

1. Programación orientada a componentes
2. Características de un componente
3. JavaBeans
4. Aspectos avanzados de las propiedades
5. Generación de eventos
6. Componentes visuales
7. Descripción detallada de beans

## Ó

- A pesar de la aparente evolución que ha sufrido la programación a lo largo de los últimos 40 años, el desarrollo de una aplicación continúa siendo una actividad costosa en tiempo y dinero, y los errores en el producto final son frecuentes
- El problema es que el nivel de reutilización sigue siendo muy escaso, a pesar de la multitud de librerías, toolkits y frameworks existentes

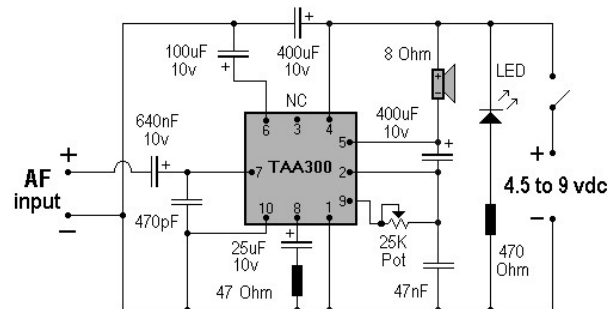
•El uso de librerías gratuitas/comerciales presenta varios inconvenientes:

- Su instalación puede ser compleja
- Es frecuente que tengan dependencias con otras librerías
- Suelen tener una aplicación demasiado general: EEDD, GUIs, comunicación por red, etc. pero rara vez solucionan aspectos de un problema concreto
- Aunque es evidente que ahorran trabajo, sigue siendo necesario una gran cantidad de programación, aunque a otro nivel
- Pueden requerir un tiempo de aprendizaje largo, incompatible con la realización de un proyecto con un plazo de ejecución limitado

•Estos inconvenientes impiden la generalización de la reutilización de código existente en el desarrollo de software

•Por tanto en general cada nuevo proyecto sigue requiriendo la escritura de una gran cantidad de código original

- En cambio, en otras ingenierías como la electrónica el grado de reutilización es altísimo
- En este ámbito, la construcción de un dispositivo electrónico se reduce a la integración de la manera adecuada de distintos componentes comerciales.
- Existe fabricantes especializados en la fabricación de componentes y otros en la de dispositivos finales
- El producto final obtenido es de calidad y se realiza en un tiempo relativamente corto



- La solución a la crisis del software pasa por cambiar el modelo de desarrollo a un desarrollo **orientado a componentes**
- Se basa en la construcción de una aplicación a partir de componentes software comerciales o gratuitos ya existentes, limitando al mínimo necesario el desarrollo de código nuevo
- Para que este cambio se produzca es necesaria la creación de un mercado amplio de componentes software de calidad y con un ámbito de aplicación general

- También será necesaria la adopción de una arquitectura estandarizada que garantice la interoperatividad de los componentes en distintas plataformas, lenguajes de programación y entornos de desarrollo

- La implantación de este modelo tiene multiples beneficios

- El desarrollo será mucho más sencillo y en un tiempo y con un coste menores
- La posibilidad de errores es mínima ya que los componentes son sometidos a un riguroso control de calidad antes de ser comercializados
- Las empresas de desarrollo especializadas pueden obtener ingresos adicionales vendiendo componentes a otras empresas

- También será necesaria la adopción de una arquitectura estandarizada que garantice la interoperatividad de los componentes en distintas plataformas, lenguajes de programación y entornos de desarrollo

- La implantación de este modelo tiene multiples beneficios

- El desarrollo será mucho más sencillo y en un tiempo y con un coste menores
- La posibilidad de errores es mínima ya que los componentes son sometidos a un riguroso control de calidad antes de ser comercializados
- Las empresas de desarrollo especializadas pueden obtener ingresos adicionales vendiendo componentes a otras empresas

## •Un componente debe tener las siguientes características:

- Es una unidad software compilada reutilizable, con una interfaz bien definida
- Se distribuye en un único paquete instalable que contiene en sí todo lo necesario para su funcionamiento, con ninguna o muy pocas dependencias con otros componentes o librerías
- Puede estar implementado en cualquier lenguaje de programación y ser utilizado también para el desarrollo en cualquier lenguaje de programación
- Normalmente es un producto comercial de calidad, realizado por un fabricante especializado. Por supuesto pueden existir componentes gratuitos



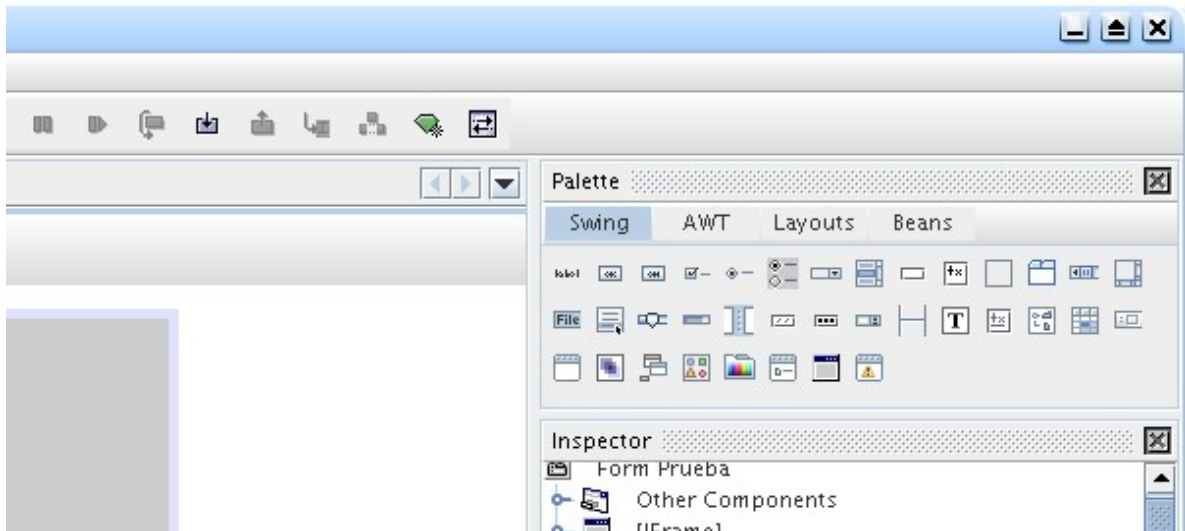
- Un componente puede ser implementado mediante cualquier lenguaje de programación, aunque los lenguajes orientados a objetos son especialmente adecuados para este fin
- El entorno de desarrollo es ahora más que un simple editor de código. Un entorno de desarrollo orientado a componentes debe facilitar la instalación de componentes y su configuración e integración en una aplicación. El código fuente adicional necesario es mínimo

•Existen tres tecnologías de componentes predominantes en la actualidad e incompatibles entre sí:

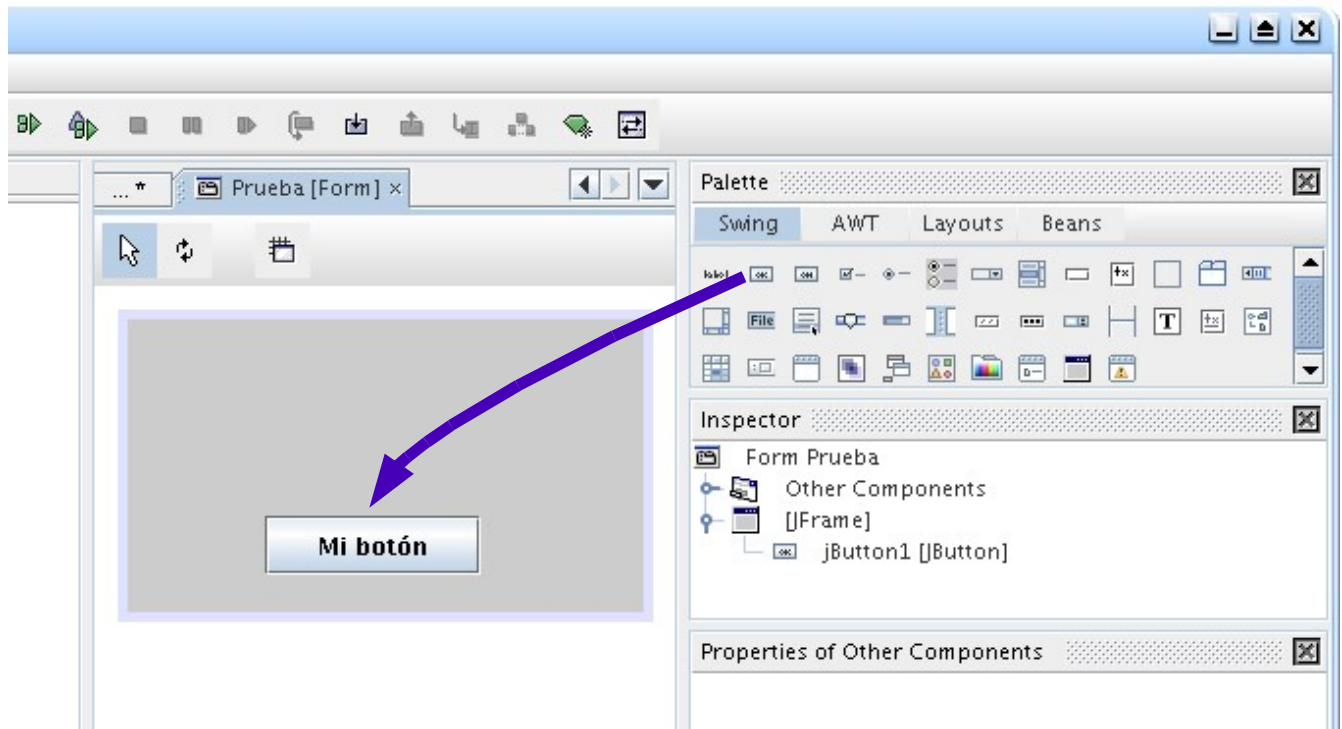
- **VBX/OCX/ActiveX** de Microsoft. Los VBX surgieron como una forma de poder distribuir controles entre los desarrolladores de VisualBasic. VisualBasic puede ser considerado como el primer entorno de desarrollo orientado a componentes que tuvo realmente éxito y aceptación. Los OCX y ActiveX son evoluciones posteriores de los VBX
- **VCL** de Borland. Es la tecnología de componentes utilizados por los entornos de desarrollo Delphi y C++ Builder. Curiosamente estos dos entornos también permiten la utilización de componentes OCX y ActiveX
- **JavaBeans** de Sun. Es la tecnología de componentes basada en Java. En principio no permite la utilización de otro lenguaje de programación. Al contrario que las otras tecnologías, JavaBeans funciona en cualquier plataforma

- Las características funcionales de estas tres tecnologías son similares

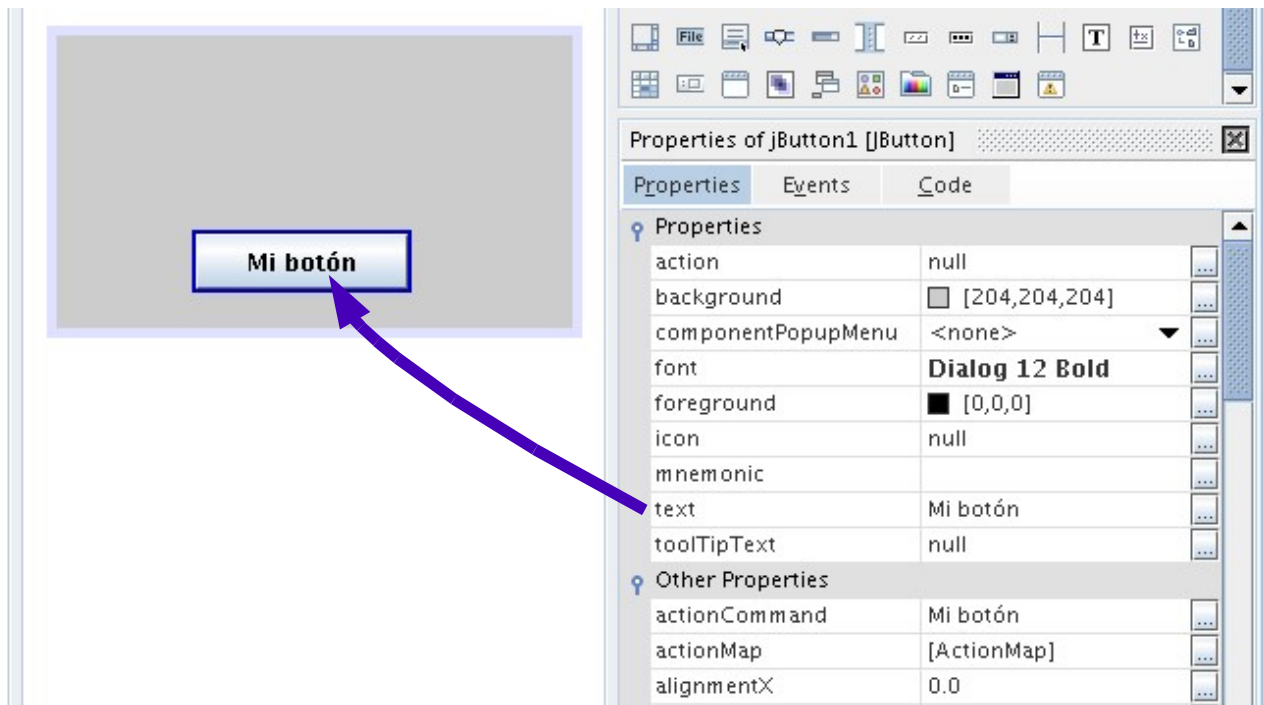
- Los componentes instalados son accesibles en el entorno de desarrollo a través de una paleta de componentes y pueden ser “arrastrados” a la aplicación en la que estamos trabajando



- En el caso de tratarse de componentes “visuales”, es decir, relacionados con la interfaz de usuario, es posible su manipulación de manera interactiva



- Cada componente expone públicamente un conjunto de **propiedades** que determinan su estado interno. La mayoría de estas propiedades pueden ser modificadas de manera interactiva



- Cada componente genera una serie de **eventos** que pueden ser capturados. El código asociado a estos eventos se realiza en el lenguaje nativo del entorno de desarrollo: Visual Basic, Object Pascal, C++, Java, etc.

The image illustrates the process of capturing events in a Java Swing application. It shows a visual representation of a button, a list of its events, and the corresponding code in the IDE.

**Visual Representation:** A window titled "Prueba [Form]" contains a button labeled "Mi botón".

**Properties of jButton1 [JButton]:**

Events	
itemStateChanged	<none>
actionPerformed	jButton1ActionPerformed
componentHidden	<none>
componentMoved	<none>
componentResized	<none>
componentShown	<none>
mouseDragged	<none>
mouseMoved	<none>
propertyChange	<none>
focusGained	<none>

**Code in the IDE:**

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

public class Prueba {
    public void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        jButton1.setText("Mi botón");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        getContentPane().add(jButton1);
        jButton1.setBounds(70, 100, 110, 30);
        pack();
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
}
```

A red arrow points from the "actionPerformed" event in the Properties window to the corresponding code in the IDE.

- JavaBeans es la tecnología de componentes de Java
- Los componentes de JavaBeans se conocen simplemente como **beans**
- En principio un bean es simplemente una clase de objetos, aunque con ciertas características especiales:
  - Es una clase pública que implementa la interfaz Serializable
  - Expone una serie de propiedades que pueden ser leídas y modificadas desde el entorno de desarrollo
  - Expone una serie de eventos que pueden ser capturados y asociados a una serie de acciones
- Aunque los beans han sido pensados para ser utilizados desde el entorno de desarrollo, también pueden ser utilizados directamente

- La apariencia y comportamiento de un bean son determinados por sus propiedades

Una propiedad simple queda identificada por un par de operaciones con la siguiente forma:

```
public <TipoProp> get<NombreProp>() { ... }  
public void set<NombreProp>(<TipoProp> p) { ... }
```

- De esta forma queda definida la propiedad <NombreProp> de tipo <TipoProp>
- Si una propiedad no incluye la operación set, entonces es una propiedad de solo lectura



► Vamos a transformar la clase *TPAviso* (tema 2) en un bean. Vamos a incluir tres propiedades: *Mensaje* (mensaje a imprimir), *Activa* (estado activo o no de la tarea) y *PeriodoSegs* (periodo de ejecución del aviso en segundos)

```
import java.util.*;
import java.awt.event.*;
import java.io.Serializable;
import javax.swing.JOptionPane;
import javax.swing.Timer;

public class TPAviso implements ActionListener, Serializable {
    int periodoSegs;
    String mensaje;
    Timer t;

    public TPAviso() {
        periodoSegs = 60;
        t = new Timer(1000 * periodoSegs, this);
    }

    public TPAviso(String aMensaje, int aPeriodoSegs) {
        periodoSegs = aPeriodoSegs;
        mensaje = aMensaje;
        t = new Timer(1000 * periodoSegs, this);
    }

    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, mensaje, "Aviso",
            JOptionPane.INFORMATION_MESSAGE);
        setActiva(false);
    }
}
```

Propiedad  
Activa



```
public void setActiva(boolean valor) {  
    if (valor == true)  
        t.start();  
    else  
        t.stop();  
}
```

```
public boolean getActiva() {  
    return t.isRunning();  
}
```

Propiedad  
Mensaje



```
public void setMensaje(String aMensaje) {  
    mensaje = aMensaje;  
}
```

```
public String getMensaje() {  
    return mensaje;  
}
```

Propiedad  
Periodo



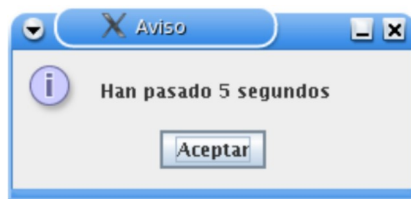
```
public void setPeriodoSegs(int aPeriodoSegs) {  
    periodoSegs = aPeriodoSegs;  
    t.setInitialDelay(1000 * periodoSegs);  
    if (t.isRunning()) {  
        t.restart();  
    }  
}
```

```
public int getPeriodoSegs() {  
    return periodoSegs;  
}
```

```
}
```

► Naturalmente este bean es utilizable como cualquier otra clase de Java. En principio no tiene ninguna cualidad extraordinaria. Los valores de las propiedades pueden ser modificados mediante las operaciones set correspondientes

```
public class TPAvisoPrueba {  
    public static void main(String[] args) {  
  
        TPAviso tpa = new TPAviso();  
        tpa.setPeriodoSegs(5);  
        tpa.setMensaje( Han pasado 5 segundos );  
        tpa.setActiva(true);  
  
        try {  
            System.in.readLine();  
        }  
        catch(IOException e) {};  
    }  
}
```



- La forma habitual de distribución de un bean es en un fichero *jar*. Este fichero debe contener además un fichero *manifest* con el siguiente contenido:

```
Name: <NombreBean>.class  
Java-Bean: True
```

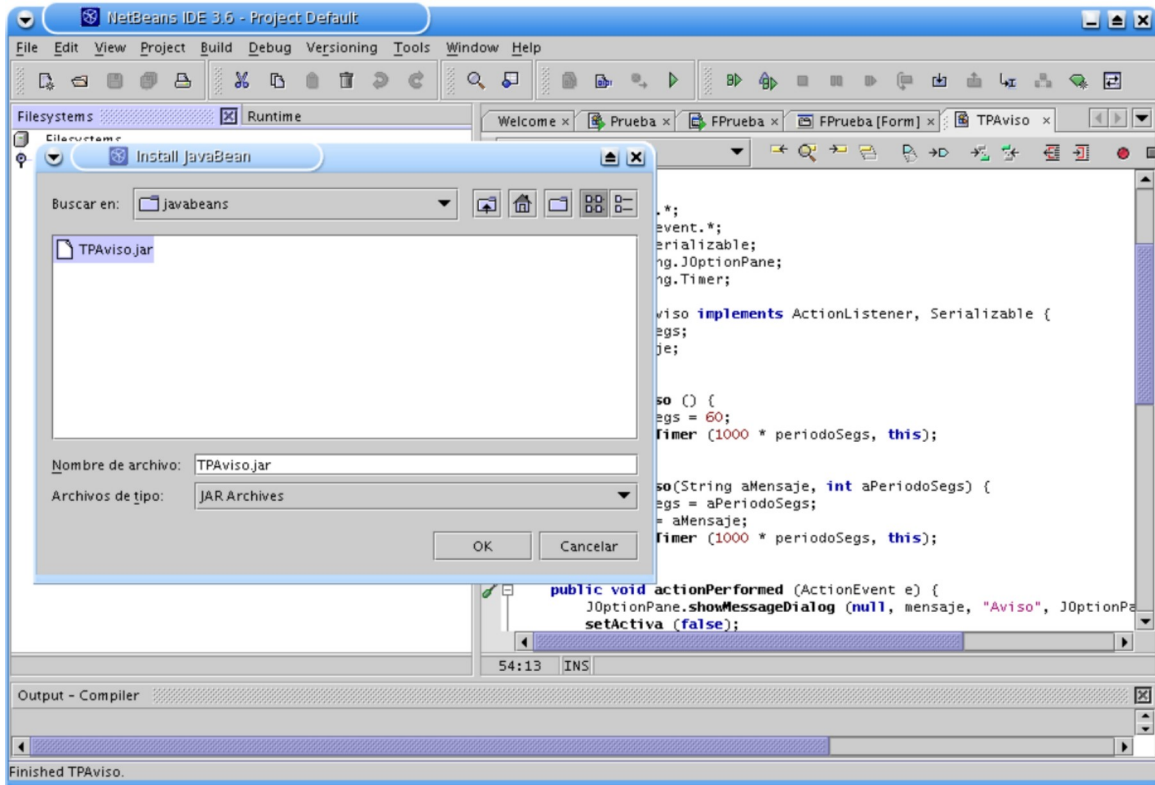
- ▶ Para empaquetar el bean definimos un fichero manifest.tmp con el siguiente contenido:

```
Name: TPAviso.class  
Java-Bean: True
```

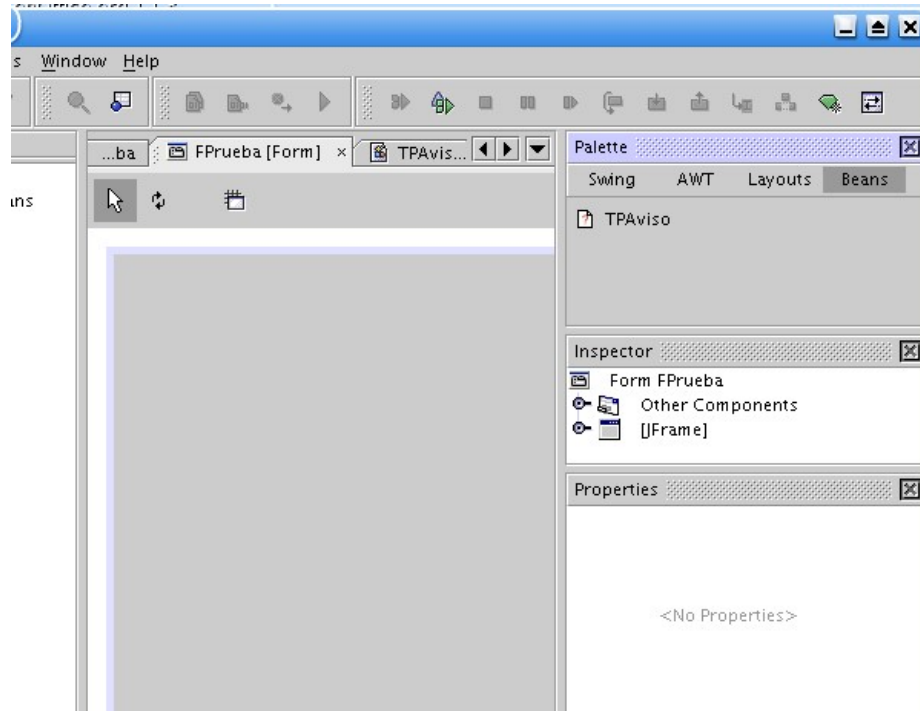
- ▶ A continuación generaríamos el fichero *jar*:

```
jar cfm TPAviso.jar manifest.tmp TPAviso.class
```

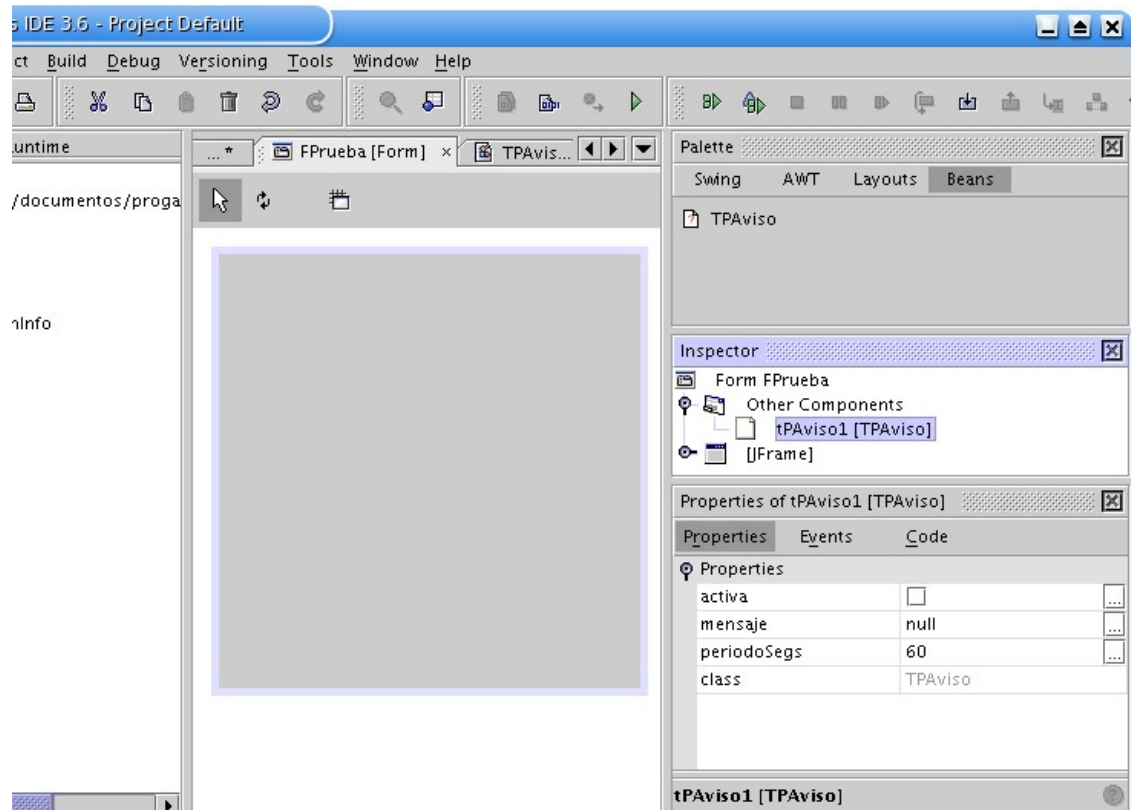
► Ya podemos instalar el bean en el netBeans (o cualquier otro entorno de desarrollo Java):



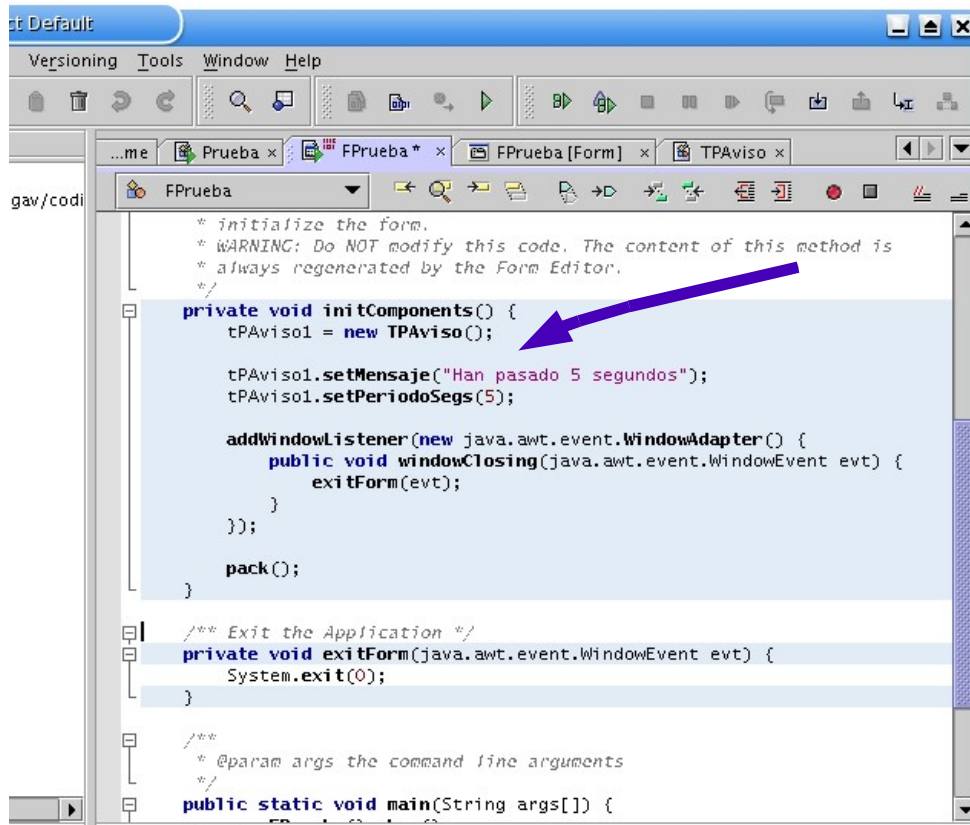
- Y lo tendremos disponible en la paleta de componentes:



- Podemos seleccionarlo y utilizarlo en la aplicación. Sus propiedades son editables de manera interactiva



- netBeans se encarga de generar automáticamente el código de creación del componente y asignación de las propiedades:



```

* initialize the form.
* WARNING: Do NOT modify this code. The content of this method is
* always regenerated by the Form Editor.
*/
private void initComponents() {
    TPAviso1 = new TPAviso();

    TPAviso1.setMensaje("Han pasado 5 segundos");
    TPAviso1.setPeriodoSegs(5);

    addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(java.awt.event.WindowEvent evt) {
            exitForm(evt);
        }
    });

    pack();
}

/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt) {
    System.exit(0);
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {

```



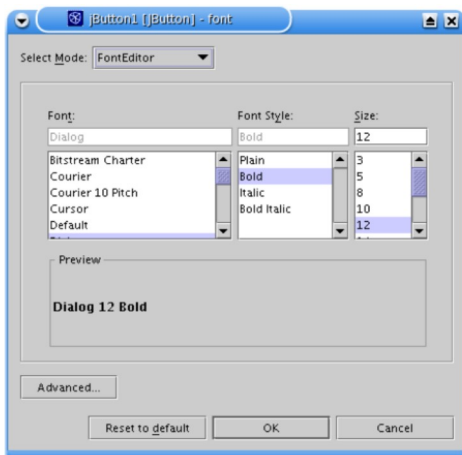
- No hay que confundir “propiedad” con “atributo”

- Ambos describen el estado interno y comportamiento, pero en dos contextos diferentes
- Normalmente los atributos nunca son públicos. En cambio las propiedades siempre son características públicas de los componentes
- Los atributos son accesibles únicamente a nivel de programación. Las propiedades son accesibles tanto a nivel de programación como interactivamente desde el entorno de desarrollo

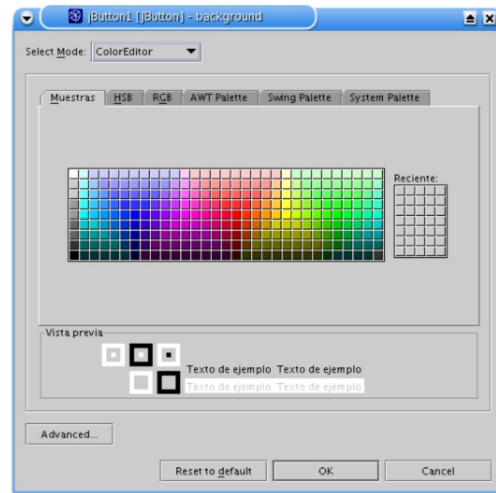
- Cuando se implementa un componente mediante una clase de objetos, cada propiedad suele estar asociada internamente a un atributo, aunque esto no ocurre siempre

- Una vez instalado el bean, el entorno de desarrollo es capaz de identificar sus propiedades simplemente detectando parejas de operaciones get/set, mediante la capacidad de Java denominada **introspección**
- El entorno de desarrollo es capaz de editar automáticamente cualquier propiedad de los tipos básicos y las clases *Color* y *Font*. Sin embargo, lógicamente no tiene capacidad para editar una propiedad de un tipo arbitrario (p. e., de la clase *Cliente*, *Producto*, etc.)
- JavaBeans permite definir editores a medida para editar estas propiedades

- Para ello seguiremos los siguientes pasos:
  - Construir una clase para la edición de la propiedad que implemente la interfaz *PropertyEditor*
  - El nombre de esta clase debe coincidir con el de la propiedad, terminando en “Editor” (por ejemplo, *ClienteEditor*)
  - Empaquetar el editor en el fichero *jar* del bean. El entorno de desarrollo utilizará el editor cuando necesite mostrar/editar la propiedad.
- Ver “Bean Customization” del tutorial Java para más información



FontEditor



ColorEditor

- Es posible definir también propiedades indexadas. Estas propiedades representan colecciones de elementos y se identifican mediante los siguientes patrones de operaciones:

```
public <TipoProp>[] get<NombreProp>()  
public void set<NombreProp> (<TipoProp>[] p)  
public <TipoProp> get<NombreProp>(int index)  
public void set<NombreProp> (int index, <TipoProp> p)
```

- La edición de este tipo de propiedades requiere la implementación de un editor a medida, como hemos estudiado anteriormente

## Ó

- Un bean puede generar una serie de eventos, que pueden ser capturados y procesados externamente
- Los pasos a seguir para incluir el soporte de un evento en un bean son los siguientes:

- Definir una interfaz que represente el listener asociado al evento. Esta interfaz debe incluir una operación para el procesamiento del evento
- Definir dos operaciones, para añadir y eliminar listeners. Estos listeners deben ser almacenados internamente en una estructura de datos como *ArrayList* o *LinkedList*:

```
public void add<NombreListener>(<NombreListener> l)  
public void remove<NombreListener>(<NombreListener> l)
```

- Finalmente, recorrer la estructura de datos interna llamando a la operación de procesamiento del evento de todos los listeners registrados

► Vamos a incluir soporte de eventos en el bean *TPAviso*. Cada vez que se lance el aviso se generará un evento *EventoAviso* que puede ser capturado por aquellas clases interesadas

```
public class TPAviso implements ActionListener, Serializable {
    int periodoSegs;
    String mensaje;
    Timer t;

    LinkedList listeners;

    public TPAviso() {
        periodoSegs = 60;
        t = new Timer(1000 * periodoSegs, this);
        listeners = new LinkedList();
    }

    public TPAviso(String aMensaje, int aPeriodoSegs) {
        periodoSegs = aPeriodoSegs;
        mensaje = aMensaje;
        t = new Timer(1000 * periodoSegs, this);
        listeners = new LinkedList();
    }

    public void addEventoAvisoListener(EventoAvisoListener l) {
        listeners.add(l);
    }

    public void removeEventoAvisoListener(EventoAvisoListener l) {
        listeners.remove(l);
    }
}
```

```
private void fireEventAviso(EventoAviso ea)
{
    EventoAvisoListener eal;
    iterator i = listeners.listIterator(0);

    while (i.hasNext()) {
        eal = (EventoAvisoListener) i.next();
        eal.procesarAviso(ea);
    }
}

public void actionPerformed(ActionEvent e) {
    fireEventAviso(new EventoAviso(this));
    JOptionPane.showMessageDialog(null, mensaje, "Aviso",
        JOptionPane.INFORMATION_MESSAGE);
    setActiva(false);
}

// Resto de operaciones del bean
...
}
```

```
interface EventoAvisoListener extends java.util.EventListener {
    public void procesarAviso(EventoAviso ev);
}
```

```
class EventoAviso extends java.util.EventObject {
    EventoAviso(Object s) {
        super(s);
    }
}
```

- La mayoría de los componentes que se utilizan son visuales, es decir, implementan elementos de la interfaz de usuario
- Muchos de los elementos de Swing son componentes visuales
- Construir un nuevo componente visual (basado en Swing) es muy sencillo: tan solo hay que construir una clase que herede de la clase *JComponent* o de cualquiera de sus descendientes



► Vamos a construir un componente que funcione como un enlace a una página web. Es decir, cuando sea pulsado por el ratón lanzará una página determinada en el navegador web

```
import java.beans.*;
import java.awt.*;
import java.awt.event.*;
import java.io.Serializable;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

public class URLLink extends JLabel implements Serializable {

    private String URL;
    private String navegador;

    public class URLLinkMouseListener extends MouseAdapter {
        public void mouseClicked(MouseEvent e) {
            irURL();
        }
    }

    public URLLink() {
        setURL("");
        setNavegador("");
        setText("URL"); // Texto por defecto del JLabel
        setForeground(Color.blue); // Color del texto
        setCursor(new Cursor(Cursor.HAND_CURSOR)); // Cursor (mano apuntadora)
        setIcon(new javax.swing.ImageIcon("url.png")); // Triángulo azul
        addMouseListener(new URLLinkMouseListener()); // Añadir listener
    }
}
```

```
public String getURL() {          // Propiedad URL
    return URL;
}

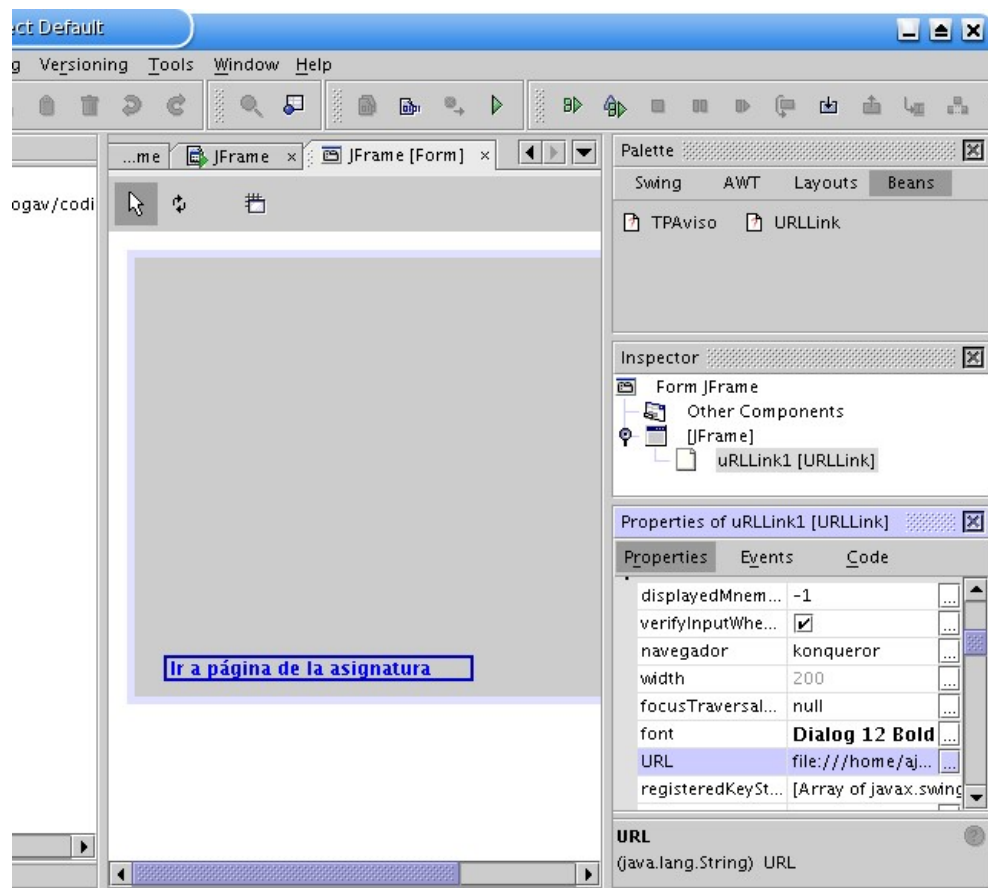
public void setURL(String value) {
    String aURL = URL;
    URL = value;
}

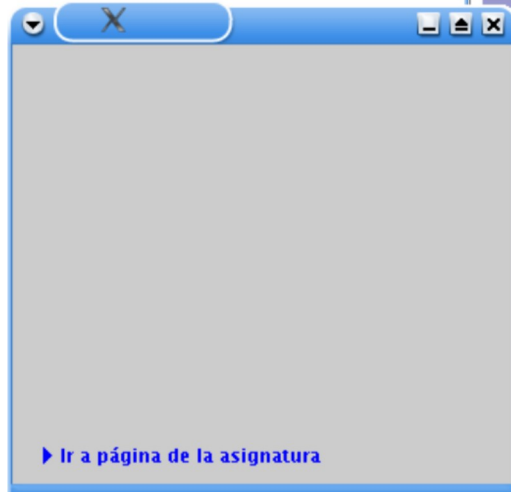
public String getNavegador() {    // Propiedad Navegador
    return navegador;
}

public void setNavegador(String value) {
    String aNavegador = navegador;
    navegador = value;
}

public void irURL() {
    try {
        Runtime.getRuntime().exec(navegador + " " + URL);
    }
    catch (java.io.IOException e) {
        JOptionPane.showMessageDialog(this, "No es posible ejecutar navegador",
            "Error", JOptionPane.ERROR_MESSAGE);
    }
}
}
```

- El componente hereda gran cantidad de propiedades y eventos de *JLabel*





Programación Avanzada - Konqueror

Dirección: [ne/ajrueda/documentos/progav/www/index.html](http://ne/ajrueda/documentos/progav/www/index.html)

Universidad de Jaén Departamento de Informática

# Programación Avanzada

*Programación orientada a objetos y técnicas avanzadas de programación con Java*

**1º Ingeniería en Informática**  
**Profesor:** [Antonio Jesús Rueda Ruiz](#)

## Índice de la asignatura

1. Introducción a la OOP y al lenguaje Java
2. Programación orientada a objetos en Java
3. Programación en entornos gráficos de usuario
4. Tecnologías de componentes
5. Programación en Internet
6. Programación distribuida

## Evaluación

La nota final de la asignatura se calculará a partir de la nota obtenida en prácticas (40%) y una prueba teórica final (60%). Eventualmente se tendrán en cuenta la nota obtenida en trabajos voluntarios propuestos en clase.

## Ó

- El mecanismo automático de introspección es suficiente para que el entorno obtenga toda la información de propiedades y eventos del bean
- Sin embargo hay otros aspectos adicionales no contemplados:
  - El icono que va a representar el bean en la paleta
  - La descripción de cada una de las propiedades
  - Especificar qué propiedades deben ser visualizadas/editables en el editor de propiedades
  - El cuadro de propiedades tiene varios apartados: propiedades principales, otras propiedades, layout, accesibilidad, etc. ¿Como asignar una propiedad a un apartado determinado, y como crear otros apartados?

- Junto a un bean es posible crear una clase auxiliar **<NombreBean>BeanInfo** que va a proporcionar toda esta información. Esta clase debe extender la clase base *SimpleBeanInfo*
- NetBeans permite generar automáticamente esta clase y realizar gran parte de su configuración de manera interactiva

