

Construir Interfaces Gráficas de Usuario

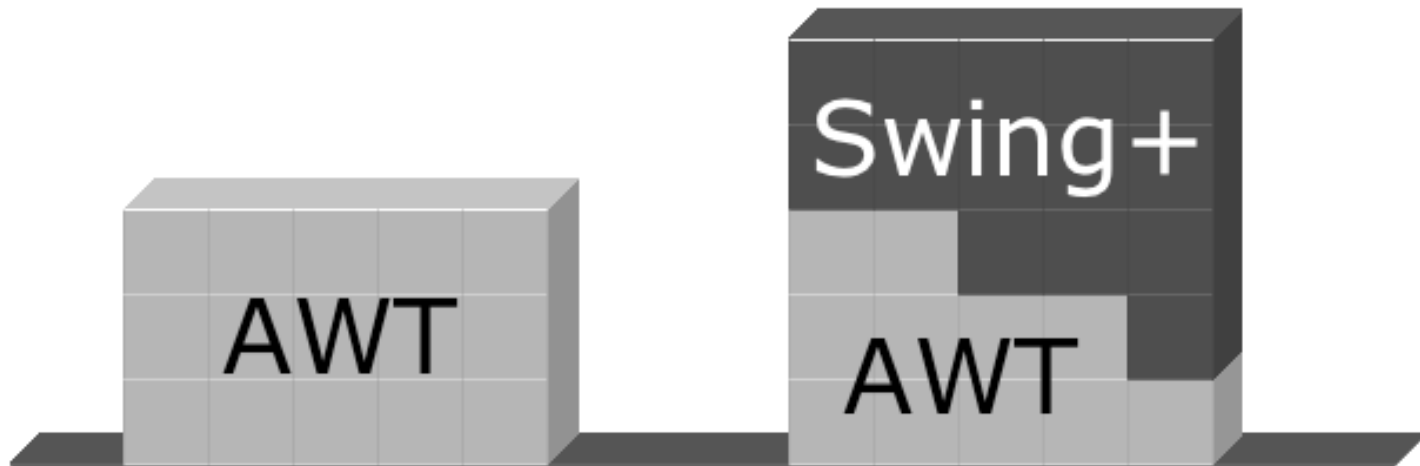
Principales Conceptos a ser Cubiertos

- Construcción de la IGU
- Componentes de la interfaz
- Esquema de disposición de los componentes
- Manejo de eventos

Principios de IGU

- Componentes: Bloques básicos la IGU.
 - Botones, menús, barras, etc.
- Disposición de los componentes: Orden para hacer usable una IGU.
 - Uso de *Gestores de Disposición*.
- Eventos: reacción ante la entrada generada por el usuario.
 - Presión de un botón, selección en un menú de opciones, etc.

AWT y Swing



Crear una Ventana

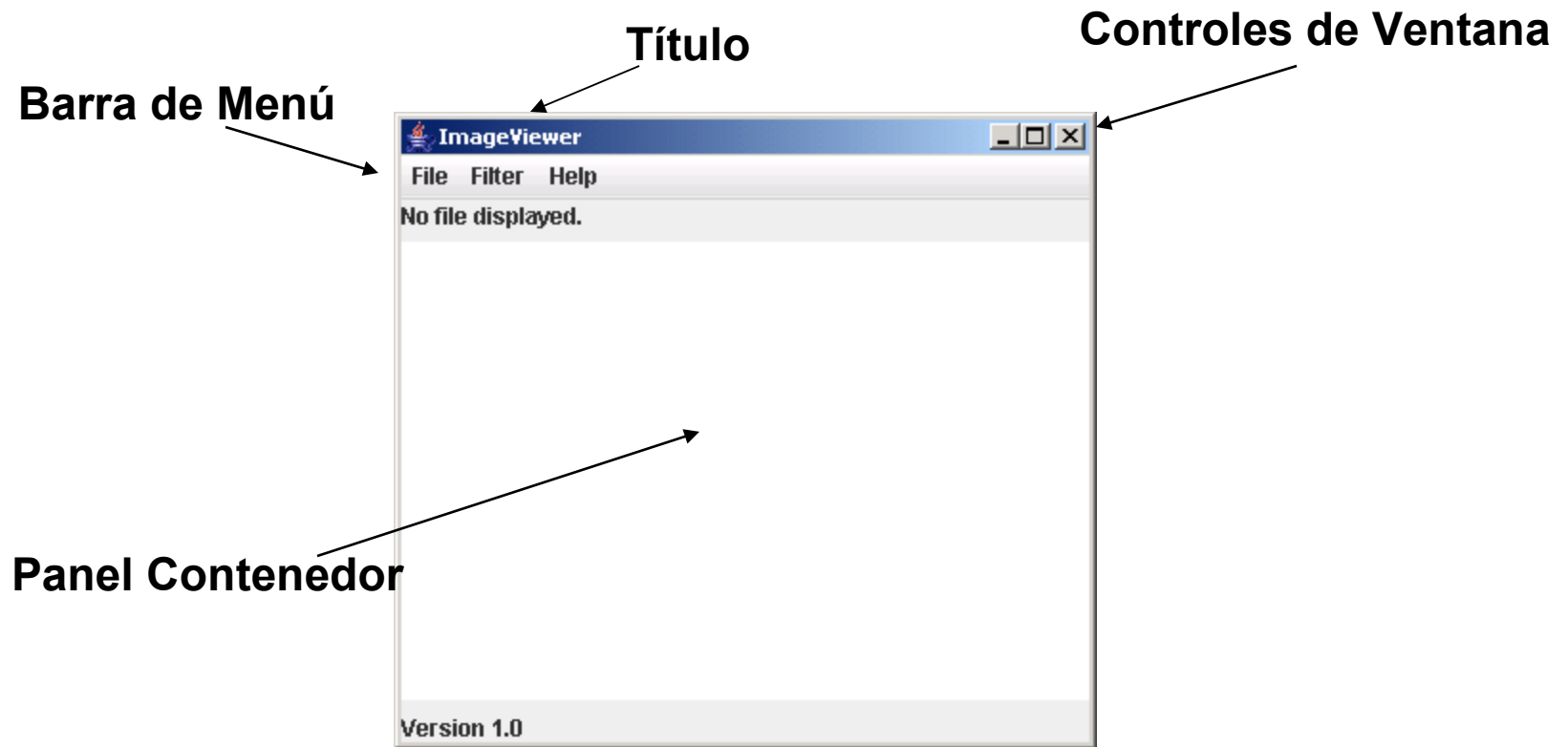
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class VisorDeImagen
{
    private JFrame ventana;

    /**
     * Crea un VisorDeImagen y lo muestra en pantalla.
     */
    public VisorDeImagen()
    {
        construirVentana();
    }

    // se omite el resto de la clase.
}
```

Elementos de una Ventana



El Panel Contenedor

```
/**
 * Crea la ventana Swing y su contenido.
 */
private void construirVentana()
{
    ventana = new JFrame("Visor de Imágenes");
    Container panelContenedor = ventana.getContentPane();

    JLabel etiqueta = new JLabel("I am a label.");
    panelContenedor.add(etiqueta);

    ventana.pack();
    ventana.setVisible(true);
}
```



Agregar Menús

- **JMenuBar**
 - Se muestra debajo del título.
 - Contiene los menús.
- **JMenu**
 - ej. *Archivo*. Contiene los elementos del menú.
- **JMenuItem**
 - ej. *Abrir*. Un único elemento.


```
private void ConstruirBarraDeMenu(JFrame ventana)
{
    JMenuBar barraDeMenu = new JMenuBar();
    ventana.setJMenuBar(barraDeMenu);

    // crea el menu Archivo
    JMenu menuArchivo = new JMenu("Archivo");
    barraDeMenu.add(menuArchivo);

    JMenuItem elementoAbrir = new JMenuItem("Abrir");
    menuArchivo.add(elementoAbrir);

    JMenuItem elementoSalir = new JMenuItem("Salir");
    menuArchivo.add(elementoSalir);
}
```

Manejo de Eventos

- Los eventos corresponden a la interacción del usuario con los componentes.
- Los componentes se asocian con diferentes tipos de eventos.
 - Las ventanas se asocian con **WindowEvent**.
 - Los menús se asocian con **ActionEvent**.
- Cuando ocurre un evento se puede notificar a los Objetos.
 - Tales objetos se llaman *oyentes de eventos*.

Recepción Centralizada de Eventos

- Un único objeto maneja todos los eventos.
 - Implementa la interfaz **ActionListener**.
 - Define un método **actionPerformed**.
- Registra un oyente con cada componente.
 - **elemento.addActionListener(this)**
- Debe determinar cuál componente ha generado el evento.

```

public class VisorDeImagen implements ActionListener
{
    ...
    public void actionPerformed(ActionEvent e)
    {
        String comando = e.getActionCommand();
        if(comando.equals("Abrir")) {
            ...
        }
        else if (comando.equals("Salir")) {
            ...
        }
        ...
    }
    ...
    private void ConstruirBarraDeMenu(JFrame ventana)
    {
        ...
        elementoAbrir.addActionListener(this);
        ...
    }
}

```

Recepción Centralizada de Eventos

- Este abordaje funciona.
- Sin embargo...
 - No es suficientemente flexible.
 - La identificación de componentes por el texto del nombre es frágil.
- Se prefiere una aproximación alternativa.

Sintaxis de las Clases Internas

- Las definiciones de clases pueden ser internas a otra clase.

```
- public class ClaseEnvolvente
{
    ...
    private class ClaseInterna
    {
        ...
    }
}
```

Clases Internas

- Las instancias de las clases internas se encuentran dentro de la clase envolvente.
- Las instancias de las clases internas tienen acceso a los miembros privados de la clase envolvente.

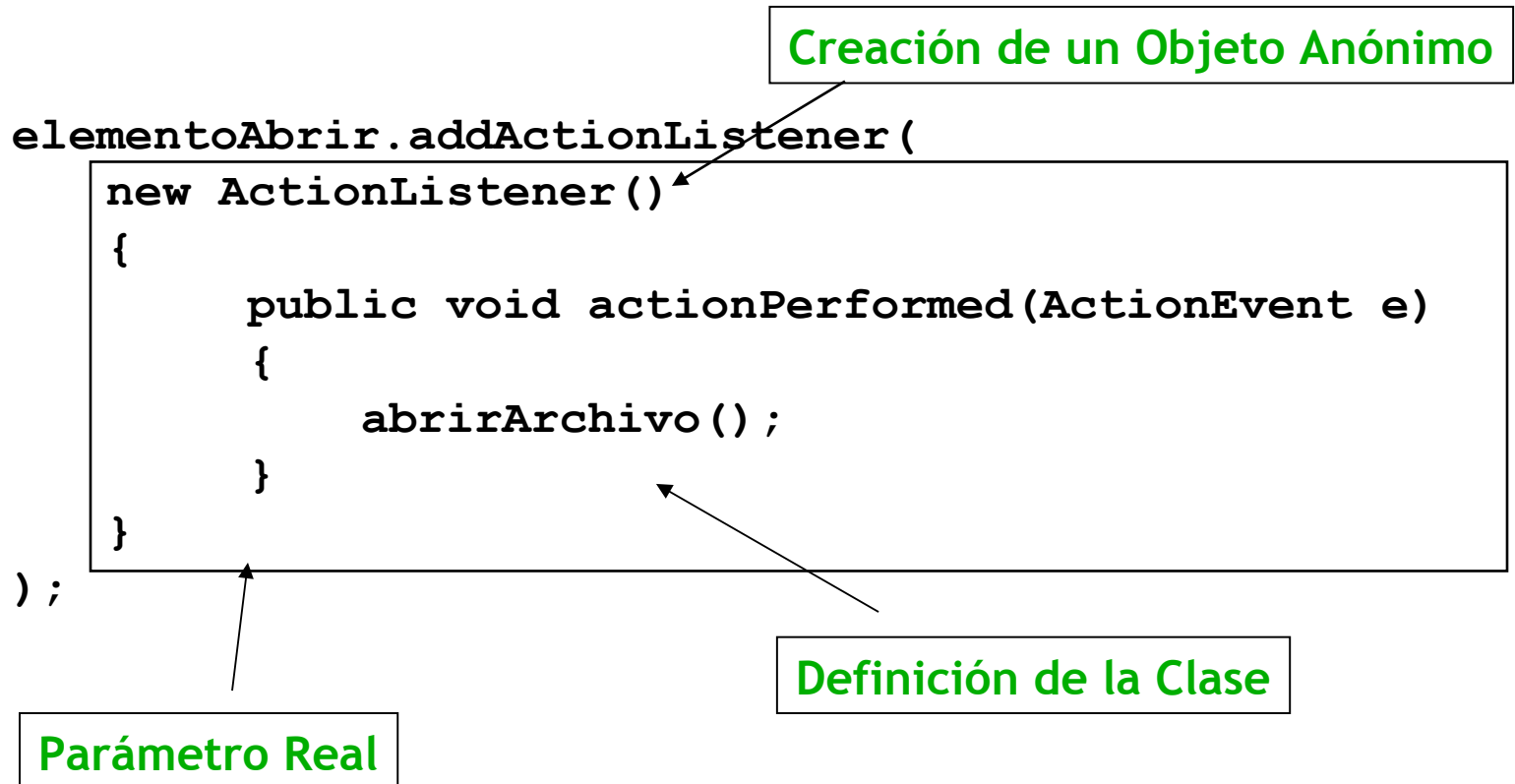
Clases Internas Anónimas

- Obedecen las reglas de las clases internas.
- Se utilizan para crear objetos de un único uso que no se requiere que tengan un nombre.
- Se utiliza una sintaxis especial.
- La instancia se referencia a través del supertipo, porque no tiene nombre de subtipo.

Oyente Anónimo

```
JMenuItem elementoAbrir = new JMenuItem("Abrir");  
  
elementoAbrir.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        abrirArchivo();  
    }  
});
```

Elementos de Clase Anónimos

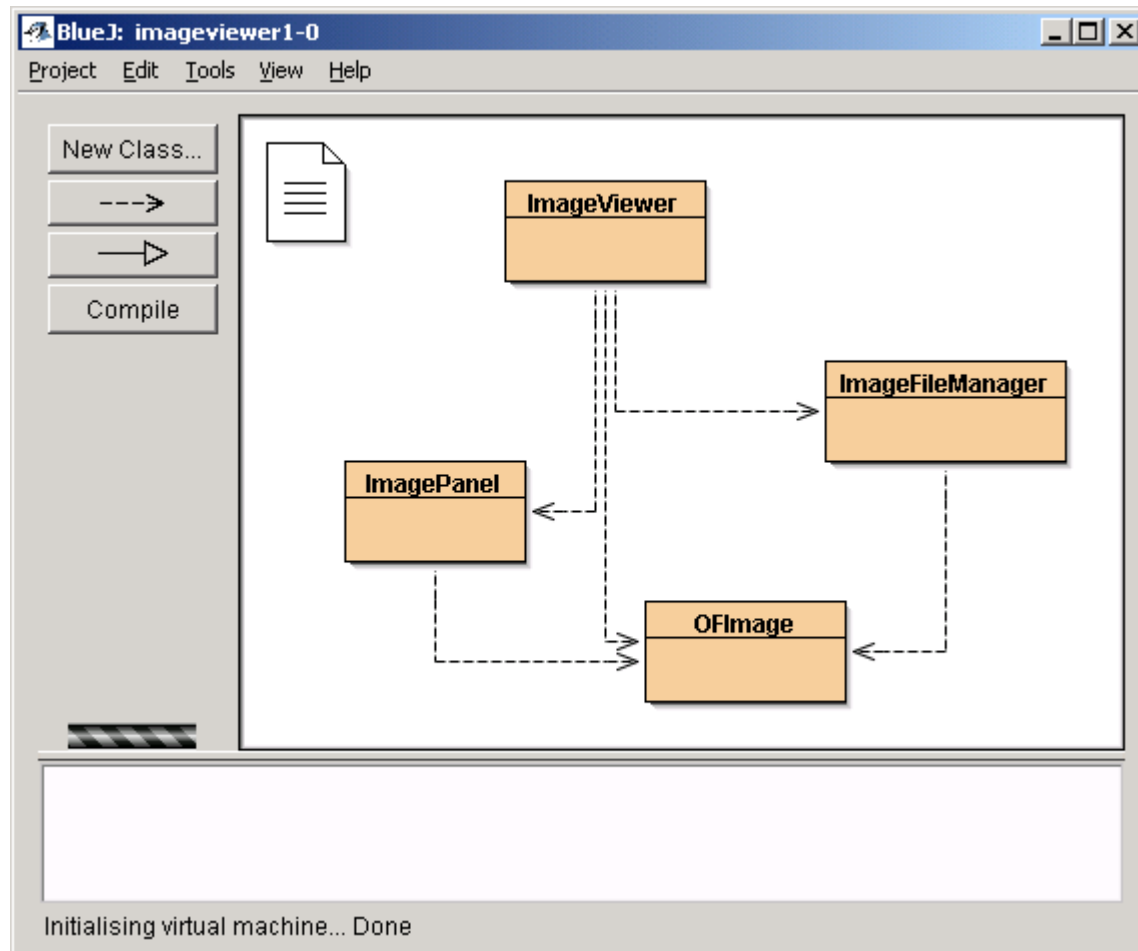


Salir al Cerrar la Ventana

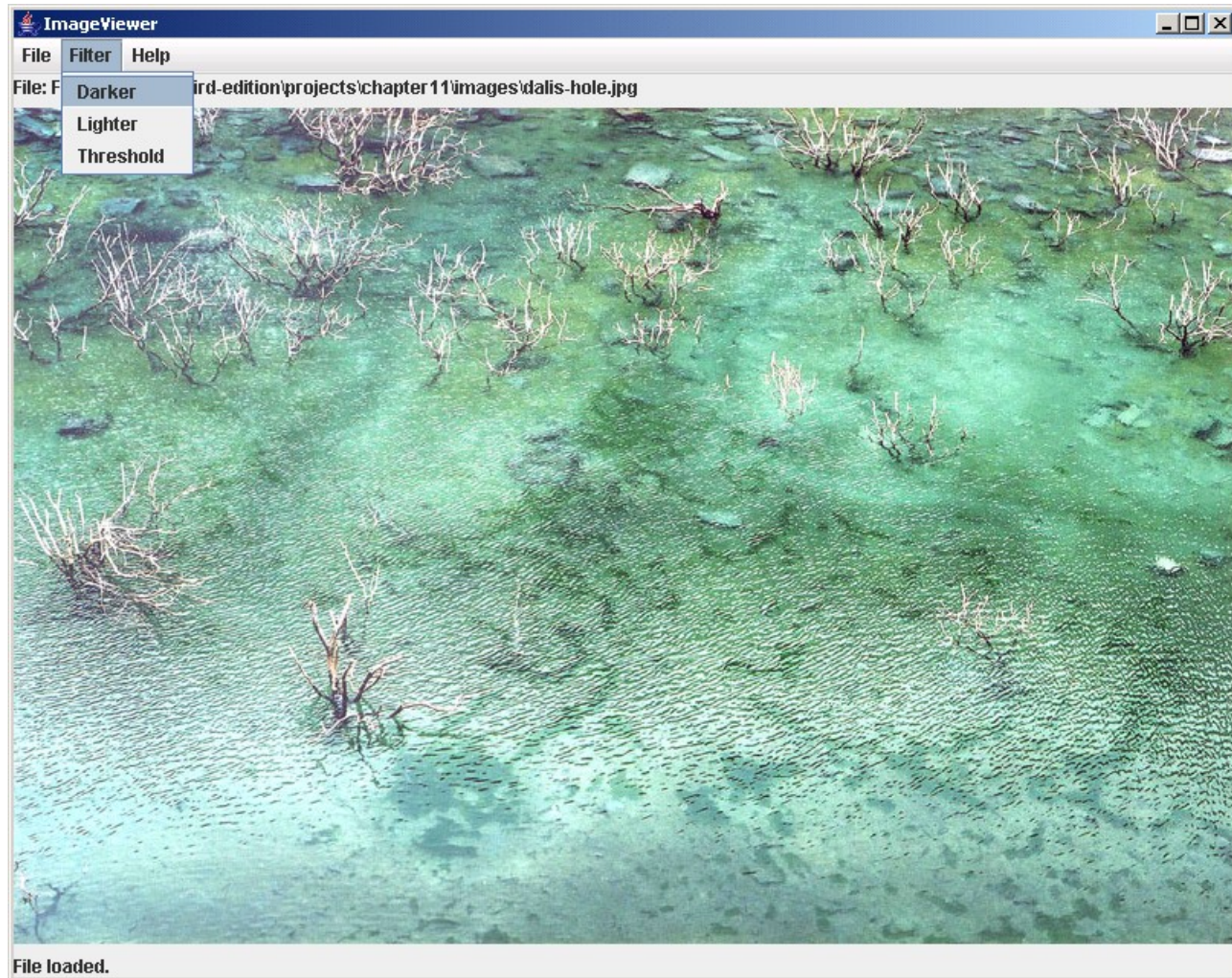
```
ventana.addWindowListener(new WindowAdapter() {  
    public void windowClosing(WindowEvent e)  
    {  
        System.exit(0);  
    }  
});
```

WindowAdapter provee una
implementación no-op de la interfaz
WindowListener.

El Proyecto VisorDeImagen



Procesamiento de Imagen



Responsabilidades de la Clase

- **VisorDeImagen**
 - Configura la estructura de la IGU.
- **AdministradorDeArchivos**
 - Métodos estáticos para carga y grabación de archivos de imagen.
- **PanelDeImagen**
 - Muestra la imagen dentro de la IGU.
- **ImagenOF**
 - Modela una imagen 2D.

ImagenOF

- Nuestra subclase de **BufferedImage**.
- Representa un arreglo 2D de píxeles.
- Métodos importantes:
 - **getPixel, setPixel**
 - **getAncho, getAlto**
- Cada pixel tiene un color.
 - Utilizamos **java.awt.Color**.

Agregando un PanelDeImagen

```
public class VisorDeImagen
{
    private JFrame ventana;
    private PanelDeImagen panelDeImagen;

    ...

    private void construirVentana()
    {
        Container panelContenedor
            = ventana.getContentPane();
        panelDeImagen = new PanelDeImagen();
        panelContenedor.add(panelDeImagen);
    }

    ...
}
```


Agregar la imagen

```
public class VisorDeImagen
{
    private JFrame ventana;
    private PanelDeImagen panelDeImagen;

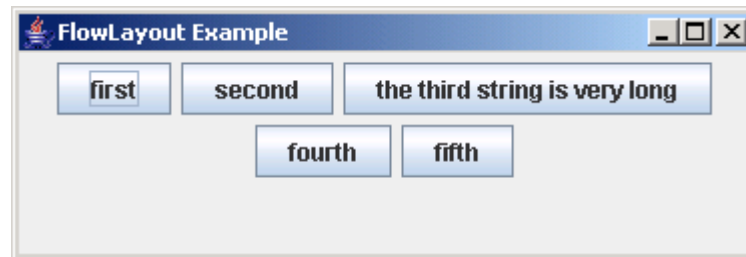
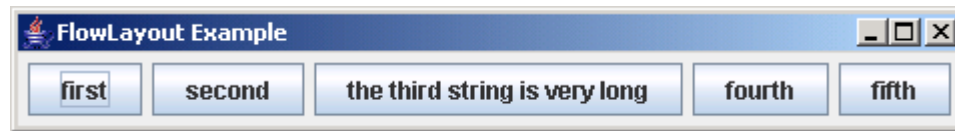
    ...

    private void abrirArchivo()
    {
        File archivoSeleccionado = ...;
        ImagenOF imagen =
            AdministradorDeArchivos.loadImage (
                                    archivoSeleccionado) ;
        panelDeImagen.setImage(imagen) ;
        ventana.pack() ;
    }
}
```

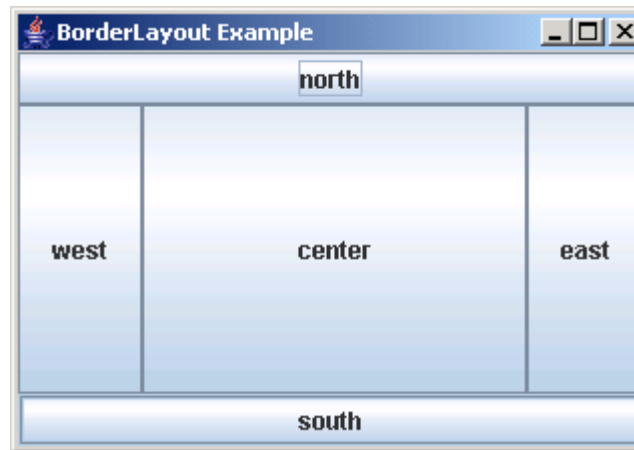
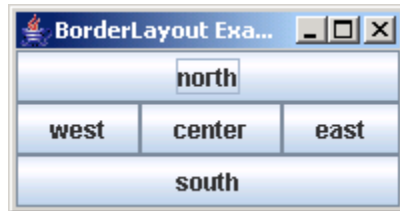
Gestores de Disposición

- Administran el espacio entre los diversos componentes.
 - `FlowLayout`, `BorderLayout`,
`GridLayout`, `BoxLayout`,
`GridBagLayout`.
- Administran los objetos `Container`, ej. un panel contenedor.
- Cada uno impone su propio estilo.

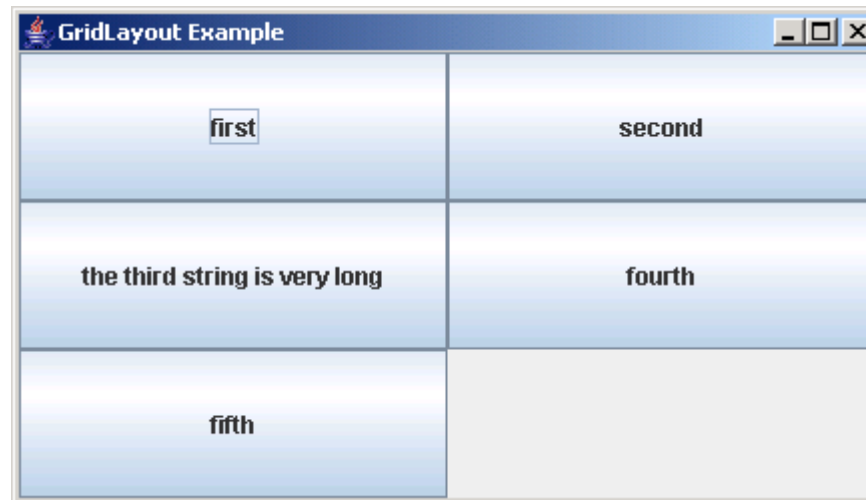
FlowLayout



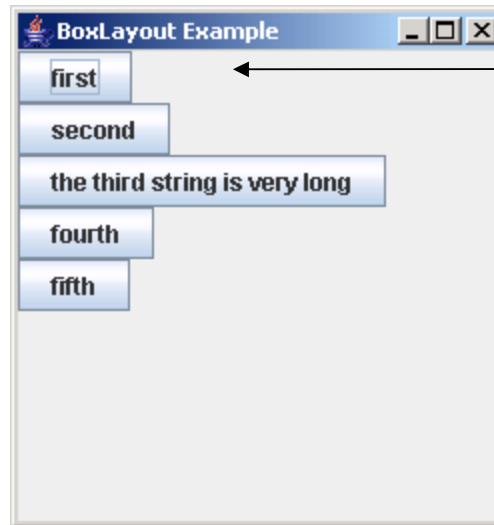
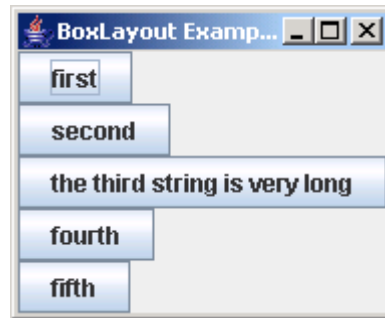
BorderLayout



GridLayout



BoxLayout



**Nota: no se cambia
el tamaño del
componente.**

Contenedores Anidados

- Se pueden obtener disposiciones sofisticadas anidando contenedores.
 - Use `JPanel` como un contenedor básico.
- Cada contenedor tendrá su propio gestor de disposición.
- Se prefiere su uso antes que **`GridBagLayout`**.

Strut y Glue

- Componentes invisibles para espaciamiento.
- Disponibles desde la clase **Box**.
- **Strut**: ancho fijo.
 - `Component createHorizontalStrut(int ancho)`
 - `Component createVerticalStrut(int alto)`
- **Glue**: rellena el espacio disponible.
 - `Component createHorizontalGlue()`
 - `Component createVerticalGlue()`

Diálogos

- Los diálogos modales bloquean cualquier otra interacción.
 - Fuerza una respuesta del usuario.
- Los diálogos no modales permiten otra interacción.
 - A veces puede ser deseable.
 - Puede dificultar el evitar inconsistencias.

Diálogos Estándar

JOptionPane

- Diálogo de mensaje
 - Mensaje de texto más un botón OK.
- Diálogo de confirmación
 - Opciones Sí, No, Cancelar.
- Diálogo de entrada
 - Mensaje de texto y un campo de entrada.
- Se puede tener variantes.

Diálogo de Mensaje

```
private void mostrarAcercaDe()  
{  
    JOptionPane.showMessageDialog(ventana,  
        "VisorDeImagen\n" + VERSION,  
        "Acerca de Visor De Imagen",  
        JOptionPane.INFORMATION_MESSAGE) ;  
}
```




Filtros de Imagen

- Funciones aplicadas a toda la imagen.

```
int alto = getAlto();  
int ancho = getAncho();  
for(int y = 0; y < alto; y++) {  
    for(int x = 0; x < ancho; x++) {  
        Color pixel = getPixel(x, y);  
        alterar el valor del color del pixel;  
        setPixel(x, y, pixel);  
    }  
}
```

Agregando Filtros Adicionales

```
private void aplicarClaro()
{
    if(imagenActual != null) {
        imagenActual.aclarar();
        ventana.repaint();
        showStatus("Filtro aplicado: Claro");
    }
    else {
        showStatus("No hay ninguna imagen
cargada.");
    }
}
```



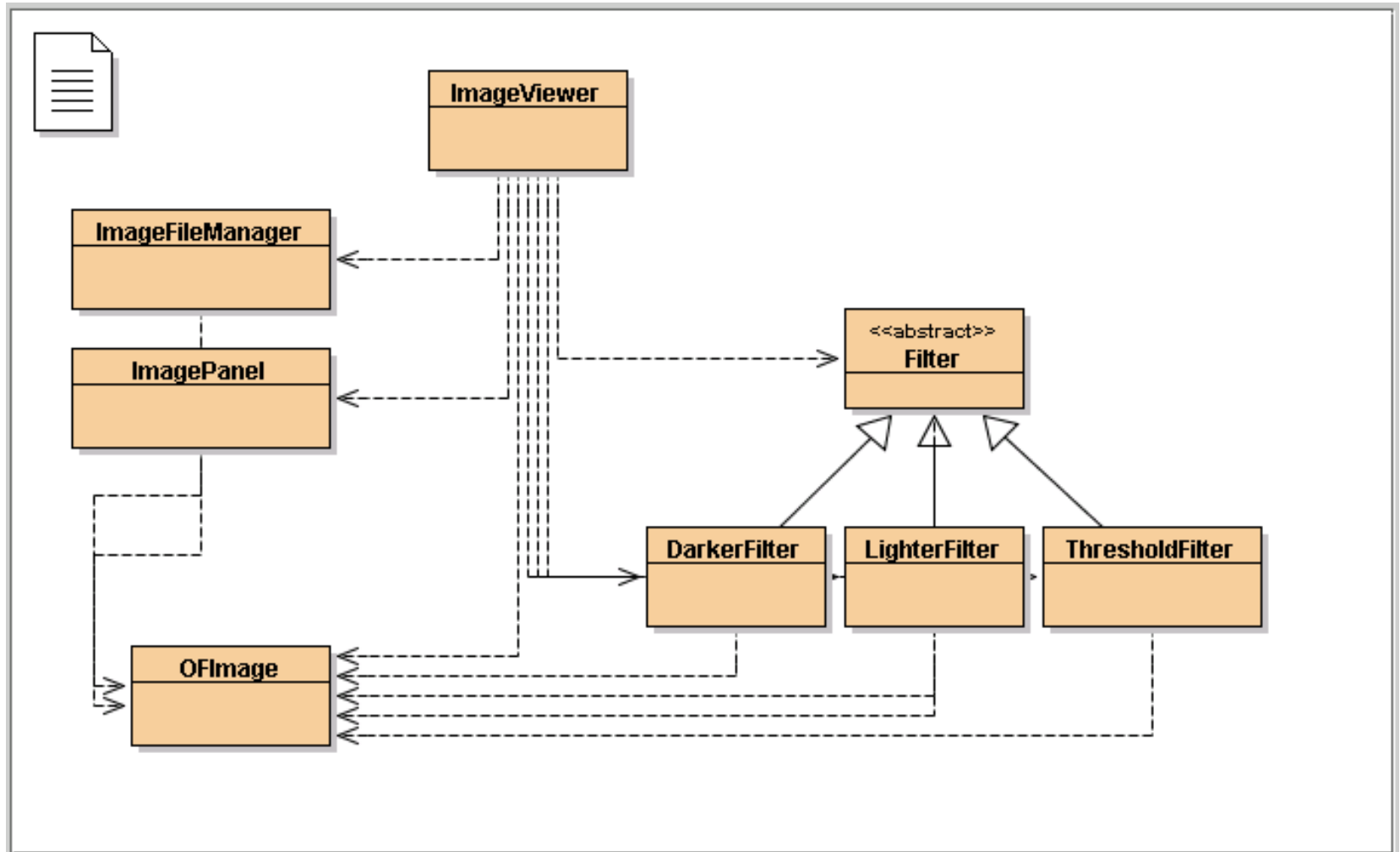
¿Duplicación de
Código?
¡Refactorizar!

```
private void aplicarUmbral()
{
    if(imagenActual != null) {
        imagenActual.umbral();
        ventana.repaint();
        showStatus("Filtro aplicado: Umbral");
    }
    else {
        showStatus("No hay ninguna imagen cargada.");
    }
}
```

Agregando Filtros Adicionales

- Definir una superclase **Filtro** (abstracta).
- Crear subclases para las funciones específicas.
- Crear una colección de instancias de subclases en **VisorDeImagen**.
- Definir un método genérico **aplicarFiltro**.
- Ver *visordeimagen2-0*.

visordeimagen2-0



Botones y Disposiciones Anidadas

Un GridLayout dentro de un FlowLayout dentro de un BorderLayout.



Bordes

- Se utilizan para decorar alrededor de los componentes.
- Definidos en `javax.swing.border`
 - `BevelBorder`, `CompoundBorder`,
`EmptyBorder`, `EtchedBorder`,
`TitledBorder`.

Agregando Espaciado

```
JPanel panelContenedor =  
    (JPanel) ventana.getContentPane();  
panelContenedor.setBorder(  
    new EmptyBorder(6, 6, 6, 6));  
  
// Especifica un gestor de disposición con un  
// espaciado agradable  
panelContenedor.setLayout(new BorderLayout(6, 6));  
  
panelDeImagen = new PanelDeImagen();  
panelDeImagen.setBorder(new EtchedBorder());  
panelContenedor.add(panelDeImagen,  
    BorderLayout.CENTER);
```

Otros Componentes

- Slider
- Spinner
- Tabbed pane
- Scroll pane

Repaso

- Se debe tratar de mantener cohesividad en las estructuras de la aplicación.
 - Esto se logra separando los elementos de la IGU de la funcionalidad de la aplicación.
- Los componentes predefinidos simplifican la creación de IGUs sofisticadas.
- Los gestores de disposición acomodan los componentes por yuxtaposición.
 - La anidación de contenedores nos permite tener un mejor control.

Repaso

- Algunos componentes reconocen la interacción de ellos con el usuario.
- Los componentes reactivos envían eventos a los oyentes de eventos.
- Las clases internas anónimas generalmente se utilizan para implementar los oyentes de eventos.