Java Avanzado

El paquete AWT

Copyright

AWT

- AWT: Abstract Window Toolkit.
- Es una librería de clases Java para el desarrollo de interfaces de usuario gráficas (GUI).
- Por tratarse de código Java, los aplicativos serán independientes de plataforma. No así su apariencia visual.
- Es la librería básica. Sobre ella se construyó a posteriori otra mas flexible y potente: JFC/Swing
- La AWT se encuentra en el paquete: java.awt.*

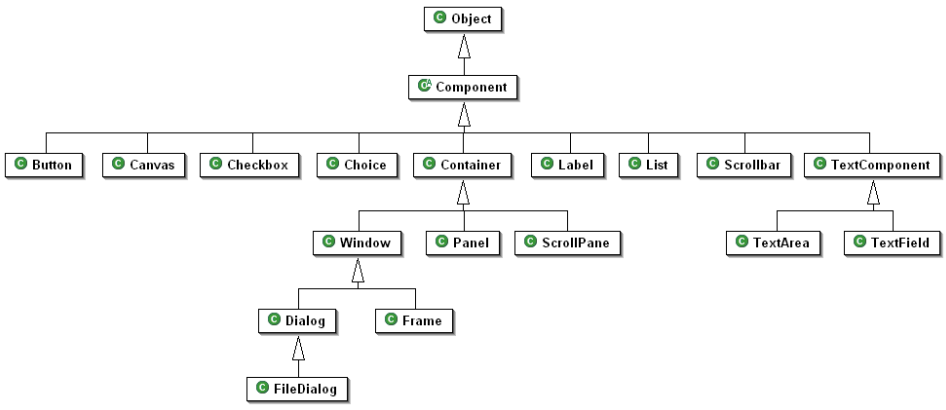
AWT

- Dispone de la mayoría de controles visuales estándar:
  - Button (push, radio y check).
  - Canvas.
  - Frame, Dialog.
  - Label.
  - List, Choice.
  - ScrollBar, ScrollPane.
  - TextField, TextArea.
  - Menu.

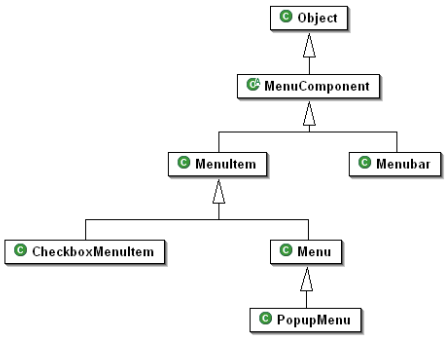
Elementos

- 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.

Jerarquía de clases



Jerarquía de clases



java.awt.Component

- Se trata de una clase abstracta que implementa toda la funcionalidad básica de las clases visuales:
- Métodos para:
  - Mostrar y esconder.
  - Rendering.
  - Habilitar y deshabilitar, etc....
- Atributos como:
  - Color del foreground y background.
  - Tamaño y posición.

java.awt.Container

- Se trata de una clase que implementa la funcionalidad de contener a otros componentes.
- Los contenedores son a su vez componentes.
- Algunos contenedores:
  - Window.
  - Dialog y FileDialog.
  - Frame.
  - Panel: contenedor invisible.

java.awt.LayoutManager

- 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.

Coordenadas y posicionamiento

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

Coordenadas y posicionamiento

- La clase java.awt.Container posee varios métodos para acceder a los componentes que contienen como por ejemplo:

add(Component c); o add(Component c, Object o);
Inserta el componente c en el contenedor referenciado.

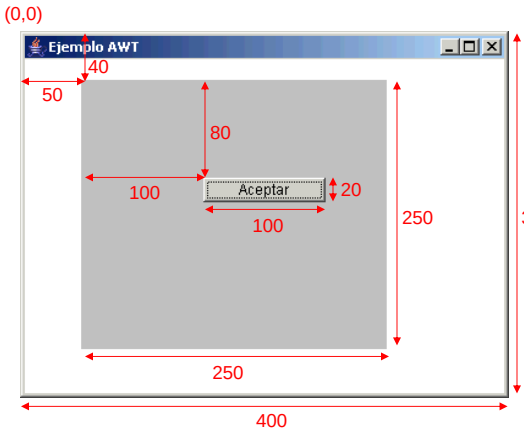
remove(Component c);
Elimina el componente c del contenedor referenciado.

Component[] getComponents();
Devuelve un array con los componentes del contenedor referenciado.

Pasos a seguir

- 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”);

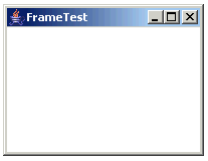
Ejemplo




```
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);
   }
}
```

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);
  }
}
```

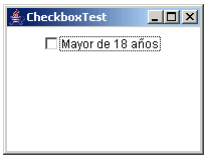


java.awt.Button

```
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);
  }
}
```

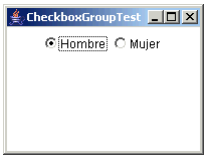
java.awt.Checkbox

```
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);
  }
}
```



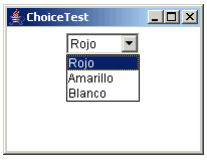
java.awt.CheckboxGroup

```
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);
  }
}
```



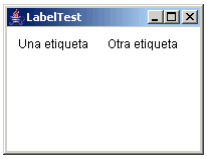
java.awt.Choice

```
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);
  }
}
```



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);
  }
}
```

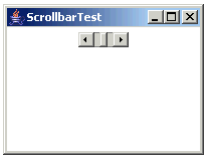


java.awt.List

```
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);
  }
}
```

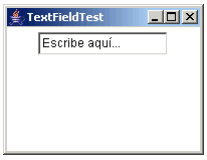
java.awt.Scrollbar

```
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);
  }
}
```



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);
  }
}
```

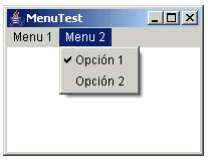


java.awt.TextArea

```
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);
  }
}
```

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);
  }
}
```

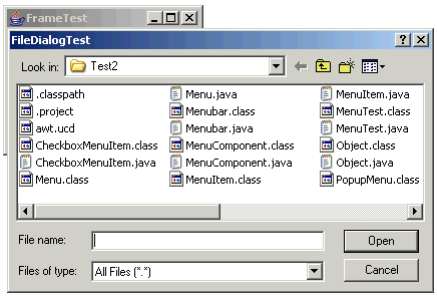


java.awt.Dialog

```
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);
  }
}
```

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.
  }
}
```



Layout Managers

- Todos los contenedores AWT tienen asociado un LayoutManager para coordinar el tamaño y la situación de sus componentes.
  - Panel -> FlowLayout
  - Frame -> 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 usar Layout Managers?

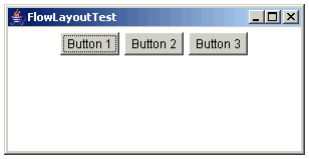
- Determinan el tamaño y la posición de los componentes en un contenedor.
- Tiene un API que permite al contenedor y al LayoutManager gestionar el cambio de tamaño del contenedor de manera transparente.
- Consiguen que la aplicación sea independiente de la resolución de las máquinas donde se ejecuta.

Layout Managers

- Implementan el interface java.awt.LayoutManager.
  - FlowLayout: sitúa los componentes de izquierda a derecha. Les modifica la posición pero no les modifica el tamaño.
  - BorderLayout: se basa en los puntos cardinales. Modifica tanto la posición como el tamaño de los componentes.
  - CardLayout: permite al desarrollador intercambiar distintas vistas como si se tratase de una baraja. Modifica tanto la posición como el tamaño de los componentes.
  - GridLayout: usa una matriz en la que sitúa cada uno de los componentes. El tamaño de todas las celdas es igual.
  - GridBagLayout: similar al anterior, pero no fuerza a que todos los componentes tengan el mismo tamaño.

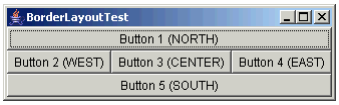
java.awt.FlowLayout

```
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);
  }
}
```



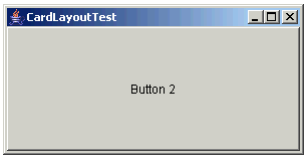
java.awt.BorderLayout

```
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
    f.setVisible(true);    // LayoutManager el tamaño mínimo para que todos sus
  }                        // componentes se puedan ver. Y se ajusta a ese tamaño.
}
```



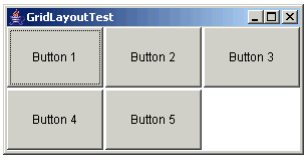
java.awt.CardLayout

```
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);
  }
}
```



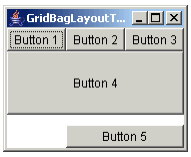
java.awt.GridLayout

```
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);
  }
}
```



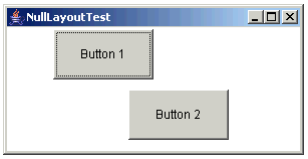
java.awt.GridBagLayout

```
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);
   }
}
```



null LayoutManager

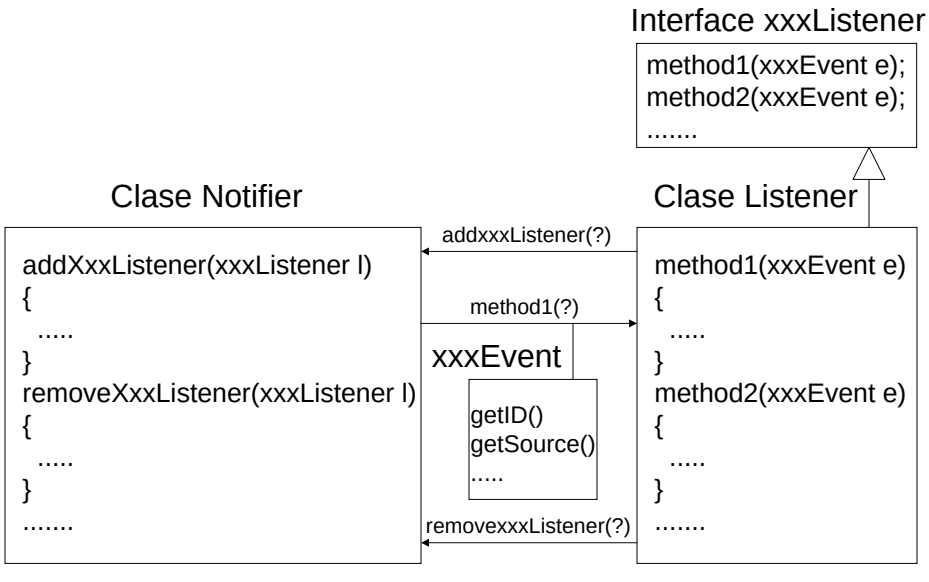
```
import java.awt.Button;
import java.awt.Frame;

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);
  }
}
```



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.

Esquema general



Manejo de eventos

- Los eventos contienen un id (int que describe el tipo de evento). También contiene información sobre el origen del evento (getSource();).
- El manejo de eventos se consigue mediante el uso de interfaces definidos en el paquete java.awt.event
  - ActionListener.
  - WindowListener.
  - KeyListener.
  - MouseListener.

  etc....

Tipos de eventos

- Físicos:

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.

Tipos de eventos

- Semánticos

ActionEvent	Una acción se ha ejecutado.
AdjustmentEvent	Un valor se ha ajustado.
ItemEvent	Un estado ha cambiado.
TextEvent	Un texto ha cambiado.

Origen de eventos

Componente AWT	**Tipos de eventos que pueden generar**										
	Action	Adjustment	Component	Container	Focus	Item	Key	Mouse	Mouse Motion	Text	Window
Button	X		X		X		X	X	X		
Canvas			X		X		X	X	X		
Checkbox			X		X	X	X	X	X		
Choice			X		X	X	X	X	X		
Component			X		X		X	X	X		
Container			X	X	X		X	X	X		
Dialog			X	X	X		X	X	X		X
Frame			X	X	X		X	X	X		X
Label			X		X		X	X	X		

Origen de eventos

Componente AWT	Tipos de eventos que pueden generar										
	Action	Adjustment	Component	Container	Focus	Item	Key	Mouse	Mouse Motion	Text	Window
List	X		X		X	X	X	X	X		
MenuItem	X										
Panel			X	X	X		X	X	X		
Scrollbar		X	X		X		X	X	X		
ScrollPane			X	X	X		X	X	X		
TextArea			X		X		X	X	X	X	
TextField	X		X		X		X	X	X	X	
Window			X	X	X		X	X	X		X

Métodos de los interfaces

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

Métodos de los interfaces

Listener interface	Adapter class	Métodos
KeyListener	KeyAdapter	keyPressed keyReleased keyTyped
MouseListener	MouseAdapter	mouseClicked mouseEntered mouseExited mousePressed mouseReleased
MouseMotionListener	MouseMotionAdapter	mouseDragged mouseMoved
TextListener		textValueChanged

Métodos de los interfaces

Listener interface	Adapter class	Métodos
WindowListener	WindowAdapter	windowActivated windowClosed windowClosing windowDeactivated windowDeiconified windowIconified windowOpened

Adapters

- 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.
- Problema: en Java no existe la herencia múltiple, por ello se suelen usar con las Clases Anónimas.

Ejemplo 1

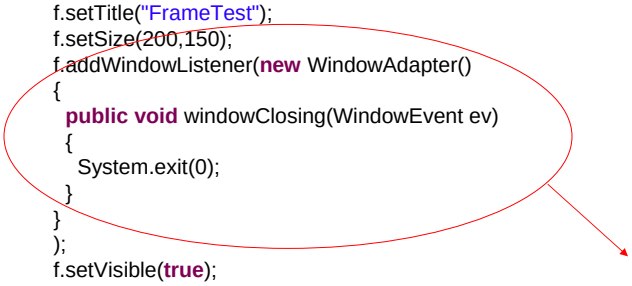
```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class WindowListenerTest extends WindowAdapter
{
  public void windowClosing(WindowEvent ev)
  {
    System.exit(0);
  }
}

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 2

```
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 3

```
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);
  }
}
```



Clase anónima

Registro a la notificación de eventos

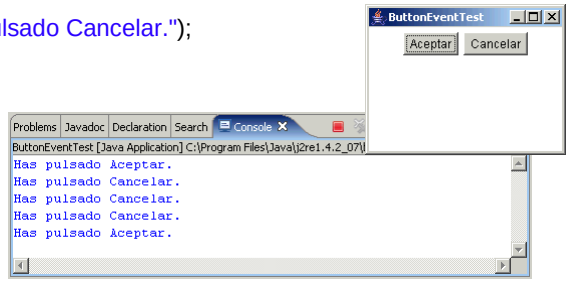
- Cuando invocamos el método addWindowListener, estamos estableciendo un ‘callback’.
- Como parámetro se manda un escuchador, el cual debe implementar el interfaz correspondiente.
- Cuando se genera un WindowEvent como consecuencia de pulsar el botón con la X, el método windowClosing() del escuchador es invocado.
- Se pueden añadir varios escuchadores a un mismo notificador de eventos.

Ejemplo

```
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);
  }
}
```

Ejemplo (cont.)

```
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.");
  }
}
```

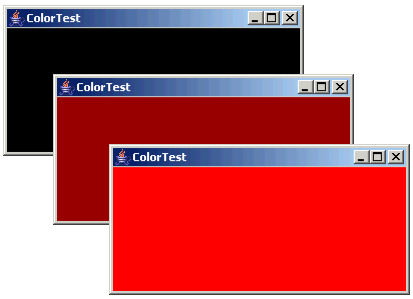


Otras clases: java.awt.Color

- 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

```
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);
     }
    });
 }
}
```



Otras clases: java.awt.Font

- Implementa la representación gráfica de una letra.
- Se define mediante:
  - Familia: nombre del tipo de letra.
  - Estilo: normal o negrita y/o cursiva.
  - Tamaño: pixels del punto utilizado para pintar la 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.

Otras clases: java.awt.Font

- 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:
  - Font.BOLD | Font.ITALIC

Ejemplo

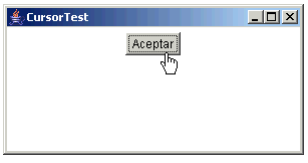
```
import java.awt.*;

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);
  }
}
```

Otras clases: 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

```
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);
  }
}
```



Otras clases: 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();

Otras clases: java.awt.Graphics

- Dibujando figuras:
  - Líneas: drawLine().
  - Rectángulos: drawRect(), fillRect(), clearRect().
  - Rectángulos 3D: draw3DRect(), fill3DRect().
  - Rectáng. redondeados: drawRoundRect(), fillRoundRect().
  - Óvalos: drawOval(), fillOval().
  - Arcos: drawArc, fillArc().
  - Polígonos: drawPolygon(), fillPolygon().
- Escribiendo texto:
  - drawBytes(), drawChars(), drawString().

Ejemplo

```
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);
  }
```

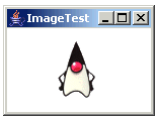

```
  public void paint(Graphics g)
  {
    g.setColor(Color.LIGHT_GRAY);
    g.draw3DRect(10,30,this.getWidth()-20,this.getHeight()-40,true);
    g.setColor(Color.BLACK);
    g.drawLine(15,35,65,65);
    g.drawRect(70,35,50,30);
    g.drawRoundRect(125,35,50,30,10,10);
    g.drawOval(180,35,50,30);
    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);
  }
}
```

Otras clases: java.awt.Graphics

- 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

```
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);
   }
}
```

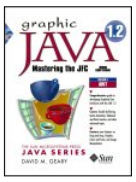


Bibliografía

- Java AWT Reference.
  John Zukowski.
  O'Reilly

- Graphic Java 1.1: Mastering the AWT
  David M. Geary y Allan McClellan.
  Prentice Hall.

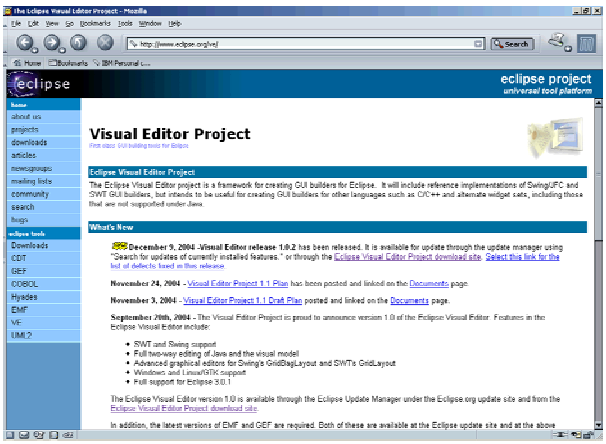
- Graphic Java 2, Volume 1: AWT (3rd edition)
  David M. Geary.
  Prentice Hall.



- The Java tutorial
  http://java.sun.com/docs/books/tutorial/information/download.html#OLDui

Apéndice A: Editor Visual

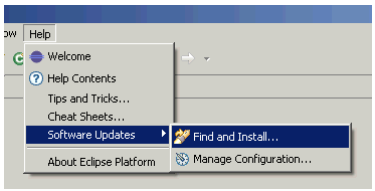
- Existe un editor visual AWT/Swing para Eclipse:



- http://www.eclipse.org/ve

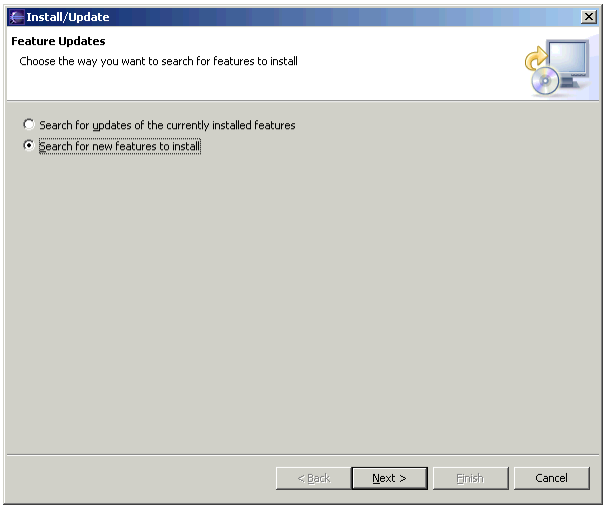
Apéndice A: Editor Visual

- Para realizar la instalación, utilizaremos el "Update Manager" de Eclipse.



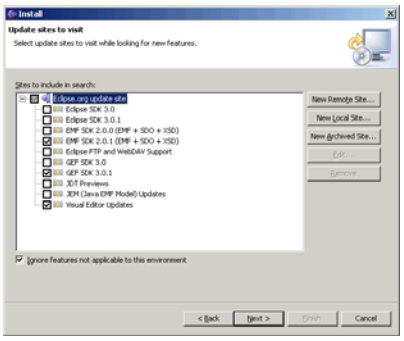
Apéndice A: Editor Visual

- Seleccionar la búsqueda de nuevas "*features*".



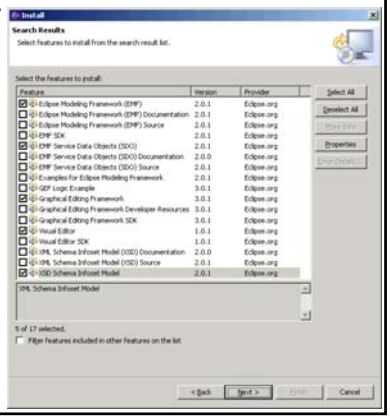
Apéndice A: Editor Visual

- El editor visual tiene algunas dependencias con otros subproyectos de Eclipse.org: EMF y GEF.
- Seleccionar por tanto la búsqueda en los tres subproyectos: EMF 2.0.1, GEF 3.0.1 y VE.



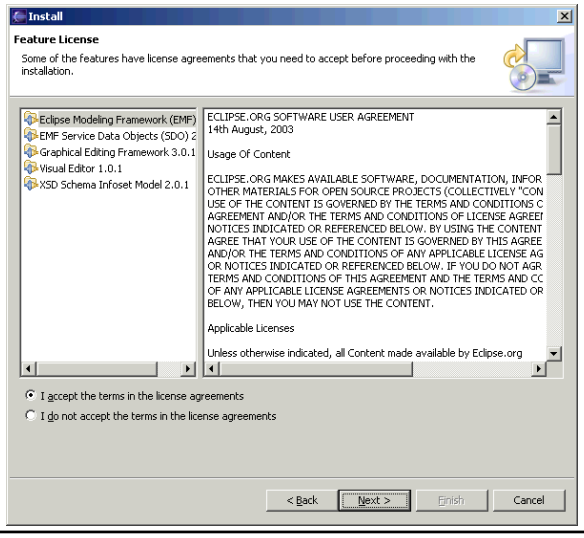
Apéndice A: Editor Visual

- Seleccionar las siguientes "*features*":
  - Eclipse Modeling Framework (EMF) 2.0.1
  - EMF Service Data Object (SDO) 2.0.1
  - Graphical Editing Framework 3.0.1
  - Visual Editor 1.0.1
  - XSD Schema Infoset Model 2.0.1



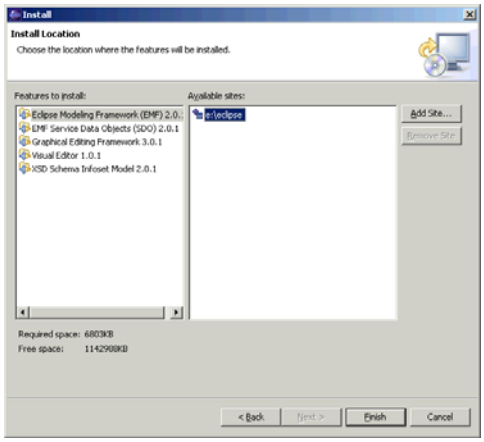
Apéndice A: Editor Visual

- Aceptar los términos de la licencia.



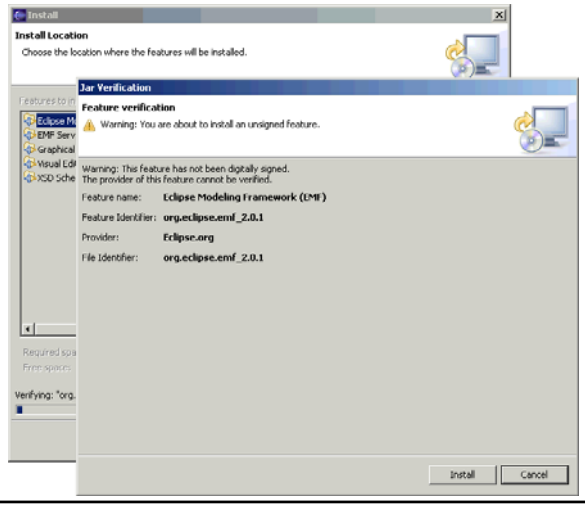
Apéndice A: Editor Visual

- Seleccionar dónde instalar estas nuevas “*features*”.
  - Nota: Solo debiera aparecer el directorio donde está instalado Eclipse.



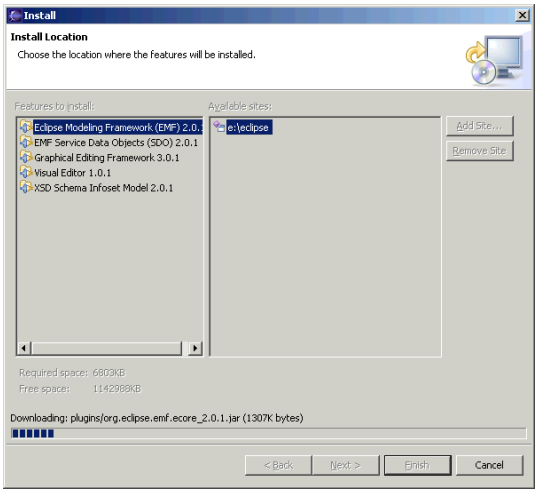
Apéndice A: Editor Visual

- Ir aceptando la instalación, una a una, de todas las “*features*” seleccionadas.



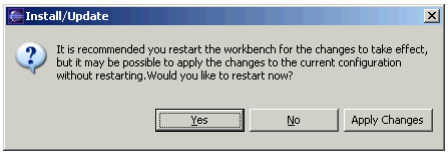
Apéndice A: Editor Visual

- Eclipse irá descargando el software de la web e instalándolo en la máquina automáticamente.

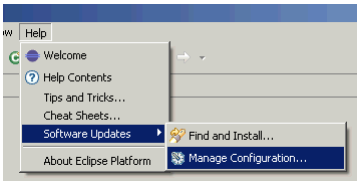


Apéndice A: Editor Visual

- Por último, nos pedirá rearrancar Eclipse.

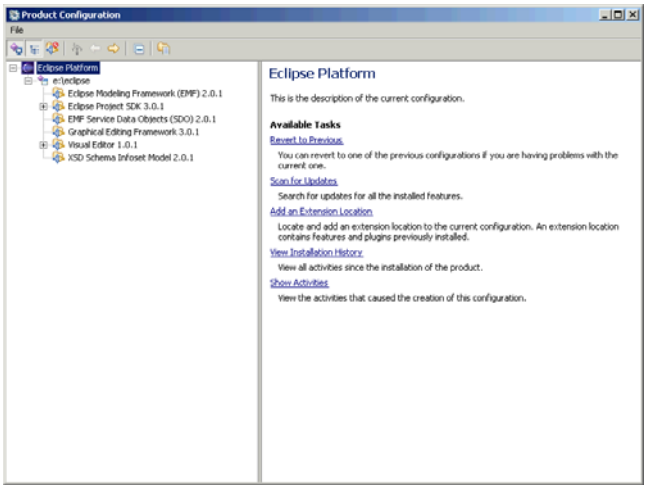


- Podemos chequear las “features” existentes en nuestra instalación a través del “Update Manager”.



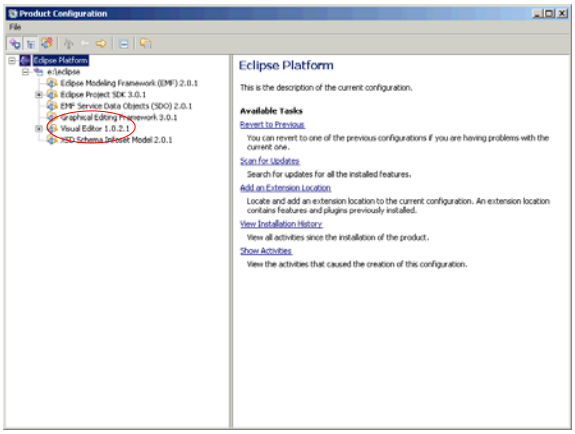
Apéndice A: Editor Visual

- Deberíamos ver activadas todas las “features” que acabamos de instalar.



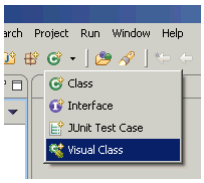
Apéndice A: Editor Visual

- A través del “Update Manager” de nuevo, podremos actualizar el editor visual a la versión 1.0.2.1

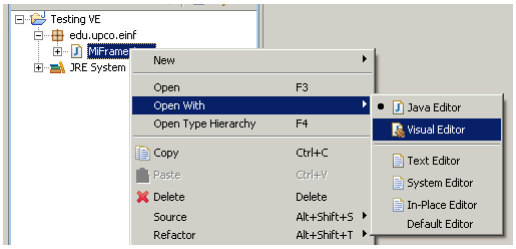


Apéndice B: Usando el Editor Visual

- Crear una clase nueva mediante el nuevo asistente:

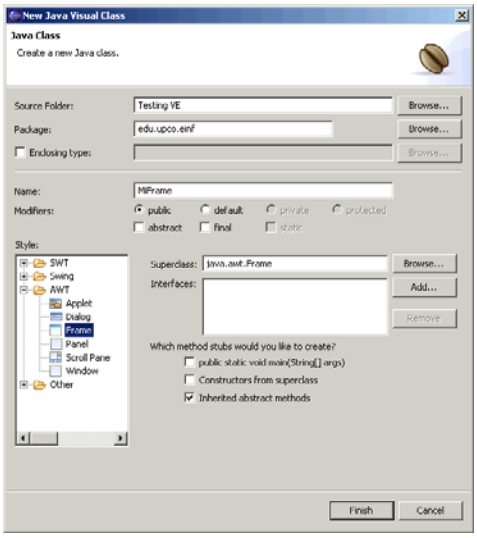


- O abrir una ya existente con la nueva opción del menú de contexto:



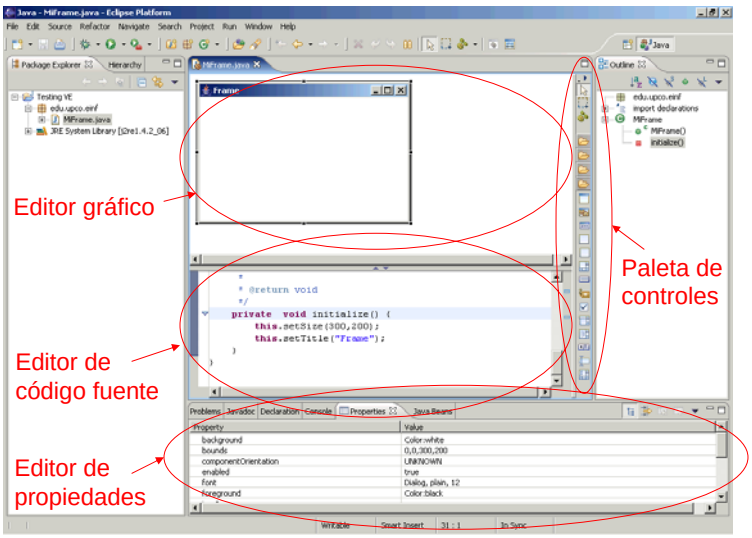
Apéndice B: Usando el Editor Visual

- Apariencia del nuevo asistente:



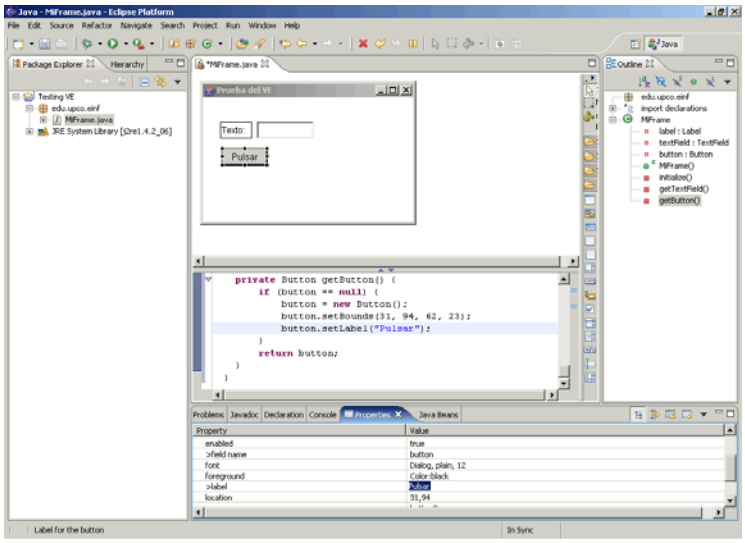
Apéndice B: Usando el Editor Visual

- Apariencia del editor:



Apéndice B: Usando el Editor Visual

- Editando:



Apéndice B: Usando el Editor Visual

- Probando el código visual (en caso de que no tenga un método main):