Estructura de las Aplicaciones Orientadas a Objetos

Interfaces gráficas de usuario

Programación Orientada a Objetos
Facultad de Informática

Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense Madrid

Basado en el curso Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Conceptos

- Interfaces gráficas de usuario (GUI: *Graphical User Interface*)
  - Componentes de las interfaces
  - Disposición (*layout*) de los elementos de la interfaz gráfica
  - Gestión de eventos
- Clases anidadas
  - Clases anidadas anónimas

Tipos de programas en Java

- Aplicaciones
  - Se pueden ejecutar directamente en un entorno Java
  - Tipos
    - Modo de consola
      - Interacción mediante teclado
      - Interfaz basado en texto
    - Aplicaciones con interfaz gráfica (GUI)
      - Ventanas graficas para entrada y salida de datos
      - Iconos
      - Dispositivos de entrada (p.ej. ratón, teclado)
      - Interacción directa
- Applets
  - Pequeñas aplicaciones que se ejecutan dentro de un navegador (o en el visualizador de applets - *Appletviewer*)
    - Interfaz gráfica
    - Limitaciones por motivos de seguridad

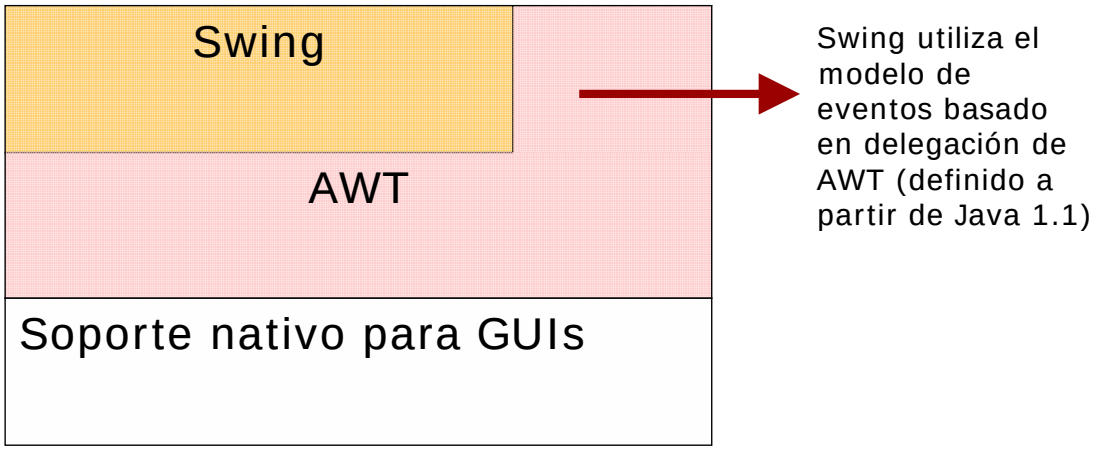
Elementos básicos de una GUI

- **Componentes** GUI (*widgets*)
  - Objetos visuales del interfaz
  - Un programa gráfico es un conjunto de componentes anidados
    - ventanas, contenedores, menús, barras, botones, campos de texto, etc.
- **Disposición** (*layout*): cómo se colocan los componentes para lograr un GUI cómodo de utilizar
  - ***Layout managers****:* Gestionan la organización de los componentes gráficos de la interfaz
- **Eventos**: interactividad, respuesta a la entrada del usuario
  - Desplazamiento del ratón, selección en un menú, botón pulsado, etc.
- Creación de gráficos y texto - Clase ***Graphics***
    - Define fuentes, pinta textos,
    - Para dibujo de líneas, figuras, coloreado,...

Bibliotecas de componentes para GUI

- Abstract Windowing Toolkit (**AWT**)
  - "Look & Feel" dependiente de la plataforma
    - La apariencia de ventanas, menúes, etc. es distinta en Windows, Mac, Motif, y otros sistemas
  - Funcionalidad independiente de la plataforma
  - Básico y experimental
  - Estándar hasta la versión JDK 1.1.5
- **Swing** / Java Foundation Classes (desde JDK 1.1.5)
  - *Look & Feel* y funcionalidad independiente de la plataforma
  - Desarrollado 100% en Java
  - Portable: si se elige un *look&feel* soportado por Swing (o se programa uno) puede asegurarse que la GUI se *verá* igual en cualquier plataforma
  - Mucho más completo que AWT

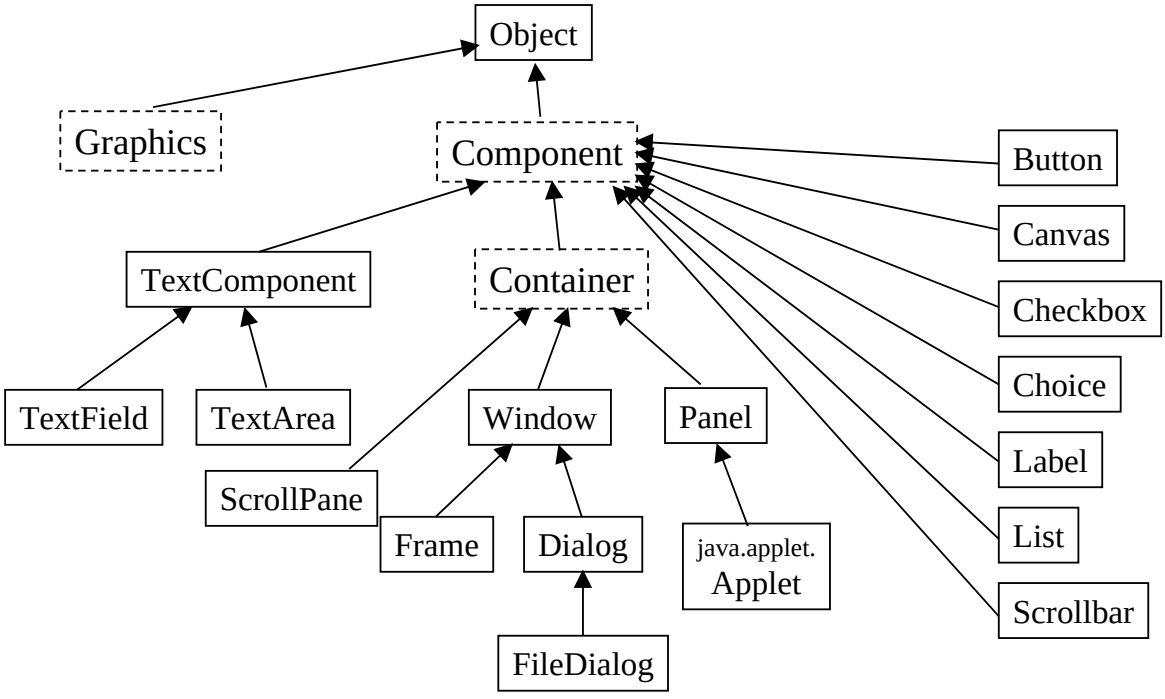
Soporte Java para el desarrollo de GUIs



Componentes del AWT

- Contenedores
  - Contienen otros componentes (u otros contenedores)
    - Estos componentes se tienen que añadir al contenedor y para ciertas operaciones se pueden tratar como un todo
    - Mediante un gestor de diseño controlan la disposición (*layout*) de estos componentes en la pantalla
    - Ejemplo: Panel, Frame, Applet
- Lienzo (clase *Canvas*)
  - Superficie simple de dibujo
- Componentes de interfaz de usuario
  - botones, listas, menús, casillas de verificación, campos de texto, etc.
- Componentes de construcción de ventanas
  - ventanas, marcos, barras de menús, cuadros de diálogo

Jerarquía de componentes del AWT



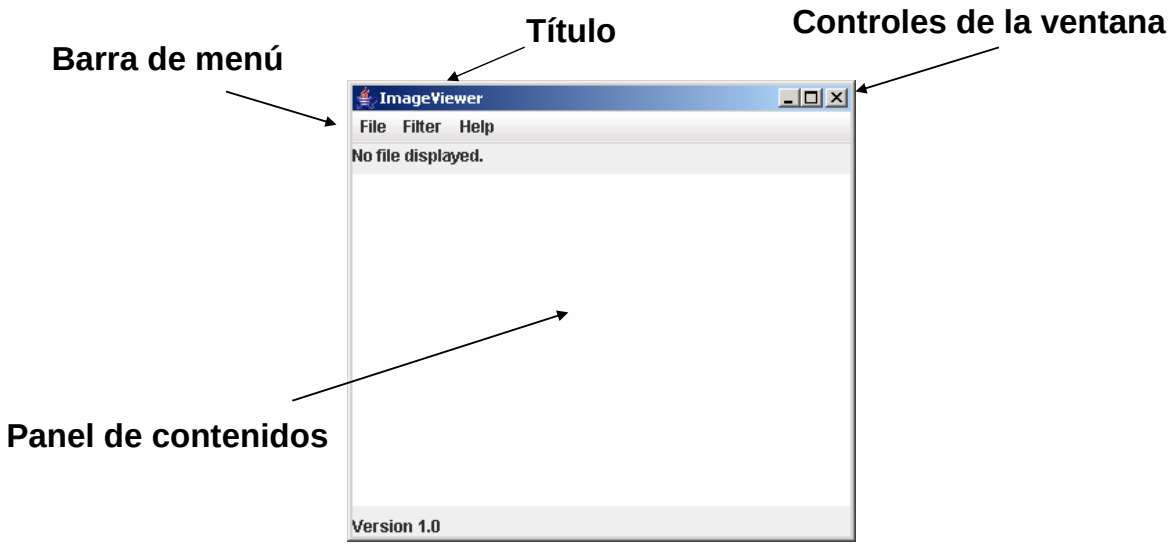
Contenedores

- Panel
  - Sirve para colocar botones, etiquetas, etc.
  - No existe sin una ventana que lo albergue
  - Un *applet* es un panel
- Window
  - Sirve para crear nuevas ventanas independientes
  - Ventanas gestionadas por el administrador de ventanas de la plataforma (Windows, Motif, Mac, etc.)
  - Normalmente se usan dos tipos de ventanas:
    - **Frame**: ventana donde se pueden colocar menús
    - **Dialog**: ventana para comunicarse con el usuario
      - Se usan para colocar botones, etiquetas, etc.
      - Cumple la misma función que un panel, pero en una ventana independiente

Creación de una ventana

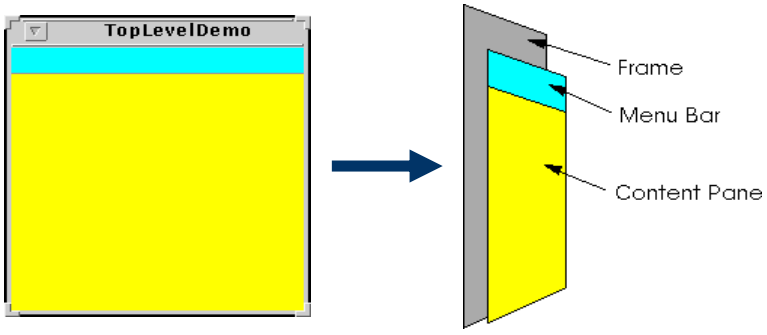
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ImageViewer
{
    private JFrame frame;

    /**
     * Create an ImageViewer show it on screen.
     */
    public ImageViewer()
    {
            makeFrame();
    }

    // rest of class omitted.
}
```

Elementos de una ventana



La clase javax.swing.**JFrame**

- Toda aplicación Swing tiene, al menos, un contenedor raíz (una ventana)
- La clase JFrame proporciona ventanas al uso (aunque puede haber otro tipo de ventanas)
- A su vez, JFrame incluye una serie de elementos:



- **Los contenidos se añaden en el *panel de contenidos* (*content pane*) accesible a través del método** getContentPane **(por defecto, un objeto de tipo** Jpane, **aunque puede cambiarse con** setContentPane**).**
- **La barra de menú puede fijarse con** setJMenuBar

Crea una ventana con un texto

```
/**
 * Create the Swing frame and its content.
 */
private void makeFrame()
{
    frame = new JFrame("ImageViewer");
    Container contentPane = frame.getContentPane();

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

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

Componentes: Etiquetas, Botones, Campos, ...

- Botón - Clase *JButton*
  - Botón de interacción que puede tener una etiqueta
- Etiqueta - Clase *JLabel*
  - Muestra una cadena de sólo lectura
    - Normalmente para asociar el texto con otro componente
- Campo de texto - Clase *JTextField*
  - Campo de una línea que permite introducir y editar texto
- Area de texto - Clase *JTextArea*
  - Campo de texto de varias líneas
  - Mayor funcionalidad
    - Añadir, reemplazar e insertar texto
    - Barras de desplazamiento
- Menús - Clase *JMenuBar, JMenu, y JMenuItem*
  - Para definir una barra de menús, cada menú y los elementos de cada menú

Uso de componentes

1) Crear el componente
   - usando new:

     JButton b = new JButton(“Correcto”);

2) Añadirlo al contenedor
   - usando add:

     contentPane.add(b); // añadir en el contenedor contentPane
     // crear y añadir el componente en una sóla operación:
     contentPane.add(new Label(“Etiqueta 1”));
     contentPane.add(new Label(“Etiqueta 2”));
   - Si luego se quiere quitar, usar *remove*(componente)

3) Invocar métodos sobre el componente y manejar eventos

   System.out.println(b.getLabel());
   b.setLabel(“etiqueta modificada”);

Creación de una barra de menús

```
private void makeMenuBar(JFrame frame) {
    JMenuBar menubar = new JMenuBar();
    frame.setJMenuBar(menubar);

    // create the File menu
    JMenu fileMenu = new JMenu("File");
    menubar.add(fileMenu); // se añade a la barra de menús

    JMenuItem openItem = new JMenuItem("Open");
    fileMenu.add(openItem);  // se añade al menú File

    JMenuItem quitItem = new JMenuItem("Quit");
    fileMenu.add(quitItem);
}
```

Tratamiento de eventos

- Los eventos se corresponden a las interacciones del usuario con los componentes
- Los componentes están asociados a distintos tipos de eventos
  - Las ventanas (p.ej. JFrame) están asociadas con *WindowEvent*
  - Los menús están asociados con *ActionEvent*
- Se pueden definir objetos que saben cómo tratar los eventos
  - Estos objetos serán notificados de la ocurrencia de un evento
  - Por eso se llaman ***listeners***
  - Tienen definidos métodos que se llaman cuando ocurre el evento asociado

Tratamiento de eventos

- Dos categorías de eventos:
  - Eventos de bajo nivel
    - Están relacionados con la interacción física con la interfaz (por ejemplo, ¿qué botón del ratón se ha pulsado?)
    - Ejemplos: MouseInputEvent, WindowEvent y KeyEvent
  - Eventos de alto nivel, o semánticos
    - Representan operaciones lógicas realizadas sobre los elementos (por ejemplo, se ha pulsado el botón “Salir” en la interfaz)
    - ActionEvent

Tratamiento de eventos

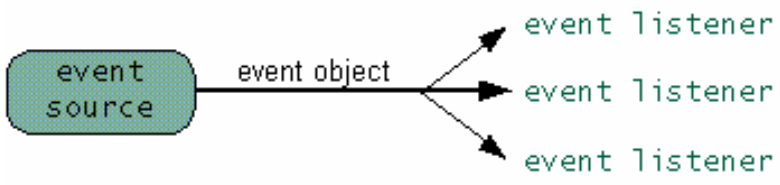
- El objeto listener debe ser de una clase que herede la interfaz correspondiente al tipo de evento que va a tratar

```
class ImageViewer implements ActionListener {
   public void actionPerformed(ActionEvent e) { … }
}
```

- El objeto listener se añade a una lista de listeners asociados al componente de GUI

```
item.addActionListener(this);
```

- Cuando se produce un evento, se notifica a todos los objetos listeners asociados



Eventos de acción: ActionEvent

- Indica que se ha producido un evento sobre un componente de la interfaz
  - JButton, JList, JTextField, JMenuItem, etc.
- El método que se invoca en los listeners está definido en la interfaz ActionListener

```
public interface ActionListener extends EventListener {
   void actionPerformed(ActionEvent e) ;
}
```

  - Luego la clase del objeto oyente debe implementar el método *actionPerformed*
- Los componentes tienen métodos para poder añadir o quitar objetos oyentes
  - **addActionListener**(ActionListener l)
  - **removeActionListener**(ActionListener l)

Ejemplo: ActionListener para ImageViewer

```
public class ImageViewer implements ActionListener
{
    …
    public void actionPerformed(ActionEvent e)
    {
        String command = e.getActionCommand();
        if(command.equals("Open")) {
            …
        }
        else if (command.equals("Quit")) {
            …
        }
        …
    }
    …
    private void makeMenuBar(Jframe frame)
    {
        …
        openItem.addActionListener(this);
        …
    }
}
```


Problemas con esta solución

- El ejemplo muestra un tratamiento centralizado de los eventos en la propia clase principal
- Aunque funciona…
- Es mejor definir clases específicas por cada evento
  - Mayor escalabilidad
  - Evitar identificar los componentes por el texto
- Veamos la alternativa, usando clases anidadas

Clases anidadas

- Las definiciones de clase se pueden anidar

```
public class Contenedora
{
    …
    private class Anidada
    {
        …
    }
}
```


Clases anidadas

- Las instancias de las clases internas estarán dentro de la clase contenedora
- Las instancias de la clase interna tienen acceso a la parte privada de la clase contenedora
  - Esto es práctico porque el tratamiento de un evento requiere normalmente acceso al estado de la aplicación

Clases anidadas anónimas

- Obedecen las reglas de las clases internas
- Se usan para crear objetos específicos en un momento, para los que no es necesario dar un nombre de clase
- Tienen una sintaxis especial
- La instancia es siempre referenciada por su supertipo, ya que no tiene nombre el subtipo

ActionListener anónimo

```
JMenuItem openItem = new JMenuItem("Open");

openItem.addActionListener(
    new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            openFile();
        }
    }
);
```

Anonymous object creation

Class definition

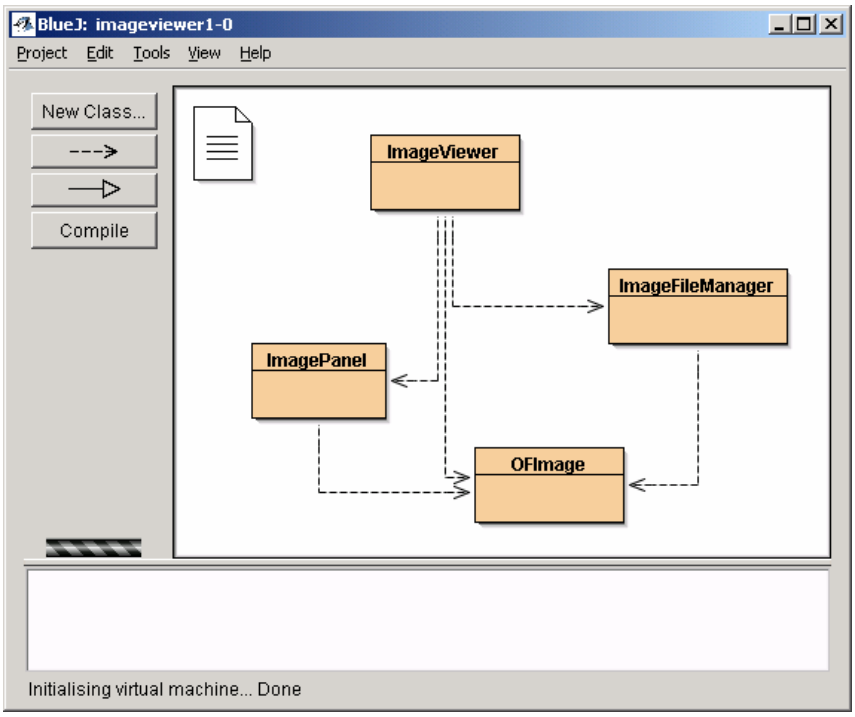
Actual parameter

Tratamiento del cierre de ventana

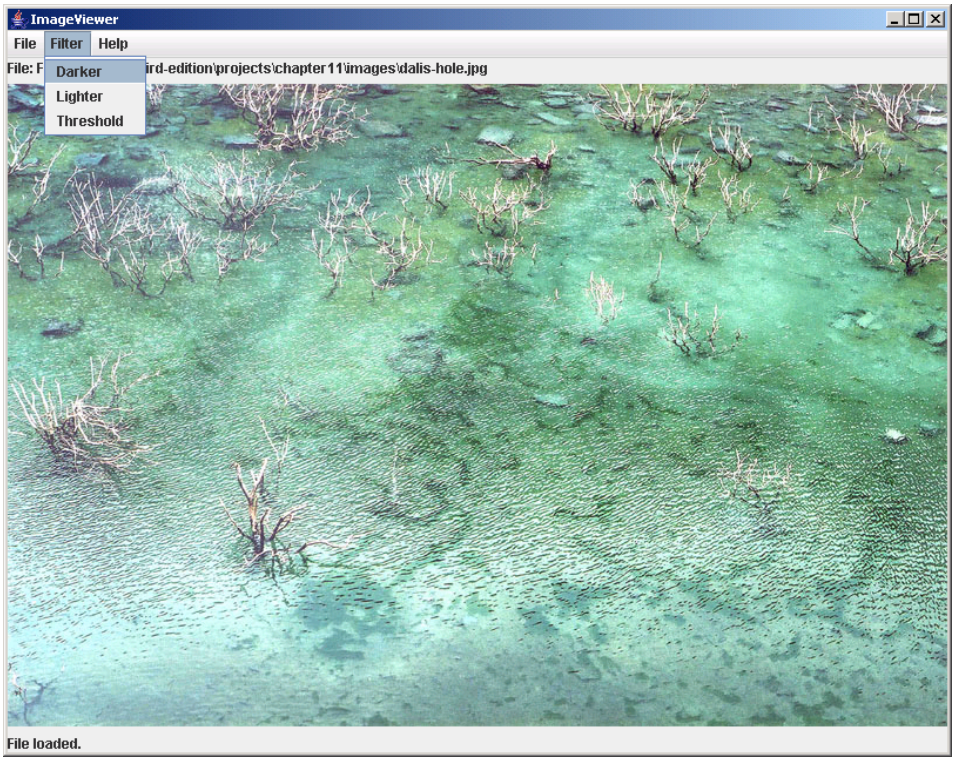
```
frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e)
        {
            System.exit(0);
        }
    });
```

- WindowAdapter es una implementación de la interfaz WindowListener donde todas las operaciones están declaradas como vacías

El proyecto ImageViewer



Procesamiento de imágenes



Responsabilidades de las clases

- `ImageViewer`
  - Se encarga de la estructura de la GUI
- `ImageFileManager`
  - Métodos estáticos para cargar y salvar las imágenes en ficheros
- `ImagePanel`
  - Visualiza la imagen dentro de la GUI
- `OFImage`
  - Modela una imagen en 2D

La clase OFImage

- Subclase de `java.awt.image.BufferedImage`
  - Representa una imagen con un buffer para acceder a los datos de la imagen
- Representa un array 2D de pixels
- Métodos importantes:
  - `getPixel`, `setPixel`
  - `getWidth`, `getHeight`
- Cada pixel tiene un color:
  - `java.awt.Color`

Añadir un panel para visualizar imágenes

```
public class ImageViewer
{
    private JFrame frame;
    private ImagePanel imagePanel;

    …

    private void makeFrame()
    {
        Container contentPane = frame.getContentPane();
        imagePanel = new ImagePanel();
        contentPane.add(imagePanel);
    }

    …
}
```

Cargar una imagen

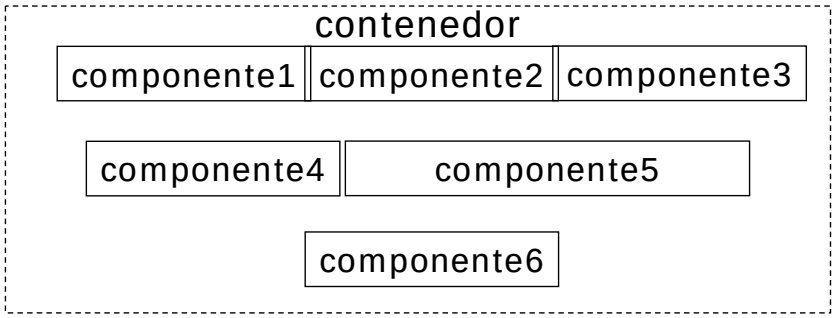
```
public class ImageViewer
{
    private JFrame frame;
    private ImagePanel imagePanel;

    …

    private void openFile()
    {
        File selectedFile = …;
        OFImage image =
            ImageFileManager.loadImage(selectedFile);
        imagePanel.setImage(image);
        frame.pack();
    }

    …
}
```


Disposición de los componentes (*layout manager*)

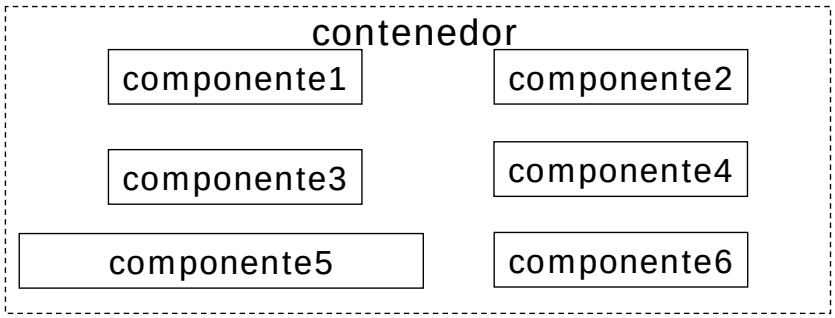
- Cómo se colocan los componentes (usando el método *add*) depende del gestor de disposición del contenedor (*layout manager*)
- Tipos de disposiciones:
  - FlowLayout
    - Los componentes se ponen de izquierda a derecha hasta llenar la línea, y se pasa a la siguiente. Cada línea se centra
      - Por defecto, en paneles y applets
  - BorderLayout
    - Se ponen los componentes en un lateral o en el centro
    - se indica con una dirección:“East”, “West”, “North”, “South”, “Center”
      - Por defecto, en ventanas JFrame
  - GridLayout
    - Se colocan los componentes en una rejilla rectangular (filas x cols)
    - Se añaden en orden izquierda-derecha y arriba-abajo
- Para poner una disposición se utiliza el método *setLayout():*

```
GridLayout  nuevolayout = new GridLayout(3,2);
setLayout(nuevolayout);
```

Disposición de los componentes (*layout manager*)



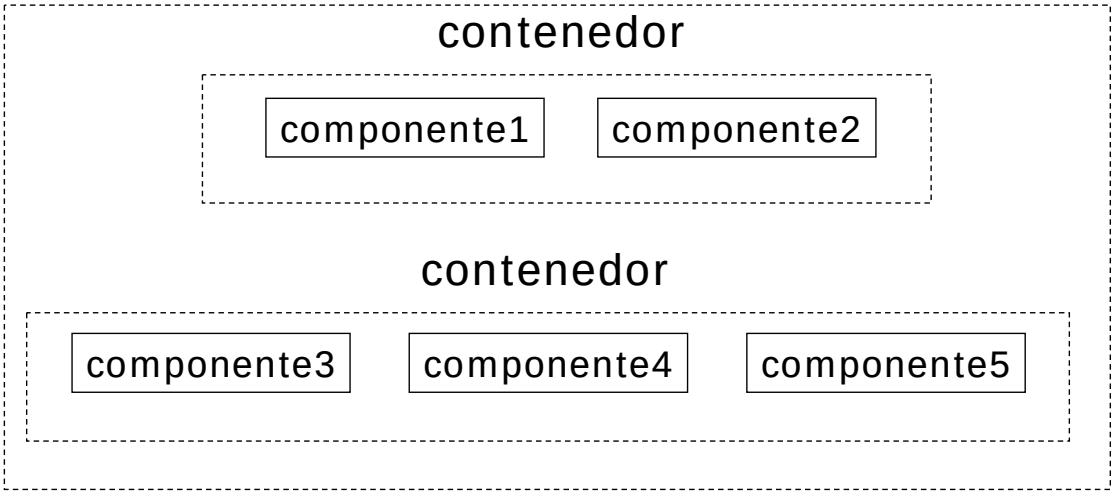
FlowLayout



GridLayout(3,2)

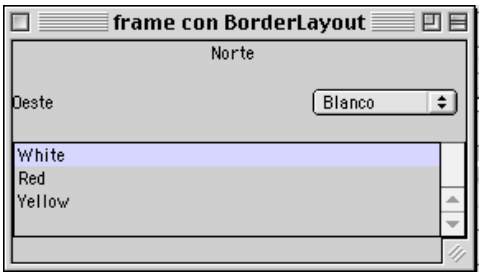
Contenedores anidados

- Para conseguir diseños sofisticados lo mejor es anidar contenedores
  - Se puede utilizar JPanel como contenedor básico
  - Cada contenedor tendrá su layout manager específico

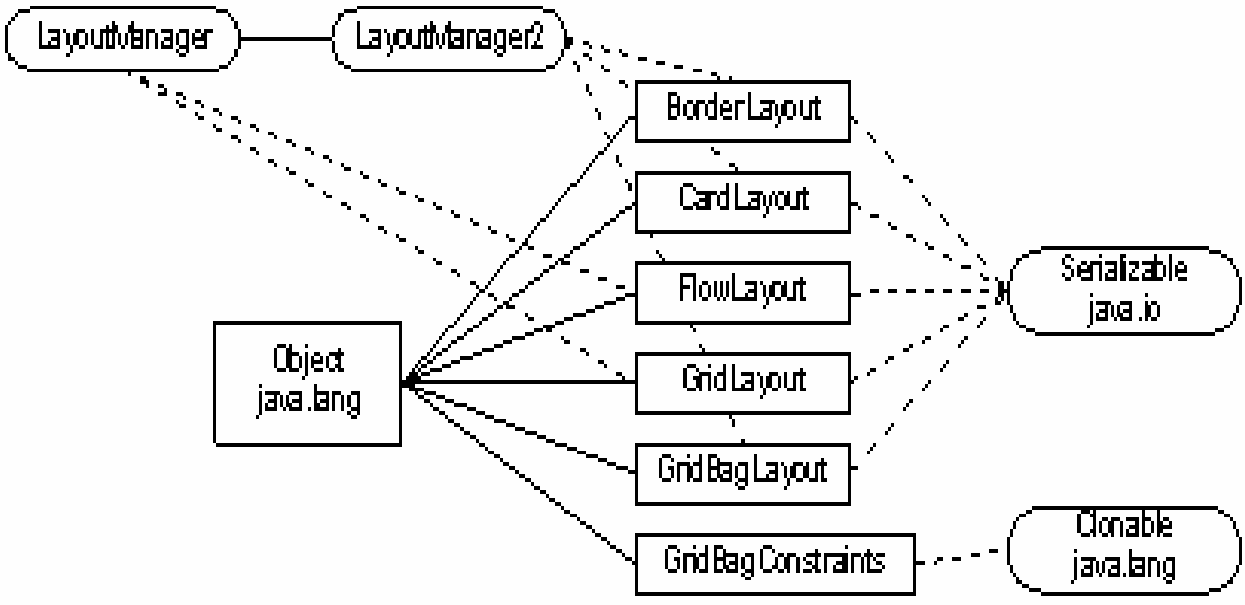


Disposición de los componentes (*layout manager*)

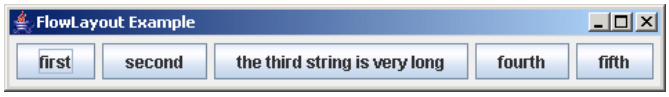
- GridBagLayout
  - Similar al GridLayout pero mas versátil
  - Presenta los componentes en una rejilla, pero:
    - Un componente puede ocupar más de una fila y más de una columna
    - Las filas y las columnas pueden tener tamaños diferentes
    - No se tiene que rellenar en un orden predeterminado
  - Utiliza *GridBagConstraints* para especificar cómo deben colocarse, distribuirse, alinearse, etc., los componentes

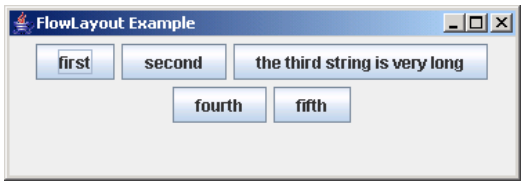


Administradores de diseño

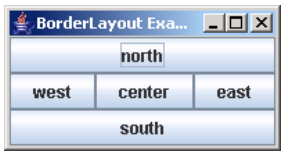


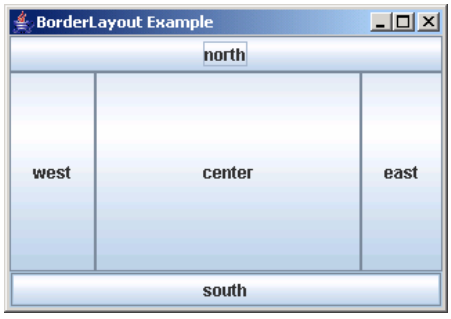
FlowLayout



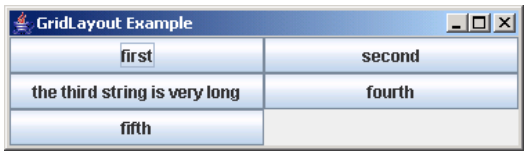


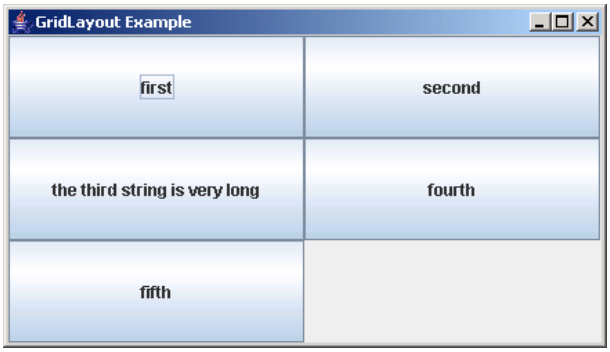
BorderLayout



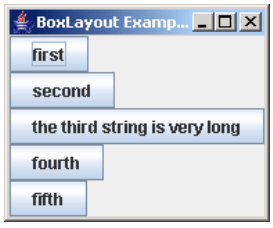


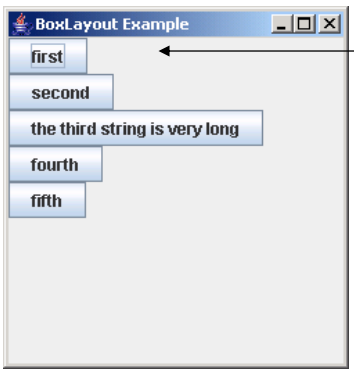
GridLayout





BoxLayout





Note: no component resizing.

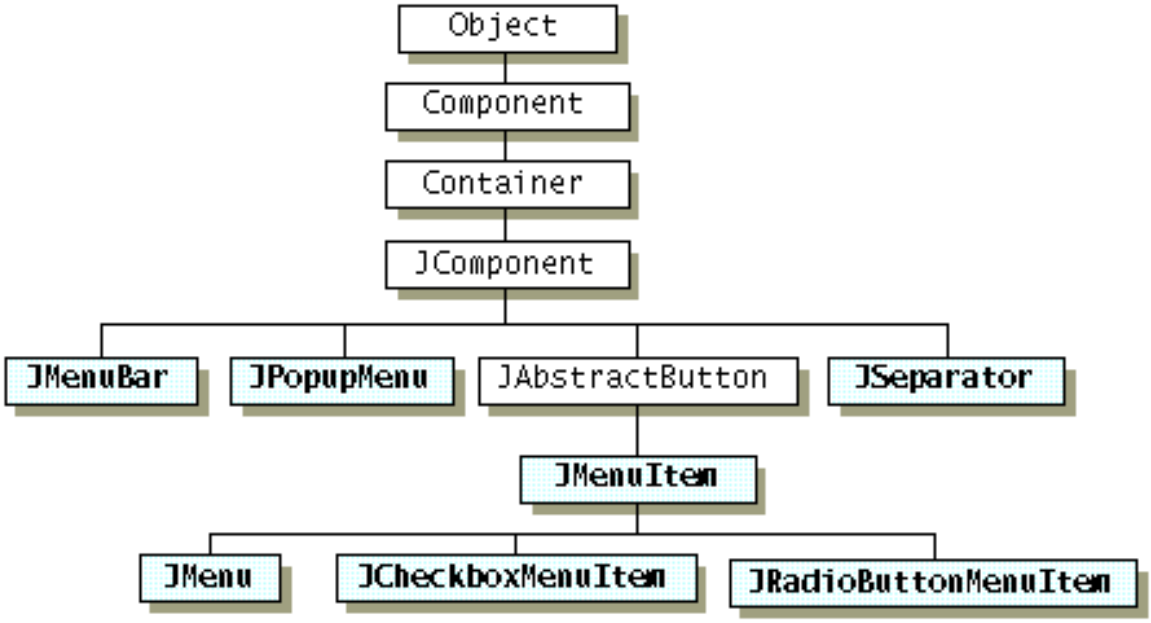
Botones

- Los botones, junto con los menús, son los controles más típicos en una GUI
- Existen diferentes tipos (especializaciones de AbstractButton)
  - **JButton**: Botón aislado. Puede pulsarse, pero su estado no cambia
  - **JToggleButton** : Botón seleccionable. Cuando se pulsa el botón, su estado pasa a *seleccionado*, hasta que se pulsa de nuevo (entonces se deselecciona)
    - *isSelected()* permite chequear su estado
  - **JCheckBox** : Especialización de JToggleButton que implementa una casilla de verificación. Botón con estado interno, que cambia de apariencia de forma adecuada según si está o no está seleccionado
  - **JRadioButton**: Especialización de JToggleButton que tiene sentido dentro de un mismo grupo de botones (*ButtonGroup*) que controla que sólamente uno de ellos está seleccionado
    - Nota: ButtonGroup es únicamente un controlador, no un componente)
- El evento semántico más común anunciado por los botones es ActionEvent

Botones


```
Box caja = Box.createHorizontalBox();
caja.add(new JButton("Un botón normal"));
caja.add(new JToggleButton("Un botón seleccionable"));
caja.add(new JToggleButton("Otro botón seleccionable",true));
caja.add(new JCheckBox("Cine"));
caja.add(new JCheckBox("Teatro"));
caja.add(new JCheckBox("Música"));
ButtonGroup grupo = new ButtonGroup();
JRadioButton r1 = new JRadioButton("Hombre");
JRadioButton r2 = new JRadioButton("Mujer");
JRadioButton r3 = new JRadioButton("Asexuado");
grupo.add(r1);
grupo.add(r2);
grupo.add(r3);
caja.add(r1);
caja.add(r2);
caja.add(r3);
```

Menús

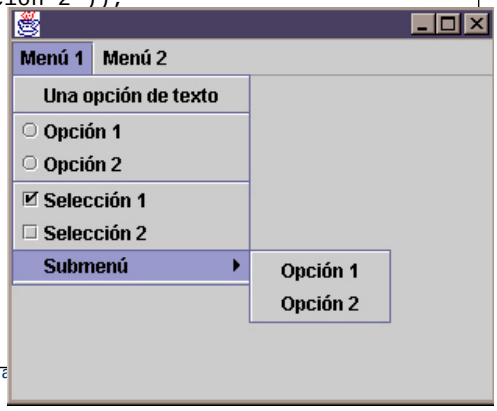


Menús

- La creación de una barra de menús básica supone:
  - Crear un objeto de tipo JMenuBar.
  - Para cada entrada, crear un objeto de tipo JMenu.
  - Incluir objetos de tipo JMenuItem en el menú. Esto puede incluir menús anidados.
  - Asociar a los *items* acciones apropiadas (notifican eventos semánticos de tipo ActionEvent, ya que, en realidad, especializan a AbstractButton).
- Con setJMenuBar es posible añadir una barra de menús a una ventana (JFrame).
- Importante: En una GUI, muchas veces existen controles ligados a la misma acción (eg. un botón que hace lo mismo que un *item* de un menú). En este caso ambos controles pueden compartir el mismo oyente (y es aconsejable hacerlo así).
- Más importante aún: El diseño de una barra de menús debe ser consistente (poner opciones semánticamente relacionadas juntas).

Menús

```
public class PruebaMenu extends JFrame {
   public void ejecuta() {
            JMenuBar barra = new JMenuBar();
            JMenu menu1 = new JMenu("Menú 1");
            menu1.add(new JMenuItem("Una opción de texto"));
            menu1.add(new JSeparator());
            ButtonGroup grupo = new ButtonGroup();
            JRadioButtonMenuItem r1 = new JRadioButtonMenuItem("Opción 1");
            JRadioButtonMenuItem r2 = new JRadioButtonMenuItem("Opción 2");
            grupo.add(r1);
            grupo.add(r2);
            menu1.add(r1);
            menu1.add(r2);
            menu1.add(new JSeparator());
            menu1.add(new JCheckBoxMenuItem("Selección 1",true));
            menu1.add(new JCheckBoxMenuItem("Selección 2"));
            JMenu menu11 = new JMenu("Submenú");
            menu11.add(new JMenuItem("Opción 1"));
            menu11.add(new JMenuItem("Opción 2"));
            menu1.add(menu11);
            barra.add(menu1);
            barra.add(new JMenu("Menú 2"));
            setJMenuBar(barra);
            pack();
            setVisible(true);
          }
        public static void main(String[] args) {
                    new PruebaMenu().ejecuta();
          }}
```



Interacción modal

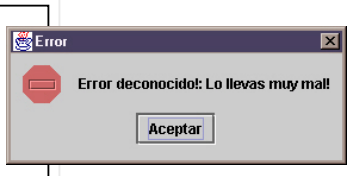
- Muchas veces surge la necesidad de lanzar, esporádicamente, ventanas secundarias para indicar algún hecho, o pedir algún dato al usuario: cuadros de diálogo
- La mayor parte de estas interacciones suelen ser modales (es decir, la ejecución del programa se interrumpe hasta que el usuario cierra el cuadro de diálogo)
- Swing ofrece la posibilidad de crear diálogos a medida…
- …pero (afortunadamente), también ofrece la posibilidad de crear/configurar diálogos prefabricados que son útiles en muchas situaciones
- ¿Cómo funciona la interacción modal?: Swing lanza un hilo adicional para manejar dichas interacciones (y el resto de los eventos rutinarios). El hilo de tratamiento de eventos se sincroniza (espera) la finalización de dicho hilo de interacción modal
- Importante: Un abuso de la interacción modal conduce a interfaces poco usables (en el límite, a interacciones tipo consola)

Interacción modal

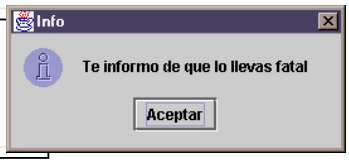
- La clase JOptionPane ofrece un conjunto de métodos estáticos que lanzan diferentes tipos de diálogos (estos métodos están sobrecargados para poder crear diálogos con diferentes grados de fineza):
  - showMessageDialog : Muestra un diálogo de mensaje
  - showConfirmDialog : Muestra un diálogo de confirmación. Permite determinar la opción elegida por el usuario de forma modal (*yes*, *no*, *cancel*)
  - showInputDialog: Muestra un diálogo en el que se solicita, de forma modal, una entrada al usuario. Dependiendo de la versión de método utilizada, el usuario puede teclear la entrada, o bien seleccionarla de una lista de entradas disponibles
  - showOptionDialog: Muestra un diálogo que puede personalizarse con botones (en general, con componentes) a medida

Interacción modal

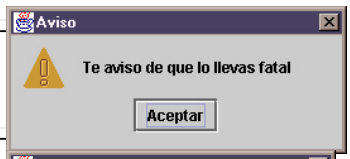
```
JOptionPane.showMessageDialog(this, // La ventana padre.
   "Error deconocido!: Lo llevas muy mal!", //El mensaje.
   "Error",  // El título de la ventana de diálogo.
   JOptionPane.ERROR_MESSAGE // El tipo de mensaje
);
```



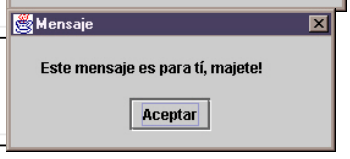

```
JOptionPane.showMessageDialog(this,
"Te informo de que lo llevas fatal", "Info",
JOptionPane.INFORMATION_MESSAGE);
```




```
JOptionPane.showMessageDialog(this,
 "Te aviso de que lo llevas fatal", "Aviso",
 JOptionPane.WARNING_MESSAGE);
```

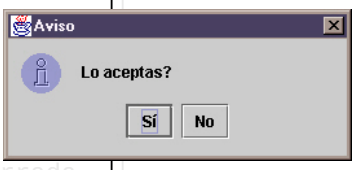



```
JOptionPane.showMessageDialog(this,
  Este mensaje es para tí, majete!", "Mensaje",
JOptionPane.PLAIN_MESSAGE);
```

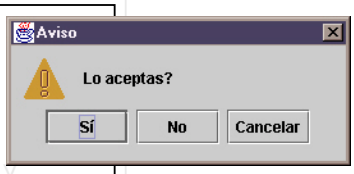


Interacción modal

```
int seleccionada =
  JOptionPane.showConfirmDialog(this,
  "Lo aceptas?", "Aviso",
  JOptionPane.YES_NO_OPTION, // Configuración del mensaje
  JOptionPane.INFORMATION_MESSAGE);
switch(seleccionada) {
  case JOptionPane.YES_OPTION: ... // tratar SI
  case JOptionPane.NO_OPTION: .. // tratar NO
  case JOptionPane.CLOSED_OPTION: .. // tratar ventana cerrada
  default: ... // esta opción nunca debería alcanzarse
}
```

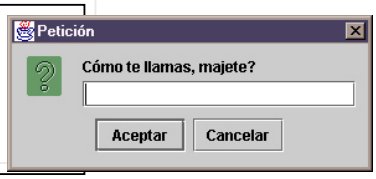



```
int seleccionada =
    JOptionPane.showConfirmDialog(this,
       "Lo aceptas?","Aviso",
       JOptionPane.YES_NO_CANCEL_OPTION,
       JOptionPane.WARNING_MESSAGE);
... // los posibles valores devueltos son los anteriores y
... // JOptionPane.CANCEL_OPTION
```

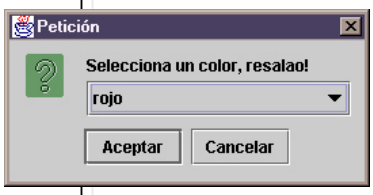


Interacción modal

```
String nombre = JOptionPane.showInputDialog(this,
  "Cómo te llamas, majete?",
   "Petición", JOptionPane.QUESTION_MESSAGE
);
// ... procesar entrada
```

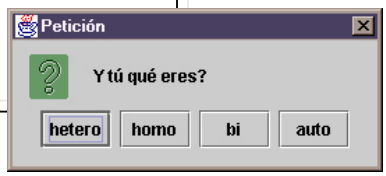



```
String[] colores = {"rojo","negro","amarillo",
                    "azul","majenta"};
Object opcion =
  JOptionPane.showInputDialog(this,
  "Selecciona un color, resalao!", "Petición",
  JOptionPane.QUESTION_MESSAGE,
  null, //icono. no hay.
  colores, // opciones. Se le podría pasar un array de
           // objetos arbitrarios
  colores[0] //opción inicial
);
 // ...procesar opción...
```



Interacción modal

```
String[] sexo = {"hetero","homo","bi","auto"};
int opcion =
  JOptionPane.showOptionDialog(this,
   "Y tú qué eres?", "Petición",
   -1, // opciones->quedan fijadas por el array de opciones
   JOptionPane.QUESTION_MESSAGE,
   null,//icono. no hay.
   sexo, // array de opciones
   sexo[0] // opcion inicial
);
... // procesamiento de la opcion ...
```



Interacción modal: selección de archivos

- Los diálogos de tipo JFileChooser facilitan esta tarea
- La forma habitual de trabajar con JFileChooser es:
  - Crear una instancia de JFileChooser.
  - Configurar dicha instancia de forma adecuada. Dicha configuración, normalmente, se limita a poner *filtros* sobre lo que puede seleccionarse con el selector de ficheros, así como a fijar la carpeta actual (con setCurrentDirectory):
    - Con setFileSelectionMode.
    - Con setFileFilter : Aquí debe añadirse una instancia de una subclase adecuada de FileFilter.
  - Mostrar un selector de ficheros utilizando alguno de los siguientes métodos:
    - showDialog : Selector *a medida*.
    - showOpenDialog : Selector para *abrir* un fichero.
    - showSaveDialog : Selector para *salvar* un fichero.
  - La interacción es modal: los métodos show devuelven el estado de la operación. Si todo ha ido bien, el método getSelectedFile devuelve un objeto de tipo File, que debe ser tratado de forma adecuada

Interacción modal: selección de archivos

- Creación y configuración de un selector

```
selector = new JFileChooser();
selector.setFileFilter(new FiltroTexto());
selector.setFileSelectionMode(JFileChooser.FILES_ONLY);
selector.setCurrentDirectory(new File(System.getProperty("user.dir")));
```

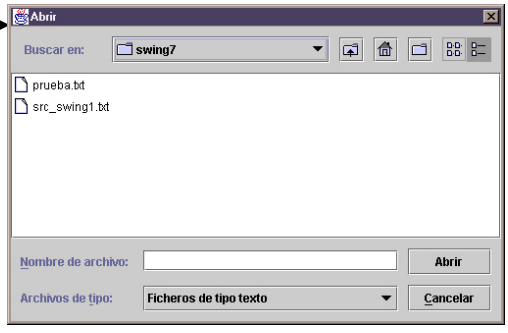
- Un filtro de ficheros

```
class FiltroTexto extends FileFilter {
    public boolean accept(File f) {
            String nombre = f.getName();
            return nombre.substring(Math.max(nombre.length()-4,0)).equals(".txt");
    }
    public String getDescription() {
            return "Ficheros de tipo texto";
     }
}
```

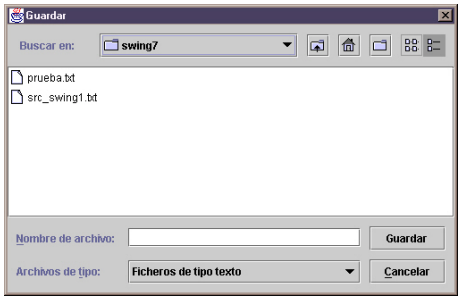

Interacción modal: selección de archivos

- El mismo selector permite crear diferentes diálogos de selección

`selector.showOpenDialog(this)` →



`selector.showSaveDialog(this)` →



Interacción modal: selección de archivos

Uso habitual de un diálogo creado a través de un FileChooser

```
if ( selector.showOpenDialog(this) ==  JFileChooser.APPROVE_OPTION )
{
        File f = selector.getSelectedFile();
        // Hacer lo que sea pertinente con el fichero ...
}
```


Otros contenedores Swing

- JPanel. Un contenedor intermedio genérico
  - Se utiliza para poner otros componentes
  - FlowLayout por defecto (los componentes se colocan uno detrás de otro)
- JScrollPane. Un panel con barras de scroll
- JTabbedPane. Un panel con diferentes vistas

- JToolBar. Una barra de herramientas
  - Puede cambiarse de situación por los bordes de su contenedor e incluso llevarse fuera
  - Las herramientas suelen ser (aunque no obligatoriamente) botones

Otros componentes atómicos Swing

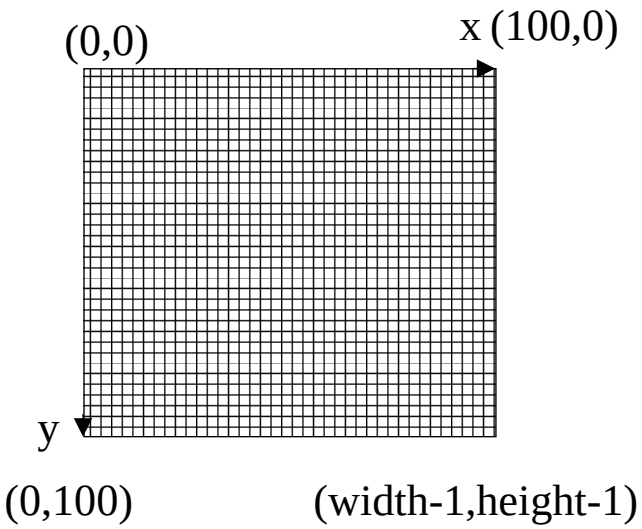
- JLabel. Etiquetas
- JTextField. Entrada de una línea de texto
- JTextArea. Entradas de varias líneas de texto
- JSlider. Una barra de selección en una escala
- JList. Una lista de elementos seleccionables
- JComboBox. Una lista de elementos desplegable
- JPopupMenu. Un menú desplegable

La clase *Graphics*

- Clase abstracta que es la base para los contextos gráficos que permiten a una aplicación dibujar los componentes independientemente del dispositivo de salida
- Un contexto gráfico es un objeto que funciona junto con las ventanas para mostrar los objetos gráficos
- Habitualmente no hay que crear ningún contexto gráfico ya que esto es parte del *framework* de AWT
  - Se obtiene mediante el método getGraphics()
- Mediante el método *paint(Graphics contexto)* se determina que es lo que se debe mostrar en dicho contexto

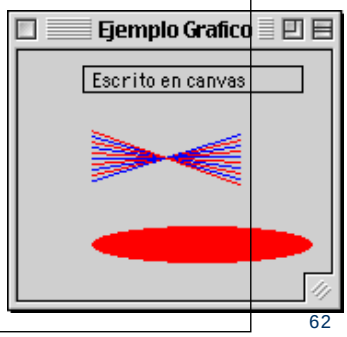
La clase *Graphics*

- Proporciona métodos para dibujar, rellenar, pintar imágenes, copiar áreas y pegar gráficos en pantalla
  - drawLine
  - drawRect y fillRect
  - drawPolygon
  - drawPolyline
  - drawOval y fillOval
  - drawArc y fillArc
- y para escribir texto
  - drawString
  - setFont



Ejemplo gráfico con Canvas

```
public class EjemploCanvas extends Canvas {
// se puede añadir a un frame para visualizarlo
        String cad = "Escrito en canvas";
// este metodo se ejecuta automaticamente cuando Java necesita mostrar la ventana
        public void paint(Graphics g) {
                // obtener el color original
                Color colorOriginal = g.getColor();
                // escribir texto grafico en la ventana y recuadrarlo
                g.drawString(cad, 40, 20);
                g.drawRect(35, 8, (cad.length()*7), 14);
                // dibujo de algunas lineas
                for (int i=20; i< 50; i= i+3) {
                        if ((i % 2) == 0)     g.setColor(Color.blue);
                        else          g.setColor(Color.red);
                        g.drawLine(40, (90-i), 120, 25+i);
                }
                // dibujo y relleno de un óvalo
                g.drawOval(40, 95, 120, 20);
                g.fillOval(40, 95, 120, 20);
                g.setColor(colorOriginal);
        }
}
```



Ejemplo gráfico con un applet

```
import java.awt.*;

public class Lámpara extends java.applet.Applet {

  public void paint(Graphics g) {
    g.fillRect(0,250,290,290);  // la mesa donde se pone la lámpara

    g.drawLine(125,250,125,160); // la base de la lámpara
    g.drawLine(175,250,175,160);

    g.drawArc(85,157,130,50,-65,312); // parte superior e inferior
    g.drawArc(85,87,130,50,62,58);

    g.drawLine(85,177,119,89);  // laterales
    g.drawLine(215,177,181,89);

    g.fillArc(78,120,40,40,63,-174);   // adornos
    g.fillOval(120,96,40,40);
    g.fillArc(173,100,40,40,110,180);
  }
}
```


Filtros para imágenes

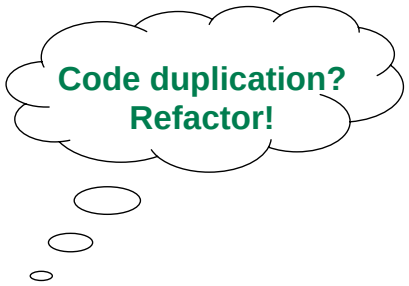
- Funciones aplicadas a las imágenes

```
int height = getHeight();
int width = getWidth();
for(int y = 0; y < height; y++) {
    for(int x = 0; x < width; x++) {
        Color pixel = getPixel(x, y);
        alter the pixel's color value;
        setPixel(x, y, pixel);
    }
}
```

Añade filtros nuevos

```
private void makeLighter()
{
    if(currentImage != null) {
        currentImage.lighter();
        frame.repaint();
        showStatus("Applied: lighter");
    }
    else {
        showStatus("No image loaded.");
    }
}
```

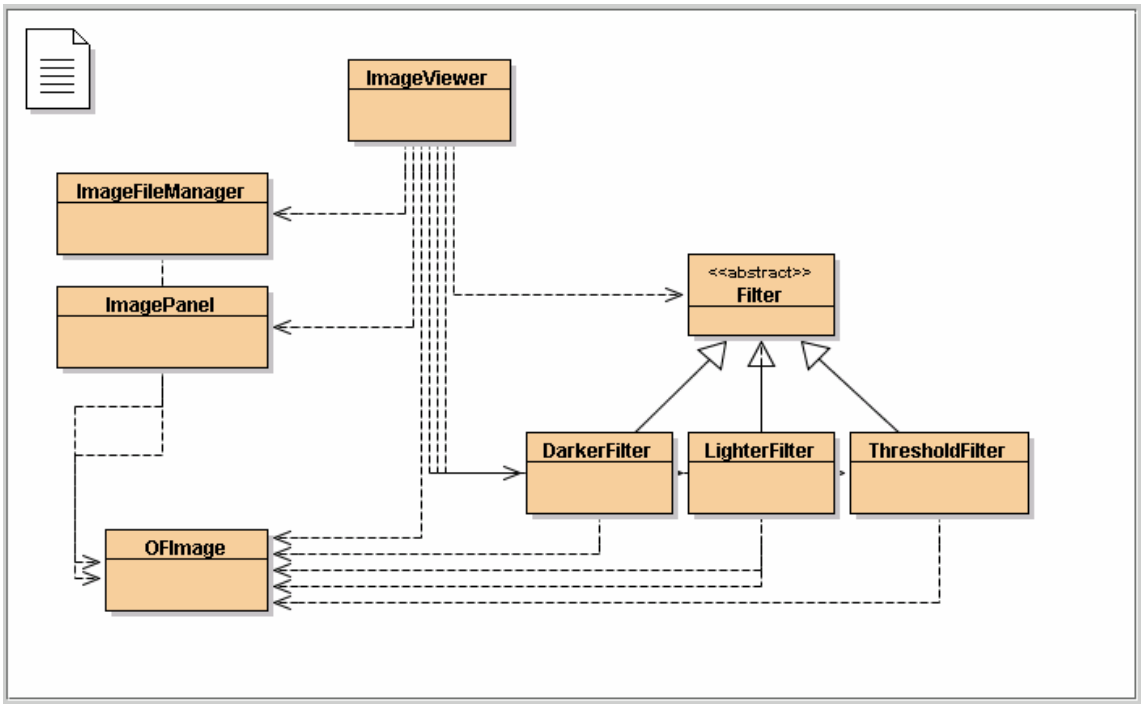
```
private void threshold()
{
    if(currentImage != null) {
        currentImage.threshold();
        frame.repaint();
        showStatus("Applied: threshold");
    }
    else {
        showStatus("No image loaded.");
    }
}
```



Más filtros

- Define una superclase (abstract) `Filter`
- Crear subclases para funciones específicas
- Crear una colección de instancias de subclases en `ImageViewer`.
- Definir un método genérico `applyFilter`
- Ver *imageviewer2-0*

imageviewer2-0



Buttons and nested layouts

A GridLayout inside a FlowLayout inside a BorderLayout.

Borders

- Used to add decoration around components.
- Defined in `javax.swing.border`
  - `BevelBorder`, `CompoundBorder`, `EmptyBorder`, `EtchedBorder`, `TitledBorder`.

Adding spacing

```
JPanel contentPane = (JPanel)frame.getContentPane();
contentPane.setBorder(new EmptyBorder(6, 6, 6, 6));

// Specify the layout manager with nice spacing
contentPane.setLayout(new BorderLayout(6, 6));

imagePanel = new ImagePanel();
imagePanel.setBorder(new EtchedBorder());
contentPane.add(imagePanel, BorderLayout.CENTER);
```

Resumen

- El diseño de toda interfaz conlleva, a grandes rasgos, los siguientes pasos:
  - Decidir la estructura de la interfaz
    - Qué componentes gráficos se van a utilizar, y cómo se van a relacionar estos componentes)
  - Decidir la disposición (*layout*) de los componentes
    - Existen dos tipos de componentes: contenedores y componentes atómicos
    - Los contenedores sirven para organizar los componentes contenidos en los mismos. Esta organización se denomina disposición (o *layout*)
  - Decidir el comportamiento de la interfaz: gestión de eventos
    - Algunos componentes son controles: permiten reaccionar ante eventos del usuario. El comportamiento se especifica programando las respuestas a dichos eventos. Normalmente, dichas respuestas supondrán invocar funcionalidades de la lógica de la aplicación
    - Conviene mantener la interfaz y la lógica lo más independientes posibles (veremos patrones que permiten lograr esto)

Resumen

- Funcionamiento de aplicaciones con GUI
  - La aplicación avanza guiada por los eventos del usuario
  - Cuando, desde el código de la aplicación, se realizan cambios sobre la interfaz (p.ej. cambiar el texto de una etiqueta, dibujar algo, limpiar un cuadro de texto, etc.) tales cambios no son inmediatos, sino que se encolan (en estructuras internas del framework AWT/Swing) para ser procesados por el hilo de tratamiento de eventos una vez que termine la ejecución del código del oyente que ha provocado los cambios
    - La ejecución de métodos que cambian la presentación de componentes no debe interpretarse como ejecuciones, sino como promesas de ejecuciones (que se realizarán cuando sea posible)
  - El tratamiento de los eventos debe ser rápido
    - La ejecución de los oyentes se lleva a cabo en el hilo de tratamiento de eventos: si tal ejecución tarda mucho, o se bloquea, bloquea a dicho hilo, que no puede tratar el resto de eventos rutinarios, por lo que la aplicación en sí se bloquea, y no responderá
    - Si un tratamiento de un evento necesita mucho tiempo para ser realizado, una solución puede ser crear otro hilo que lo lleve a cabo (no es sencillo porque aparecen los problemas de sincronización de hilos)