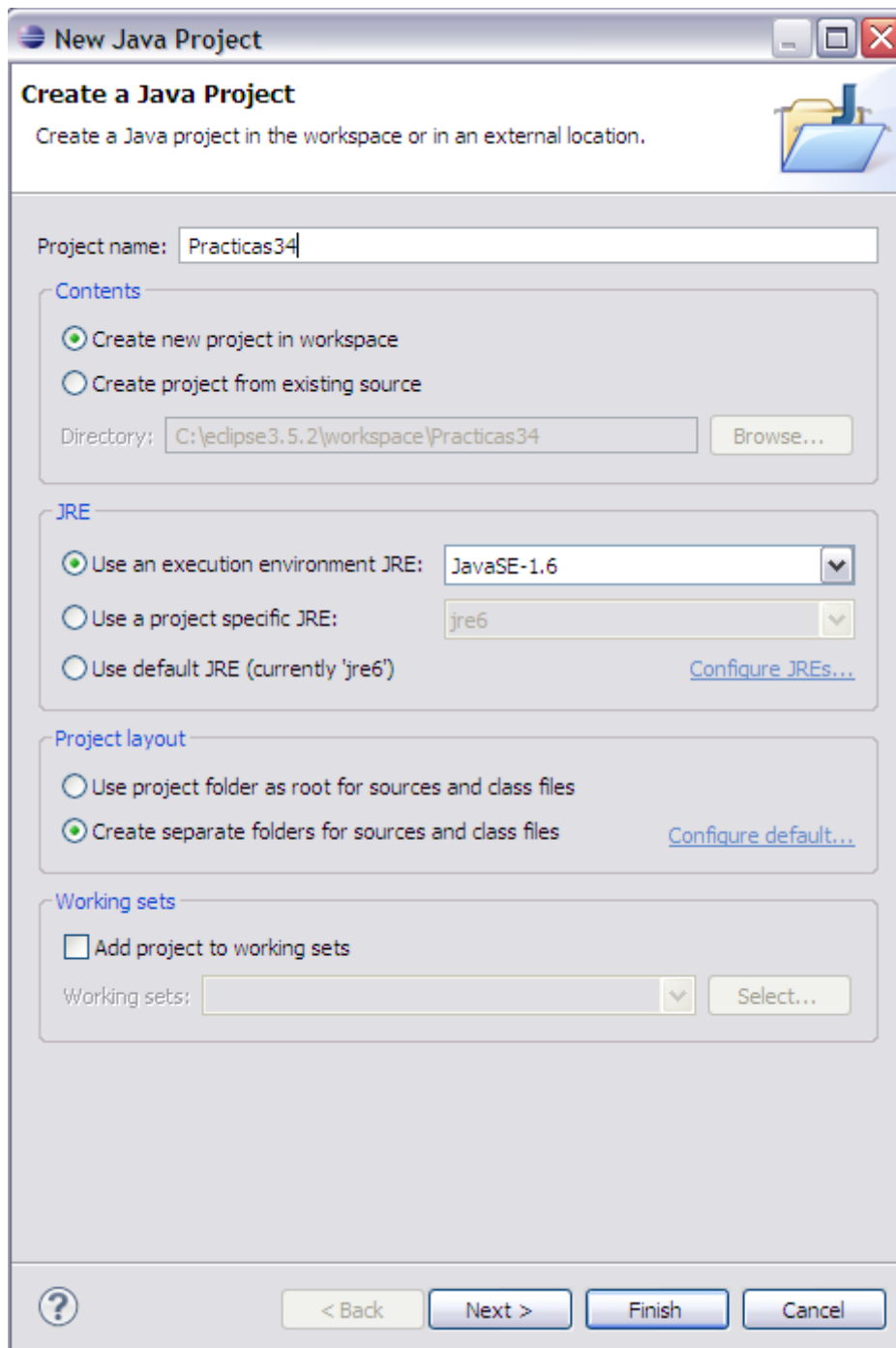
Verificar que la perspectiva Java está abierta, y sino cambiar a ella



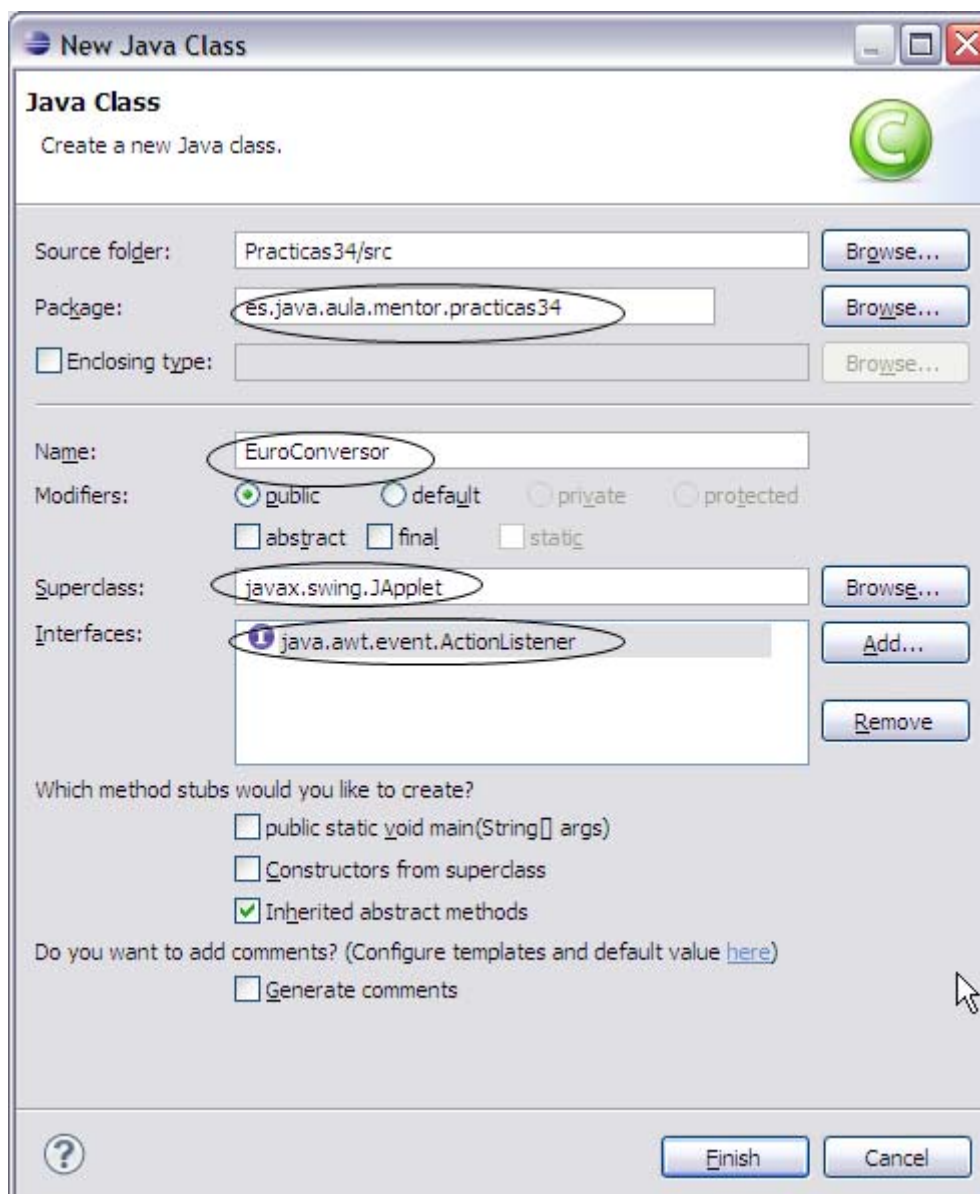
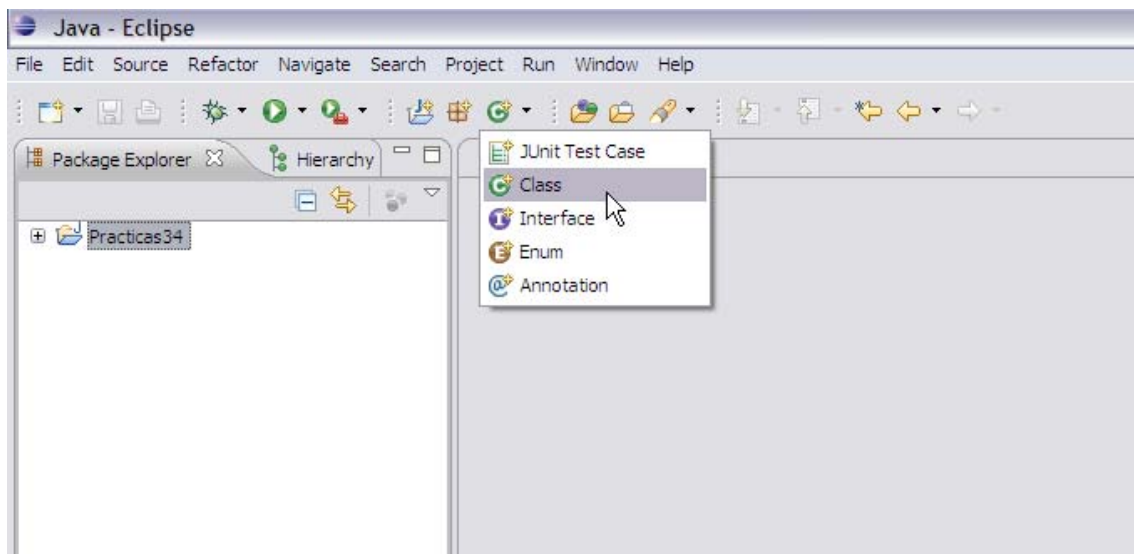
Crear un proyecto nuevo de nombre Practicas34



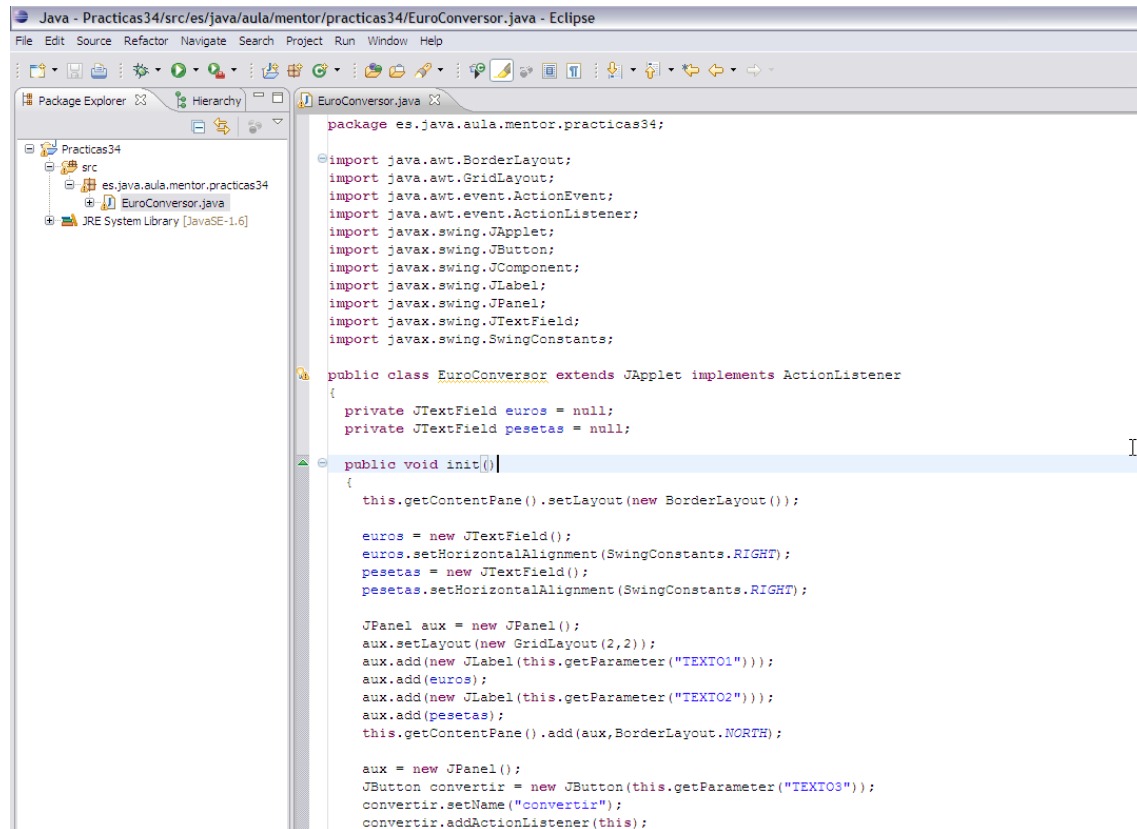
Darle el nombre y seleccionar Finish.



Crear una clase Java nueva llamado EuroConverdor, especificando el paquete es.java.aula.mentor.practicas34 e indicando que hereda de la clase javax.swing.JApplet y que implementa el interface java.awt.event.ActionListener



El código dentro del método Applet, quedaría como sigue:



```

package es.java.aula.mentor.practicas34;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class EuroConversor extends JApplet implements ActionListener
{
    private JTextField euros = null;
    private JTextField pesetas = null;

    public void init()
    {
        this.getContentPane().setLayout(new BorderLayout());

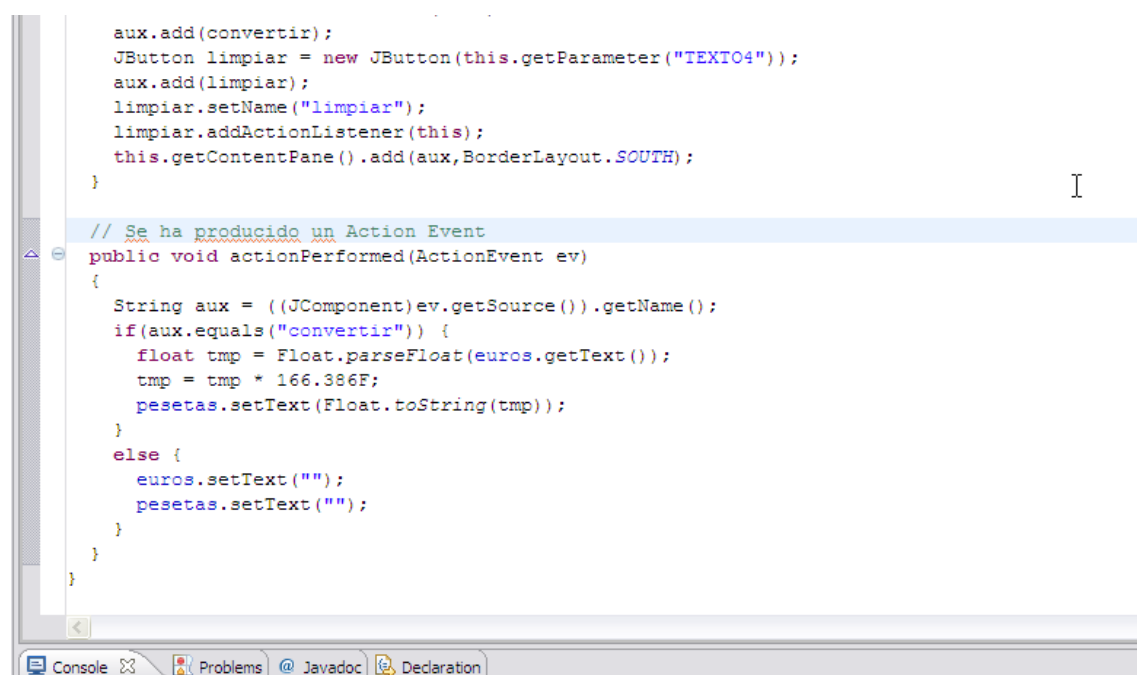
        euros = new JTextField();
        euros.setHorizontalAlignment(SwingConstants.RIGHT);
        pesetas = new JTextField();
        pesetas.setHorizontalAlignment(SwingConstants.RIGHT);

        JPanel aux = new JPanel();
        aux.setLayout(new GridLayout(2,2));
        aux.add(new JLabel(this.getParameter("TEXT01")));
        aux.add(euros);
        aux.add(new JLabel(this.getParameter("TEXT02")));
        aux.add(pesetas);
        this.getContentPane().add(aux, BorderLayout.NORTH);

        aux = new JPanel();
        JButton convertir = new JButton(this.getParameter("TEXT03"));
        convertir.setName("convertir");
        convertir.addActionListener(this);
    }
}

```

Continuación del código:



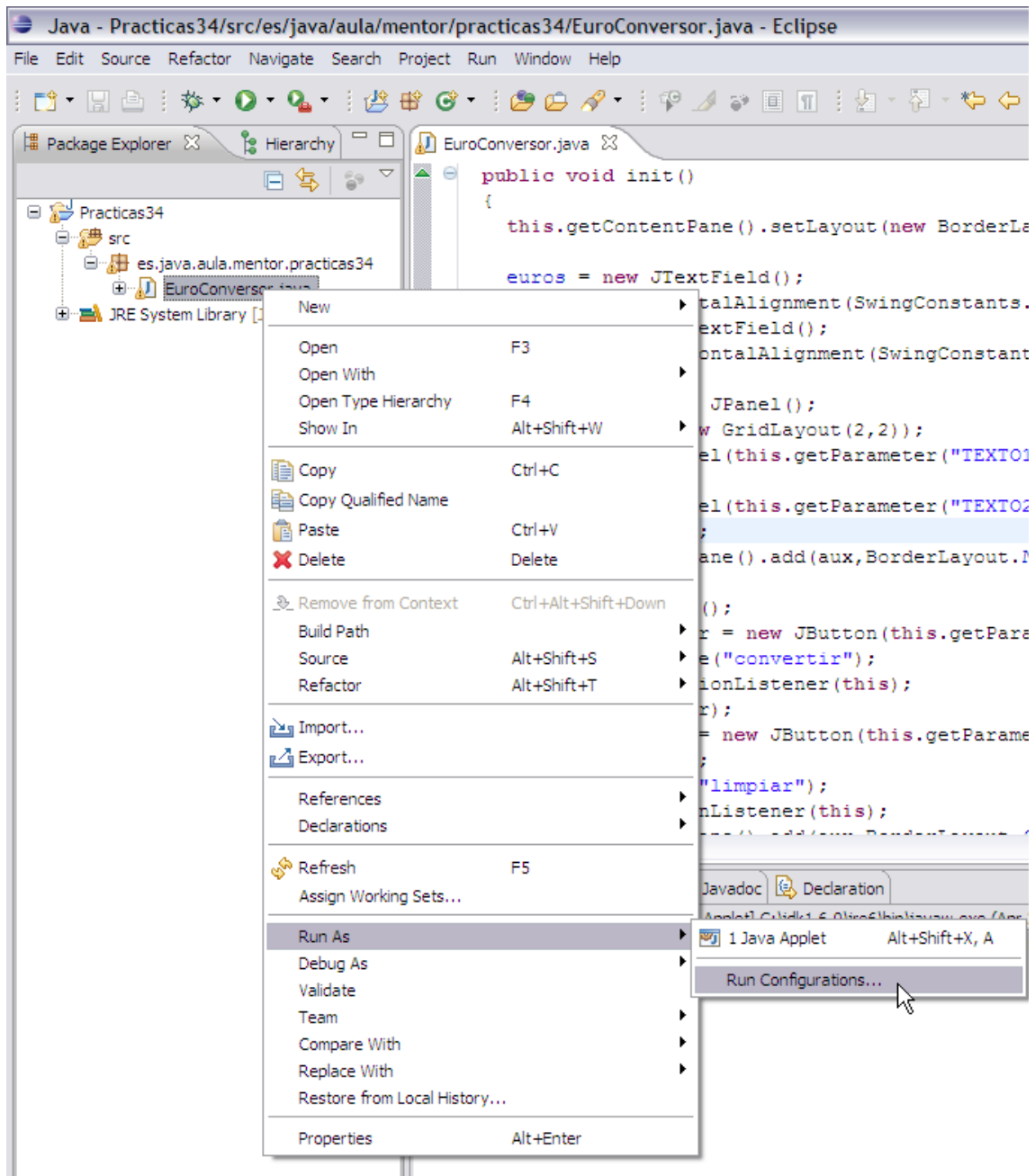
```

        aux.add(convertir);
        JButton limpiar = new JButton(this.getParameter("TEXT04"));
        aux.add(limpiar);
        limpiar.setName("limpiar");
        limpiar.addActionListener(this);
        this.getContentPane().add(aux, BorderLayout.SOUTH);
    }

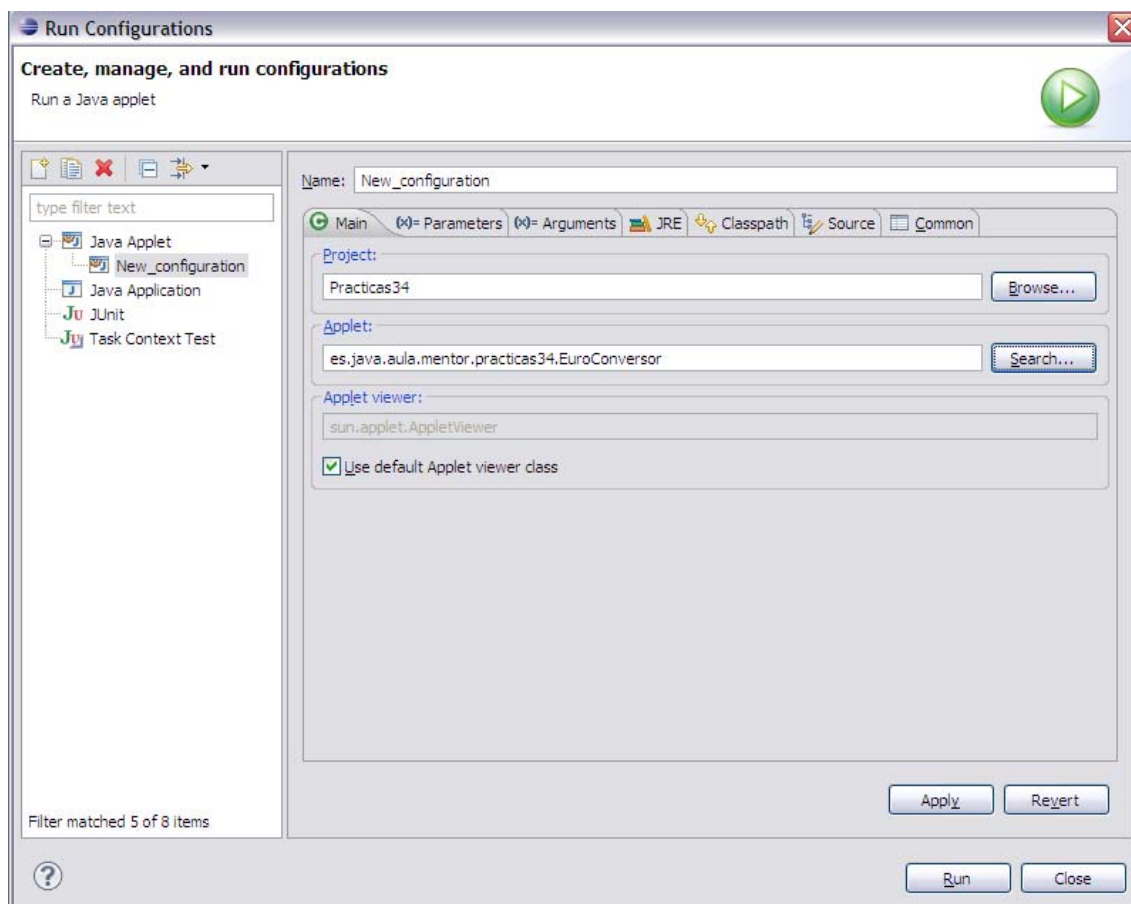
    // Se ha producido un Action Event
    public void actionPerformed(ActionEvent ev)
    {
        String aux = ((JComponent)ev.getSource()).getName();
        if(aux.equals("convertir")) {
            float tmp = Float.parseFloat(euros.getText());
            tmp = tmp * 166.386F;
            pesetas.setText(Float.toString(tmp));
        }
        else {
            euros.setText("");
            pesetas.setText("");
        }
    }
}

```

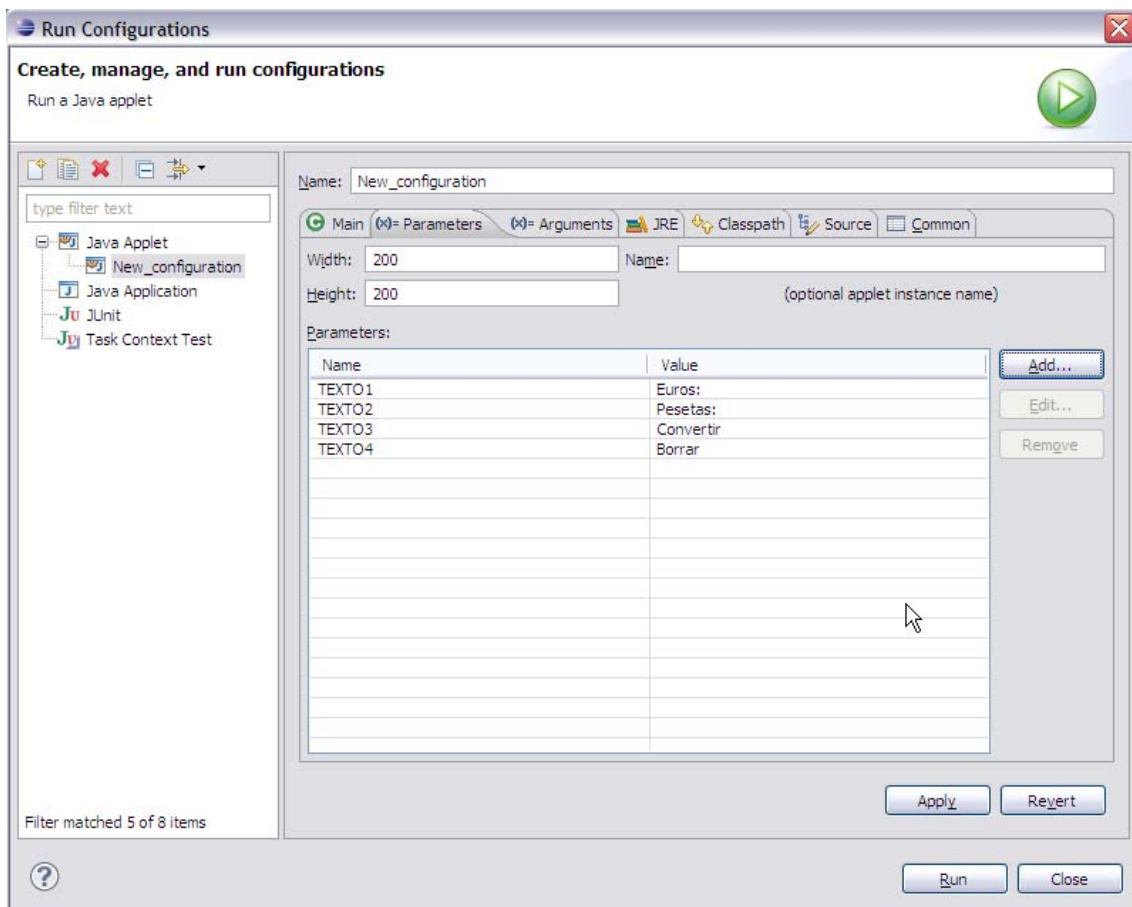
Ya podemos probar el Applet vía AppletViewer. Para ello, con el botón derecho sobre el Applet ejecutar Run As -> Run Configurations



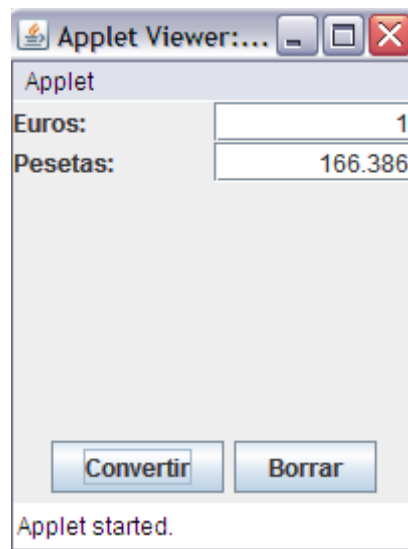
Mediante un doble click de ratón sobre Java Applet nos generará una nueva configuración y mediante el botón Search buscamos la clase del Applet que queremos ejecutar



En la pestaña Parameters, introducimos los parámetros que queremos pasarle al Applet



Con el botón Run, podemos ejecutar el AppletViewer





para recordar

En esta Unidad hemos estudiado los **Applet Java**, que son un tipo de aplicación Java que se inserta dentro de las páginas HTML. Cuando estas páginas se descargan, los Applets se ejecutan en el navegador. Hemos visto que hay dos clases que realizan la función de Applet Java que son: `java.awt.Applet` y `javax.swing.JApplet`.

El ciclo de vida de un Applet Java queda definido por los siguientes métodos:

- **init()** el cual se ejecuta una primera vez, cuando se llama al constructor del Applet.
- **start()** el cual se ejecuta tras el método `init` y cada vez que se necesite arrancar el Applet, como por ejemplo en la recarga de la página html que lo contiene.
- **stop()** el cual se ejecuta cada vez que hay que parar el Applet.
- **destroy()** el cual se ejecuta una sola vez cuando se cierra el navegador o este decide eliminarlo.
- **paint(Graphics g)** el cual se ejecuta para pintar o reescribir el aspecto visual del Applet.

Como método de ejecución de los Applet Java se han estudiado tanto su inclusión en una página **html** para ser ejecutados dentro de un navegador y mediante la herramienta **AppletViewer**

Por último se han estudiado temas específicos de Seguridad relacionados con los Applet Java, para poder ejecutar código Java que pueda interactuar con la máquina física en la que se está ejecutando.