Interfaces de Usuario Gráficas en JAVA

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Interfaces de Usuario Gráficas en Java

Una **interfaz de usuario gráfica o GUI** permite a los usuarios interactuar con dispositivos electrónicos a través de íconos gráficos e indicadores visuales.

Las **interfaces de usuario gráficas** se crean a partir de componentes de GUI o controles o *widgets*. Una componente de GUI es un objeto con el cual interactúa el usuario usando algún dispositivo como el mouse, el teclado, mediante reconocimiento de voz, etc.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Interfaces de Usuario Gráficas en Java

✓**AWT – Abstract Windowing Toolkit**

Forma parte de la distribución de JSE. Primera iniciativa de toolkit JAVA para construir interfaces de usuario gráficas.

✓**JFC, Swing – Java Foundation Classes**

Forma parte de la distribución de JSE. Se incorporó en la versión 2 de JAVA.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Abstract Window Toolkit -AWT-

El AWT es el primer framework JAVA multiplataforma para escribir interfaces gráficas de usuario.

AWT está basado en las componentes nativas del sistema operativo.

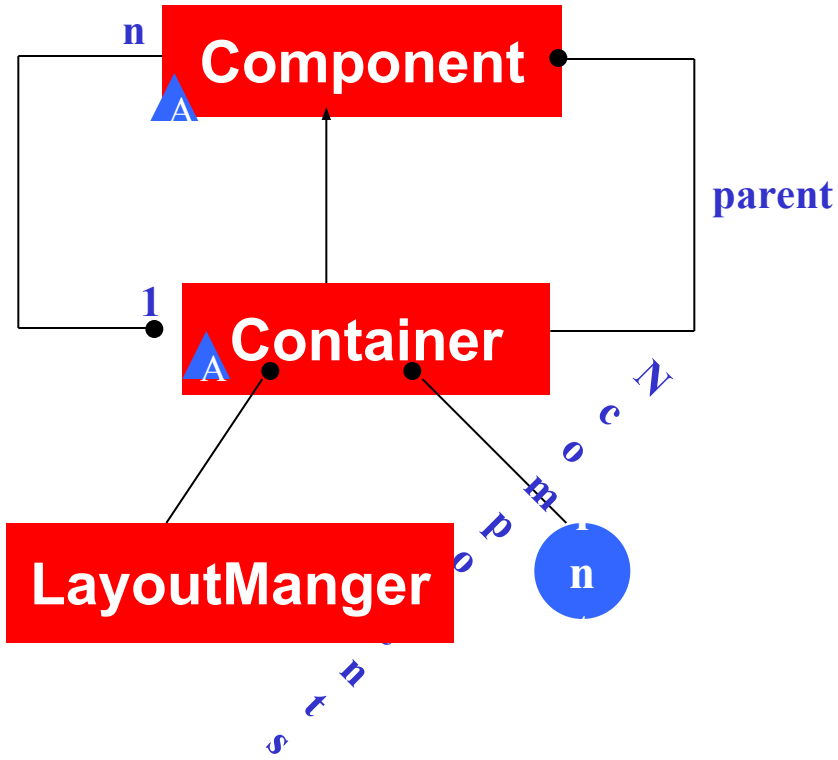
Aproximadamente la mitad de las clases del AWT extienden de la clase **java.awt.Component**.

El fundamento del AWT lo constituyen:

- ✓**Clase Component**: es una clase abstracta que agrupa componentes GUI estándares como botones, menúes, listas, etiquetas, etc.
- ✓**Clase Container**: es una clase abstracta subclase de Component. Tiene la capacidad de contener múltiples componentes. Applet, Panel, Window, Dialog y Frame son subclases de Container. Son componentes de nivel de ventana.
- ✓**Las interfaces LayoutManager**, **LayoutManager2**: definen métodos para la ubicación, tamaño, espaciado y distribución de las componentes dentro de un contenedor. Java provee varias clases que implementan estas interfaces, son los administradores de posicionamiento o *layout managers*.
- ✓**Clase Graphics**: es una clase abstracta que define métodos para realizar operaciones gráficas sobre una componente (mostrar imágenes, texto, establecer colores y fonts). Toda componente AWT tiene asociado un objeto Graphics donde dibujarse.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

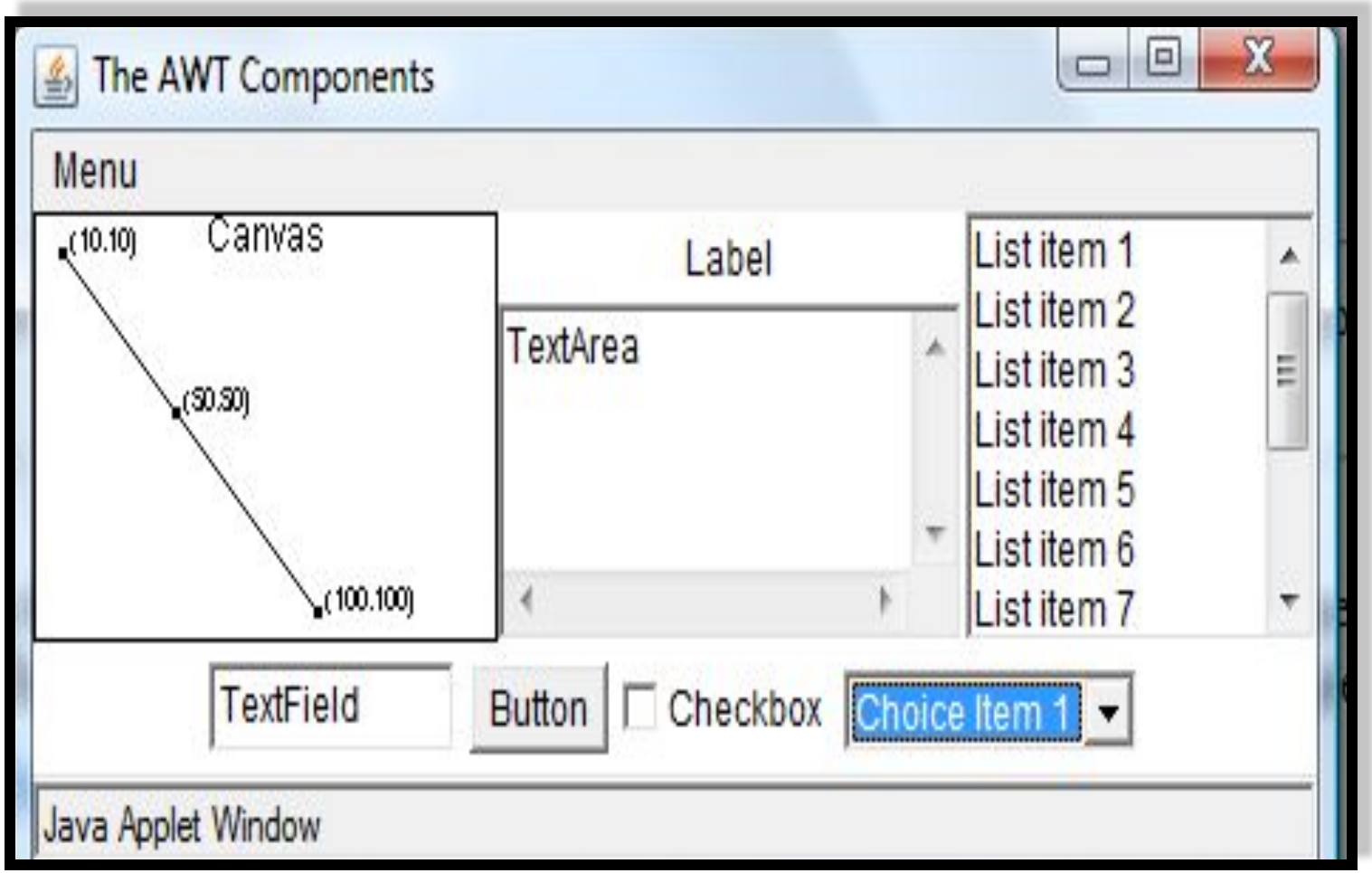
Componentes, Contenedores y Administradores de Disposición



- ✓AWT establece una relación simple y fundamental entre objetos Component y Container: *un contenedor puede contener múltiples componentes.*
- ✓Cada objeto Component tiene un Container padre.
- ✓Todos los *contenedores tienen asociado un layout manager* que establece la ubicación y el tamaño de las componentes dentro del Container. Cada *contenedor* tiene asociado un único *layout manager*.
- ✓Las responsabilidades del administrador de disposición, las definen las interfaces **java.awt.LayoutManager** y **java.awt.LayoutManager2**

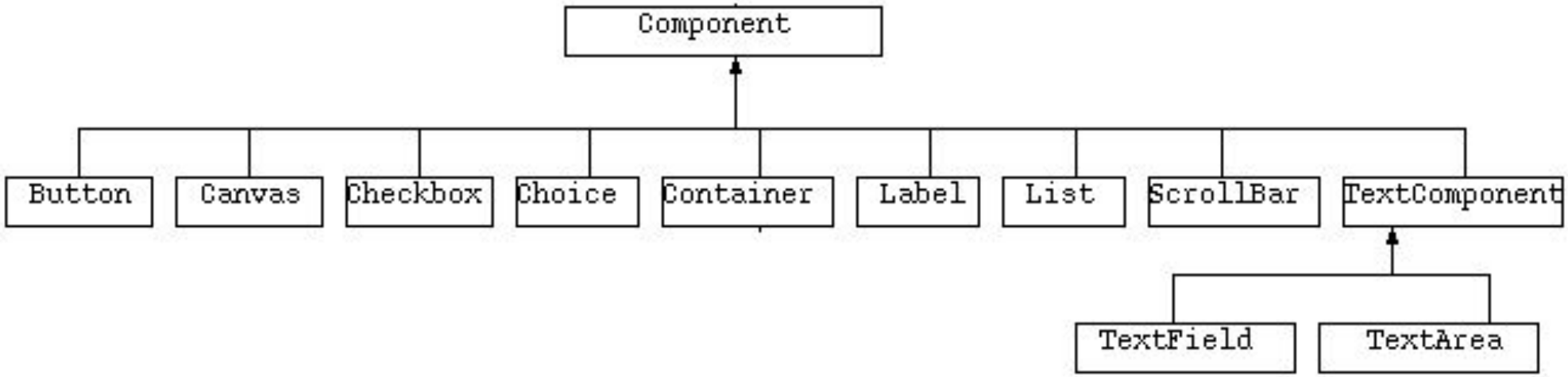
Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Componentes AWT Estándares



Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Componentes AWT



La clase **java.awt.Component** es una **clase abstracta** superclase de todas las componentes de GUI. Encapsula la funcionalidad común de todas las componentes AWT.

Cada componente AWT tiene asociada la siguiente información:

- ✓Un objet Graphics (donde dibujarse)
- ✓Posición
- ✓Tamaño
- ✓Peer nativo (componente de GUI del sistema nativo)
- ✓Contenedor padre
- ✓Fonts y dimensiones del font
- ✓Colores de fondo y de frente
- ✓Tamaño mínimo, máximo y preferido

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Componentes AWT y Peers

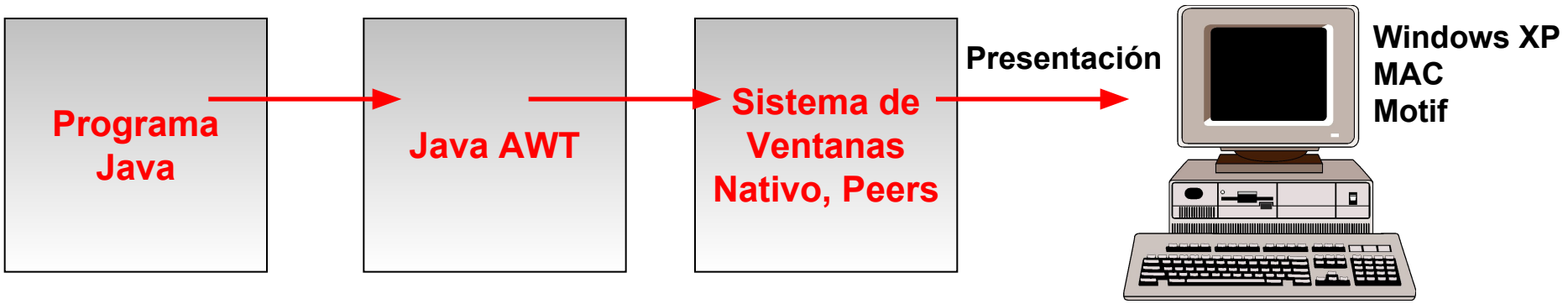
- ✓ El AWT original (versión 1.0) basó el diseño de sus componentes de GUI en **peers**.
- ✓ Los **peers** son componentes de GUI nativas del sistema operativo local (donde se ejecutan) y son manipuladas por componentes AWT JAVA. Las clases AWT encapsulan a los **peers**. El **peer** es el responsable de la interacción entre la componente JAVA y la plataforma local.
- ✓ Las componentes AWT son ***wrappers*** de los **peers** (envuelven a las componentes nativas) y delegan en ellos la mayor parte de su funcionalidad: cómo mostrarse, su apariencia o *look* y cómo comportarse ante la interacción del usuario o reacción ante eventos *feel*.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Componentes AWT y Peers

Cada componente AWT tiene una componente peer (nativa) equivalente. Está en el paquete **java.awt.peer**

Ejemplo: cuando se usa AWT para crear una instancia de la clase *Menu*, la JVM crea una instancia de un *menu peer* (java.awt.peer.MenuPeer). El peer es el que hace el trabajo real de mostrarse (look) y manejar el comportamiento del menú (feel). En este caso, el JRE de Solaris creará un objeto menú peer Motif; el JRE de Windows creará menú peer Windows; el JRE de Macintosh creará menú peer Macintosh, etc.



El programa JAVA crea y despliega una componente AWT, quién a su vez crea y despliega una componente nativa (peer).

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Componentes AWT Heavyweigth

Las componentes AWT estándares son **heavyweigth**:

- ✓ Tienen un **peer nativo**, por lo tanto no pueden extenderse para modificar el comportamiento por defecto. Están atadas a la plataforma local.

- ✓ Se **pintan en su propia ventana** nativa. En general tienen forma rectangular y no son transparentes.

En la versión 1.0 del AWT la implementación de componentes personalizadas (por ej. un botón con imágenes) implicaba extender la clase java.awt.Canvas. No se podían extender java.awt.Component ni java.awt.Container.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Componentes AWT Lightweigth

- El AWT 1.1 introduce componentes de GUI **lightweigth**:
  - No están basadas en **peers**.
  - Se **pintan en la ventana de su contenedor heavyweigth** (en el objeto graphic del contenedor). Tienen forma inorgánica y son transparentes.
  - Se implementan extendiendo las clases **java.awt.Component** ó **java.awt.Container**.
- Las componentes lightweigth necesitan un contenedor heavyweigth donde dibujarse (por eso son transparentes). Esto no es un inconveniente ya que las componentes residen en un Applet ó en un Frame que son contenedores **heavyweigth** (en el caso de aplicaciones de escritorio).
- Las componentes Swing provistas en el JDK 1.2 son **lightweigth** y reemplazan a las componentes AWT **heavyweigth**. Además proveen componentes de GUI adicionales como tablas, árboles, cajas de diálogo, color choosers, etc.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Paquetes del AWT

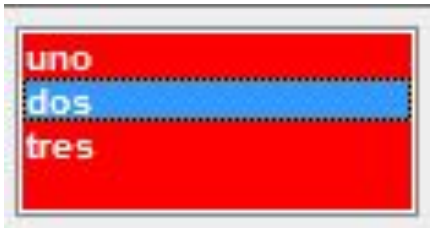
Paquetes AWT	Descripción
`java.awt`	componentes básicas de GUI
`java.awt.event`	clases de eventos e interfaces listeners
`java.awt.datatransfer`	soporte para clipboard y transferencia de datos
`java.awt.image`	clases para manipulación de imágenes
`java.awt.accessibility`	tecnologías para asistir al usuario
`java.awt.color`	manejo de colores
`java.awt.dnd`	soporte de drag & drop
`java.awt.font`	fonts 2D
`java.awt.geom`	geometría 2D
`java.awt.im`	métodos input
`java.awt.peer`	interfaces *peers* para componentes *peers*
`java.awt.print`	soporte de impresión

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Crear Componente AWT
Algunos fragmentos de código

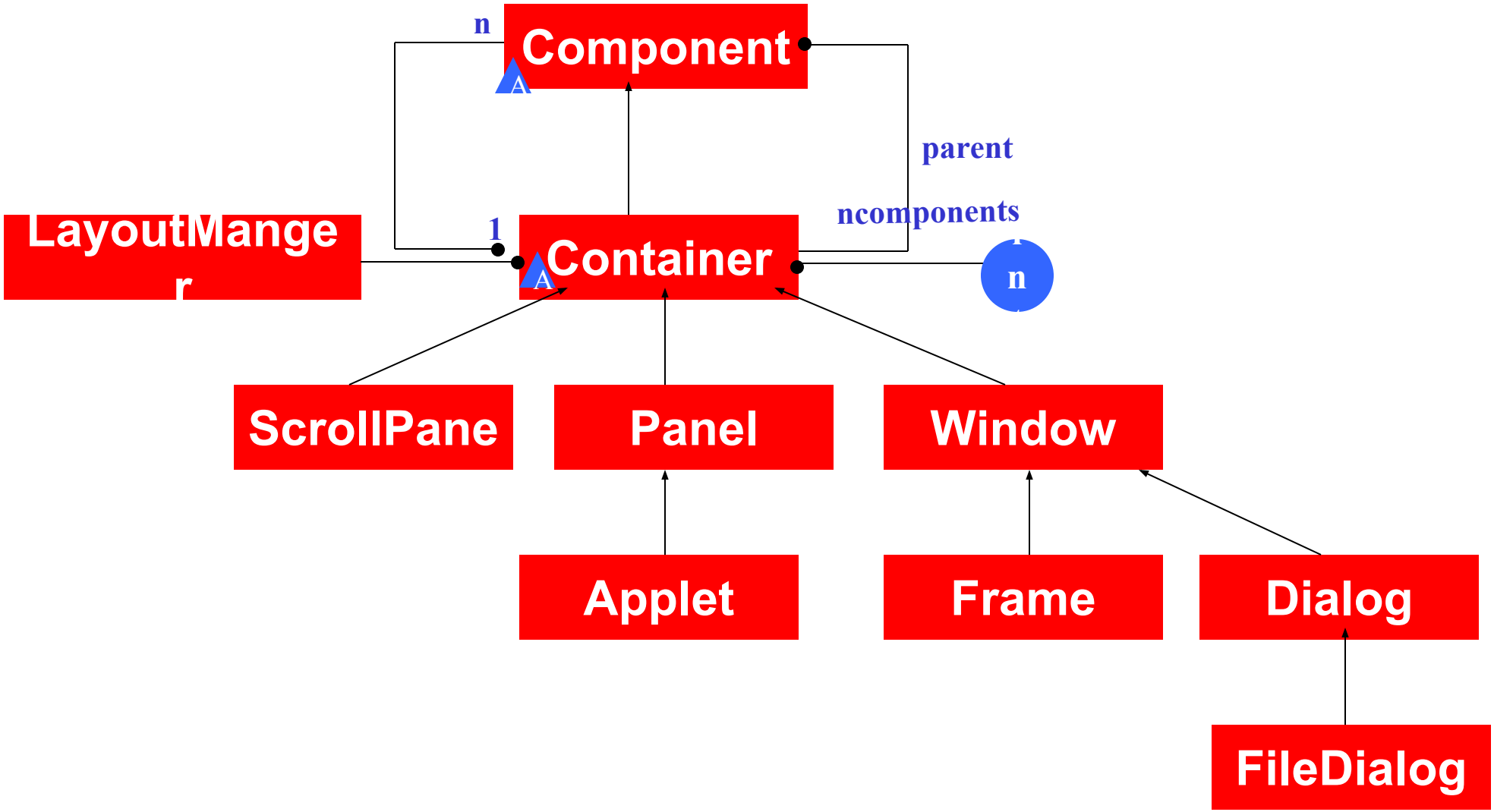
```
Button b = new Button("Hola");
b.setBackground(Color.red);
b.setForeground(Color.white);
```


```
List lista = new List();
lista.add("uno");
lista.add("dos");
lista.add("tres");
lista.setBackground(Color.red);
lista.setForeground(Color.white);
lista.setFont(new Font(Font.SANS_SERIF, Font.BOLD,10));
```



Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

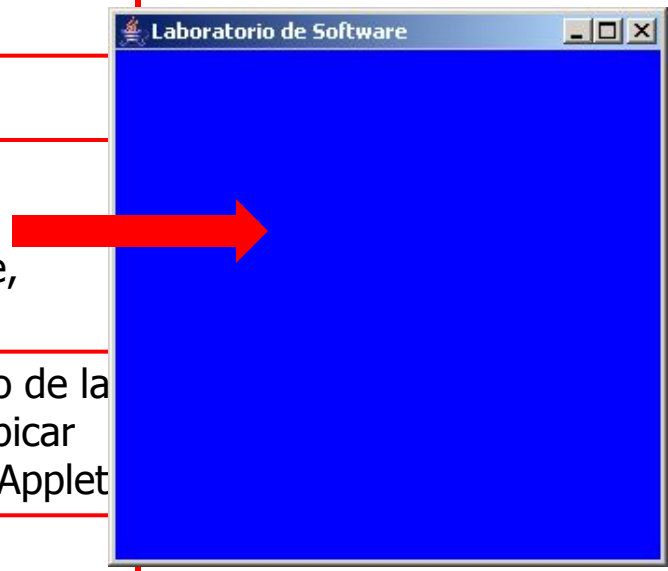
Contenedores AWT



Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Contenedores AWT

Subclases	Descripción
`Applet`	Es subclase de Panel. Es la superclase de todas los applets.
`Dialog`	Es subclase de Window. Puede ser modal o no-modal. Tiene borde y puede agrandarse, achicarse o minimizarse. No tiene barra de menúes. Es ideal para capturar inputs del usuario
`FileDialog`	Es un Dialog para elegir archivos.
`Frame`	Es subclase de Window. Es el contenedor de las aplicaciones de escritorio. Tiene borde, puede contener un menubar y agrandarse, achicarse o minimizarse. Tiene barra de título.
`Panel`	Es un contenedor simple. Es un área rectangular dentro de la que se pueden incluir componentes de GUI. Se debe ubicar dentro de un contenedor subclase de Window o en un Applet
`ScrollPane`	Permite hacer "Scroll" de una componente.
`Window`	No tiene menubar ni borde. Es la superclase de Frame y Dialog. Es el contenedor más simple, no tiene borde, ni barra de título, ni barra de menú y no puede agrandarse ni achicarse.



Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Contenedores AWT

✓La clase **java.awt.Container** es una **clase abstracta** subclase de **java.awt.Component**. Los objetos **Container** son simplemente componentes AWT que pueden contener otras componentes. Hay que tener en cuenta que los contenedores pueden contener otros contenedores ya que los **Containers** son **Component**. **Patrón Composite**

✓Todo objeto **Container** tiene una referencia a un objeto **LayoutManager**, en el que delega la responsabilidad de ubicar y distribuir las componentes. Cada vez que ocurre un evento que provoca que el contenedor tenga que acomodar sus componentes (por ej. cambió el tamaño de la ventana), el objeto **layout manager** del contenedor es invocado para reacomodar las componentes. Diferentes **Layout Managers** implementan diferentes algoritmos para reubicar componentes. **Patrón Strategy**

✓La clase **Applet** es subclase de **Panel**, que a su vez es subclase de **Container**, por lo tanto los *applets* heredan toda la capacidad de contener y acomodar componentes sobre la pantalla. Así se simplifica el diseño de *applets*. Se despliega en la ventana de un navegador de Internet.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Layout Manager

✓Los contenedores delegan en el *layout manager* todo lo relacionado a la ubicación, tamaño, espaciado y distribución de sus componentes en la pantalla.

✓Un contenedor tiene exactamente un **layout manager** y un mismo **layout manager** puede ser compartido por más de un contenedor.

✓Las interfaces **LayoutManager** y **LayoutManager2** definen métodos para calcular tamaños mínimos y preferidos de sus contenedores y ubicar las componentes en un contenedor (estrategias abstractas).

✓El AWT define clases que implementan las interfaces **LayoutManager** o **LayoutManager2** entre ellas se encuentran **BorderLayout**, **CardLayout**, **FlowLayout**, **GridBagLayout** y **GridLayout**. **Patrón Strategy**

✓**LayoutManager2** es subinterface de **LayoutManager**. **LayoutManager2** define métodos que permiten establecer restricciones sobre las componentes

✓Un contenedor puede *setear* su layout manager invocando al método **setLayout(LayoutManager)**

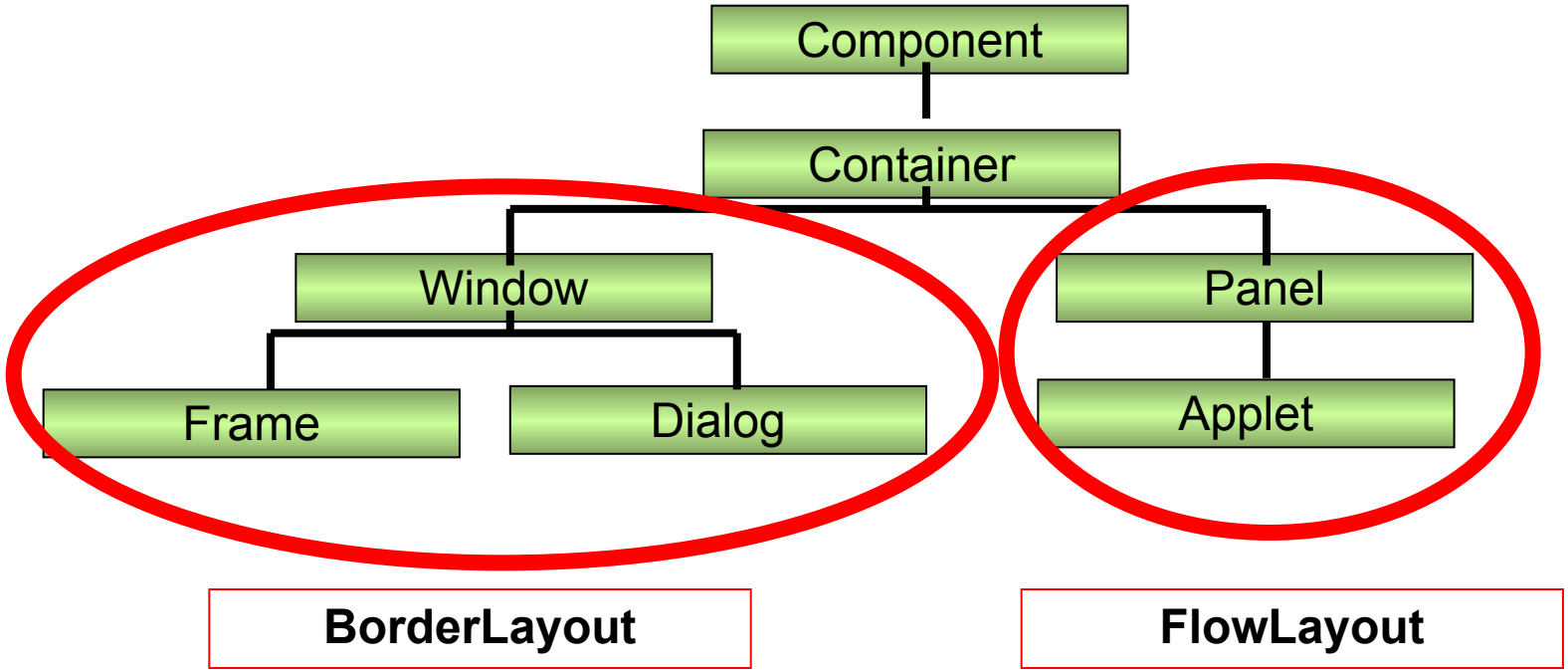
Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Layout Manager

`BorderLayout`	**Es el layout manager predeterminado de objetos Window**. Establece restricciones a las componentes acerca de dónde se ubicarán. Estas restricciones son constantes de la clase **BorderLayout**: **NORTH**, **SOUTH**, **EAST**, **WEST**, **CENTER**. Además es posible especificar el espaciado entre las componentes.
`CardLayout`	Es usado para contener en una misma área diferentes componentes en momentos diferentes. El contenedor actúa como una pila de cartas. Sólo un grupo de componentes es visible a la vez. Frecuentemente es controlado por un objeto Choice.
`FlowLayout`	**Es el layout manager predeterminado de objetos Panel**. Ubica las componentes de izquierda a derecha, usando nuevas filas si es necesario. Es posible alinear las componentes y establecer el espaciado entre las componentes.
`GridBagLayout`	Es el layout más sofisticado y flexible. Ubica las componentes en una grilla de filas y columnas permitiendo que algunas componentes usen más de una celda. No todas las filas tienen la misma altura ni todas las columnas el mismo ancho.
`GridLayout`	Divide el contenedor en una cuadrícula de manera que las componentes se ubican en filas y columnas. Todas las celdas tienen igual tamaño.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

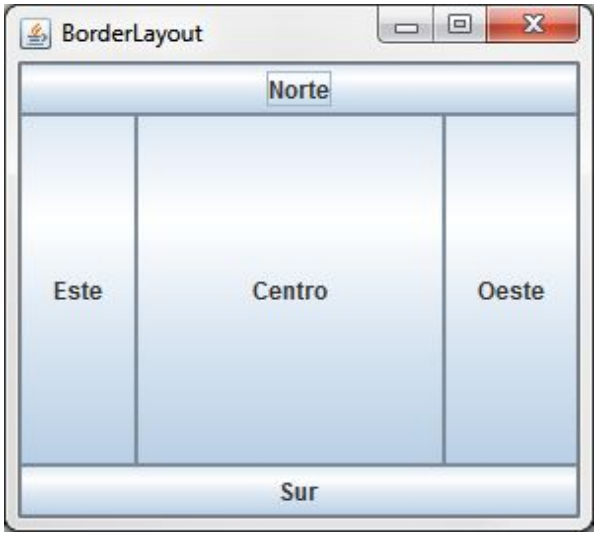
Layout Manager

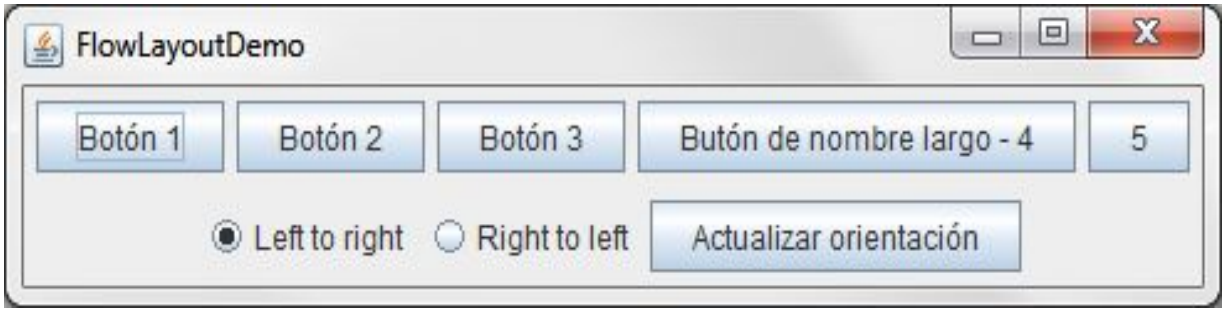


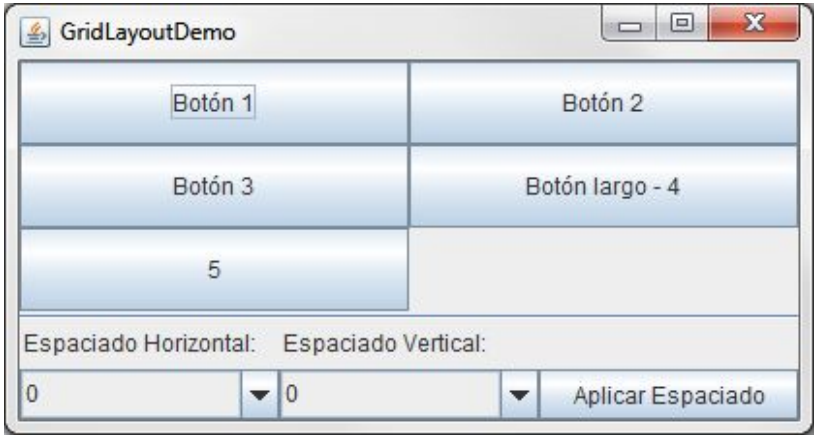
Cada **Contenedor** tiene asociado un objeto **Layout Manager** por defecto.

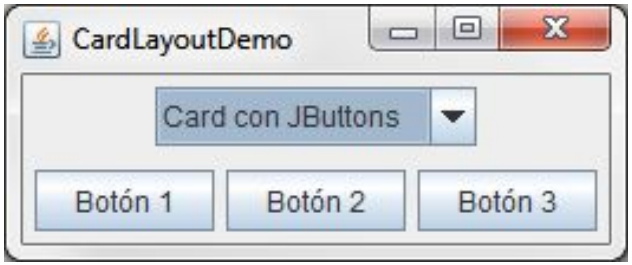
Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

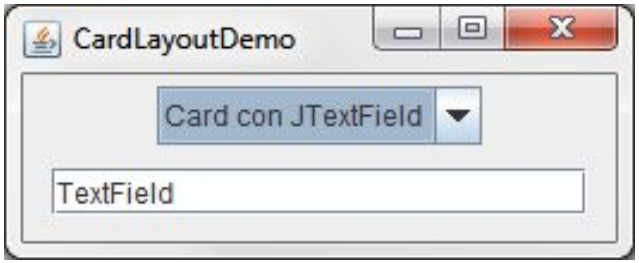
Layout Managers

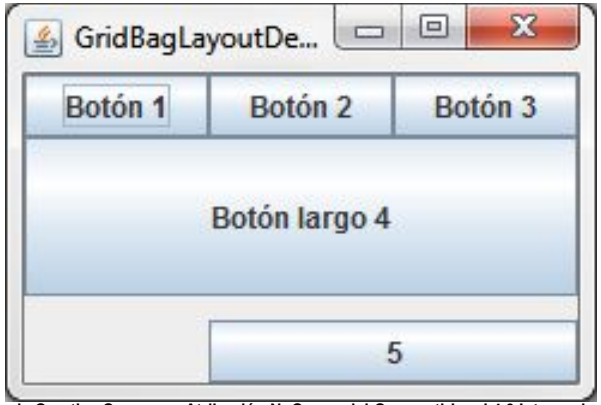












Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Layout Manager
Null y Customizados

✓En algunas situaciones puede ser útil usar un **layout manager customizado** en vez de los provistos por AWT.

✓Un **layout manager customizado** se construye escribiendo una clase que implemente la interface **java.awt.LayoutManager** o **java.awt.LayoutManager2**.

✓El método **setLayout()** de **java.awt.Container** permite establecer el objeto **layout manager customizado** que controlará la disposición de las componentes en el contenedor.

✓Es posible establecer un **layout manager** en **null** y explícitamente posicionar y establecer el tamaño de las componentes en el contenedor. Un **layout manager** en **null** no puede reaccionar ante los cambios, por ejemplo de tamaño de las componentes. En estos casos, el algoritmo de posicionamiento de las componentes tiene que escribirse en el contenedor.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

AWT - Ejemplo

```
public class Ventana {
   private Frame frame;
   private Button aceptar;
   private Button cancelar;
   public Ventana() {
      frame = new Frame("Laboratorio de Software");
      aceptar=new Button("Aceptar");
      cancelar=new Button("Cancelar");
   }
   public void comenzar() {
      frame.setBackground(Color.BLUE);
      frame.add(aceptar, BorderLayout.EAST);
      frame.add(cancelar, BorderLayout.WEST);
      frame.pack();
      frame.setVisible(true);
   }
   public static void main(String[] args) {
      Ventana miVentana = new Ventana();
      miVentana.comenzar();
   }
}
```

Contenedor

Agrega las componentes al frame

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Modelo de Manejo de Eventos de GUI

✓A partir del AWT 1.1 el manejo de eventos de la GUI está basado en el **modelo de delegación**. La versión original del AWT basó su manejo de eventos en herencia.

✓La premisa del modelo de delegación de eventos es simple: objetos que disparan eventos llamados *fuentes de eventos* y objetos que escuchan y procesan esos eventos llamados *listeners de eventos*.

✓Los objetos que disparan eventos son las *componentes* de GUI (fuentes generadoras de eventos) y *delegan* su manejo en *objetos listeners*.

✓Los *objetos listeners* son registrados sobre las *componentes* invocando al método: **addXYZListenter(XYZListener)**. Esto indica que el XYZListener procesará los eventos de tipo XYZ que genere la componente.

✓Luego de la registración, los métodos apropiados del *listener* serán invocados cuando el tipo de evento correspondiente ocurra sobre la componente.

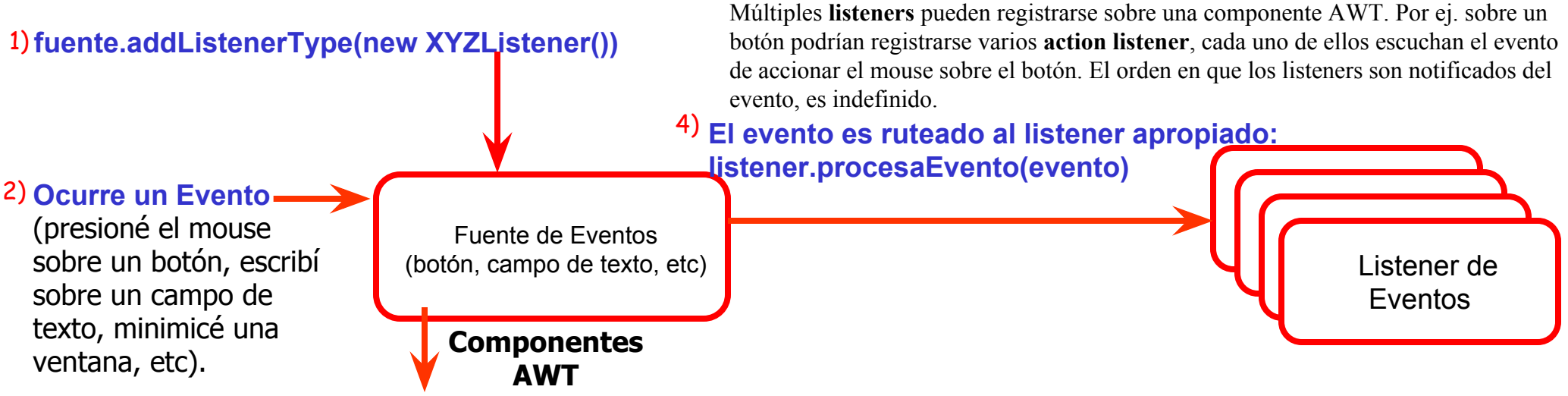
Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Manejo de Eventos en AWT 1.1

✓Las ***componentes* que generan eventos** implementan un método para la **registrar *listeners***, por ejemplo **Button.addActionListener(ActionListener)** y para **remover *listeners***, por ejemplo **Button.removeActionListener(ActionListener)**. De esta manera los objetos *listeners* registran interés en los eventos que la componente dispara o dejan de tener interés en ellos.

✓Cuando un evento ocurre, la componente que lo originó instancia un evento apropiado que es transmitido al/los *listener/s* que se registraro/n sobre la componente.

✓Los *listeners* son los responsables de implementar métodos para el manejo de eventos. Estos métodos tienen como parámetro un objeto evento que contiene información acerca del evento ocurrido y del objeto que lo originó.



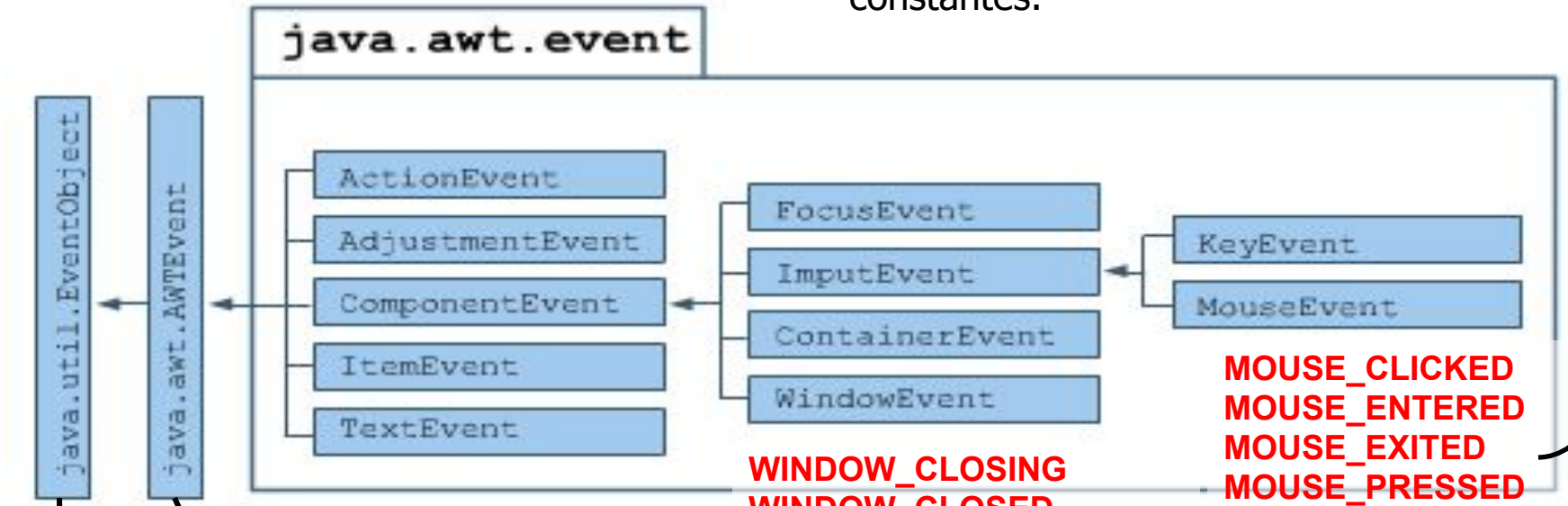
Múltiples **listeners** pueden registrarse sobre una componente AWT. Por ej. sobre un botón podrían registrarse varios **action listener**, cada uno de ellos escuchan el evento de accionar el mouse sobre el botón. El orden en que los listeners son notificados del evento, es indefinido.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Clases de Eventos en AWT

La clase **java.util.EventObject** junto con la interface **java.util.EventListener** constituyen el fundamento del modelo de delegación de eventos.

JAVA provee clases que representan los eventos de la GUI. Cada una de estas clases agrupa un conjunto de eventos relacionados codificados mediante constantes.



source

Mantiene un ID del evento

Object

int

Mantiene el objeto que originó el evento

WINDOW_CLOSING
WINDOW_CLOSED
WINDOW_ACTIVATED
WINDOW_DEACTIVATED
WINDOW_OPENED
WINDOW_ICONIFIED
WINDOW_DEICONIFIED
WINDOW_GAINED_FOCUS
WINDOW_LOST_FOCUS

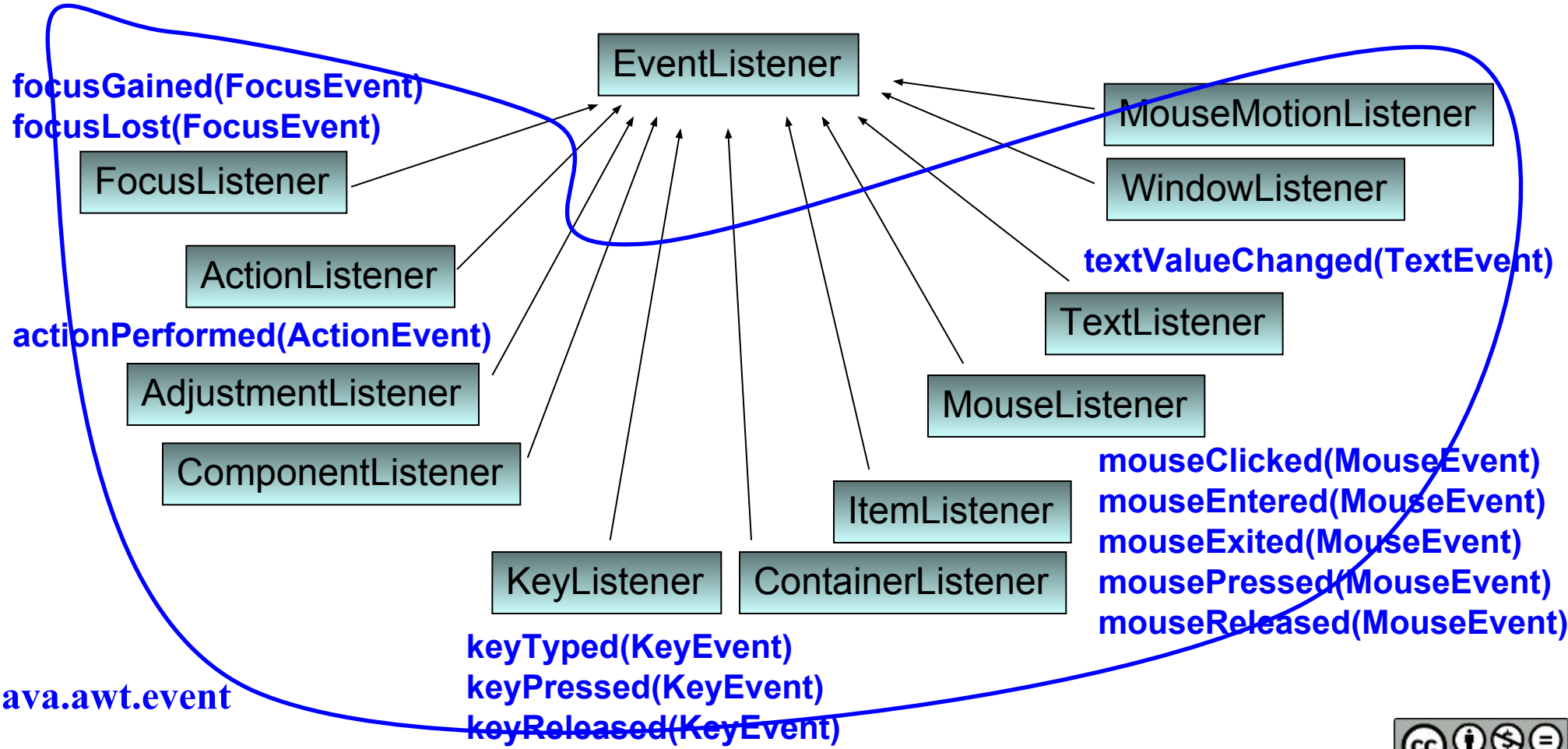
MOUSE_CLICKED
MOUSE_ENTERED
MOUSE_EXITED
MOUSE_PRESSED
MOUSE_RELEASED

Constantes de la clase **WindowEvent** que representan los eventos particulares de ventana: activar, desactivar, cerrar, abrir, minimizar y maximizar, etc.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Interfaces EventListener en AWT

Para cada tipo de eventos hay una **interface** que debe ser implementada por la clase que funcionará como **listener**. Esta interface exige que uno o más métodos sean implementados y éstos serán invocados cuando la componente genere un evento.
Las **11 interfaces listener** residen en el paquete **java.awt.event** excepto **java.util.EventListener**



Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Interfaces EventListener en AWT y las Clases Adapter

✓Cada **interface listener** define métodos que serán invocados cuando la componente genera un evento. Por ejemplo, la **interface ActionListener** declara un método llamado **actionPerformed()** que es invocado cuando un evento **action** es disparado por una componente (por ej. cuando se acciona el mouse sobre un botón).

✓ Los **listeners** registrados sobre una componente son los encargados de interceptar y procesar los eventos generados por la componente.

✓Para liberar a los programadores de tener que implementar todos los métodos definidos en la **interface listener**, AWT provee **clases adaptadoras**, que implementan las interfaces **listeners** con métodos que no hacen nada. Por ejemplo, **java.awt.event.MouseAdapter** es una clase abstracta que implementa todos los métodos definidos en la interface **java.awt.event.MouseListener** mediante métodos con cuerpo vació. El programador puede extender **MouseAdapter** y sobreescribir los métodos que atienden a los eventos de su interés en lugar de implementar la interface **MouseListener** completamente.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Ejemplo de Interfaces Listener

```
public class Ventana {
    private Frame frame;
    private Button aceptar, cancelar;
    public Ventana() {
        frame = new Frame("Laboratorio de Software");
        aceptar=new Button("Aceptar");
        cancelar=new Button("Cancelar");
    }
    public void comenzar() {
        aceptar.addActionListener(new MiListener());
        cancelar.addActionListener(new MiListener());
        frame.setBackground(Color.BLUE);
        frame.add(aceptar, BorderLayout.EAST);
        frame.add(cancelar, BorderLayout.WEST);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        Ventana miVentana = new Ventana();
        miVentana.comenzar();
    }
}
```

```
public class MiListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {
    String etiqueta="nada";
        if (e.getSource() instanceof Button){
            Button b=(Button)e.getSource();
            etiqueta=b.getLabel();
        }
        out.println("Accionar: "+etiqueta);
    }
}
```

Accionar: Aceptar
Accionar: Cancelar

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Ejemplo de Interfaces Listener

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

public class Ventana implements MouseListener {
   private Frame frame;
   private Button aceptar;
   private Button cancelar;
   private Button click;
   public Ventana (){
      frame = new Frame("Laboratorio de Software");
      aceptar=new Button("Aceptar");
      cancelar=new Button("Cancelar");
      mouse = new Button("Enter-Exit!");
   }
   public void comenzar() {
      aceptar.addActionListener(new MiListener());
      cancelar.addActionListener(new MiListener());
      mouse.addMouseListener(this);
      frame.setBackground(Color.BLUE);
      frame.add(aceptar, BorderLayout.EAST);
      frame.add(cancelar, BorderLayout.WEST);
      frame.add(mouse, BorderLayout.CENTER);
      frame.pack();
      frame.setVisible(true);
   }
```

El botón origina eventos con el mouse

```
public void mouseEntered(MouseEvent event){
System.out.println("Mouse Enter");
}
public void mouseExited(MouseEvent event){
System.out.println("Mouse Exit");
}
```

```
public void mousePressed(MouseEvent event){ }
public void mouseClicked(MouseEvent event){ }
public void mouseReleased(MouseEvent event){ }
```

```
public static void main(String[] args) {
      Ventana miVentana = new Ventana();
      miVentana.comenzar();
}
```

Mouse Entered
Mouse Exited
Mouse Entered
Mouse Exited
Mouse Entered
Mouse Exited

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Ejemplo con Clases Adaptadoras

```
public class Ventana2 {

private Frame frame;
private Button aceptar;

public Ventana2() {
   frame = new Frame("Laboratorio de Software");
   aceptar = new Button("Aceptar");
}

public void comenzar() {
   aceptar.addMouseListener(new ButtonMouseListener());
   frame.add(aceptar);
   frame.setBackground(Color.BLUE);
   frame.add(aceptar, BorderLayout.EAST);
   frame.pack();
   frame.setVisible(true);
}

private class ButtonMouseListener extends MouseAdapter {
   public void mouseEntered(MouseEvent event) {
       System.out.println("Mouse Enter");
  }

   public void mouseExited(MouseEvent event) {
       System.out.println("Mouse Exit");
   }
}
```

Las Clases Anidadas son ideales para manejo de eventos. Permiten escribir la clase que controla los eventos "cerca" del control que los dispara.
Las Clases Anónimas son un refinamiento de las clases Anidadas.

Clase anidada subclase de MouseAdapter

La clase MouseAdapter, implementa la interface MouseListener

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Java Foundation Classes -JFC-

Es un conjunto de tecnologías que permiten escribir GUI´s en JAVA para múltiples plataformas; está compuesto por:

- ✓**Componentes Swing**: implementan un conjunto de componentes de GUI enriquecidas, escritas en JAVA con "look & feel" *pluggable*.
- ✓**Look & Feel Pluggable:** las aplicaciones Swing tienen un look&feel enchufable
- ✓**API para Java 2D**: es un conjunto de clases que permiten agregar figuras, imágenes, animaciones a los controles de IU Swing.
- ✓**API para Accesibilidad**: consiste en un conjunto de clases que permiten a las componentes Swing interactuar con dispositivos de E/S alternativos que ayudan a los usuarios con discapacidades a interactuar con la aplicación, por ej. amplificadores de pantalla para personas con disminución visual, terminales braile, screen readers, sintetizador de voz, etc
- ✓**Internacionalización:** permite programar aplicaciones con las que pueden interactuar usuarios de diferentes idiomas y culturas.

La JFC se incorporó a la API Java a partir de la versión 1.2 de Java 2.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Componentes Swing

- ✓Constituyen un conjunto de componentes de interfaz de usuario, *lightweight*, construidas por encima de la infraestructura del AWT.
- ✓Están íntegramente desarrolladas en Java. Son 100% Java y no dependen de las componentes visuales del sistema operativo (peers).
- ✓Son un reemplazo de las componentes AWT. Además Swing provee múltiples componentes de GUI que no están presentes en el AWT: tabbed pane, toolbar, color chooser, table, tree, dialog boxes.
- ✓Proveen soporte para implementar interfaces de usuario gráficas con **look and feel pluggable.** Las componentes Swing pueden conservar su apariencia en plataformas diferentes.
- ✓Es posible crear componentes Swing *customizadas* con un L&F propio.
- ✓El diseño de las componentes Swing está basado en una versión modificada del MVC clásico, llamada **Arquitectura de Modelo Separado**. Cada componente consiste de un **Modelo** y un objeto de interfaz de usuario o **Delegado IU** que combina la Vista y el Controlador del MVC clásico.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

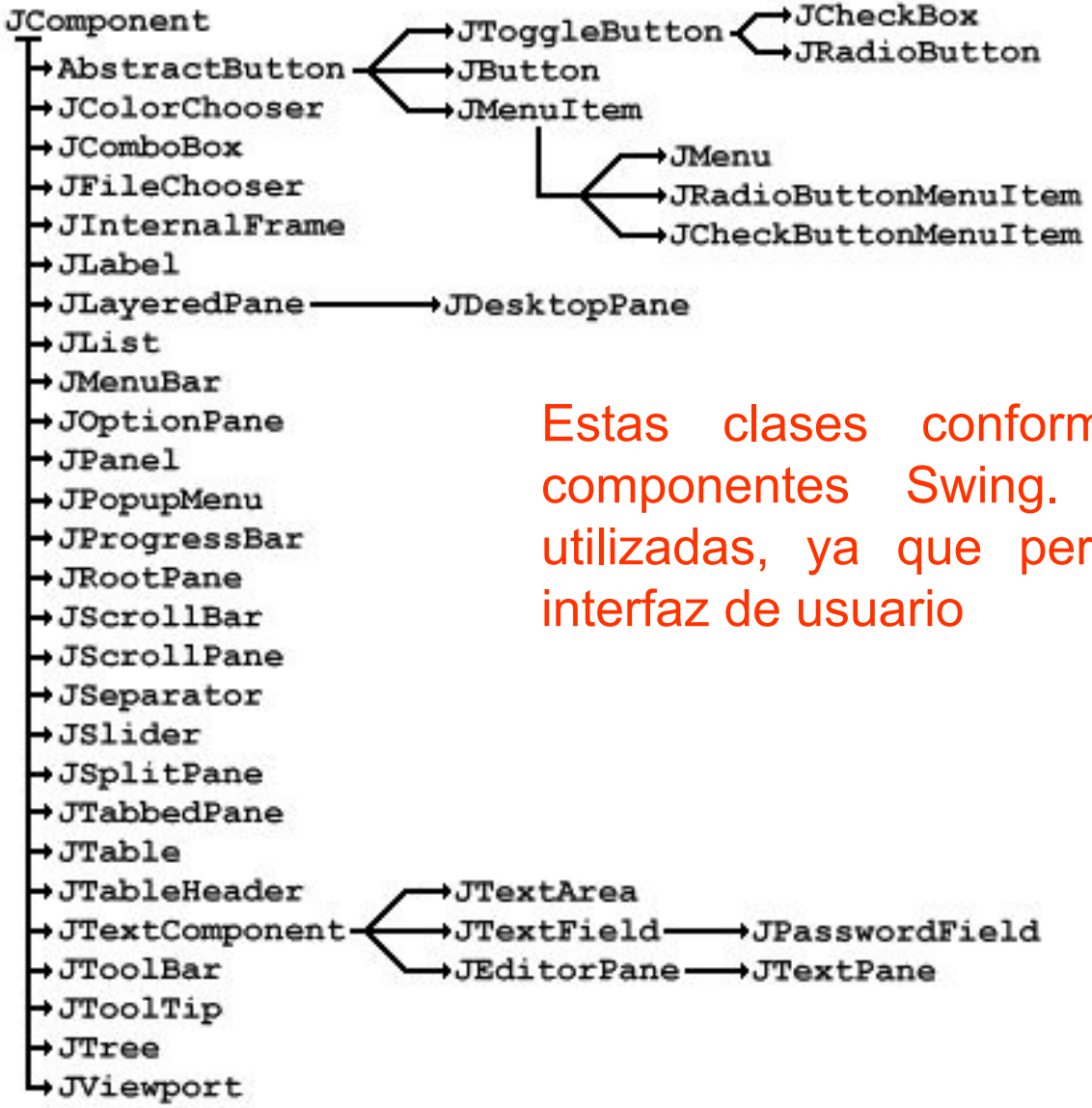
Componentes Swing

- La API Swing contiene alrededor de 250 clases. Para distinguir las clases que son componentes de interfaz de usuario de aquellas que son de soporte, los nombres de las componentes Swing comienzan con **J**.
- La API Swing está formado por 18 paquetes:

javax.swing	Componentes Swing
javax.swing.border	Bordes para componentes Swing
javax.swing.event	Clases de eventos e interfaces listener. Swing soporta los eventos y los listeners en java.awt.event
javax.swing.table	Clases de soporte para JTable
javax.swing.tree	Clases de soporte para JTree
javax.swing.colorchooser	Clases e interfaces que soportan a la componente JColorChooser usada para elegir colores de una paleta
javax.swing.filechooser	Clases e interfaces que soportan a la componente JFileChooser usada para elegir archivos
javax.swing.plaf	Contiene la API de L&F pluggable
javax.swing.text.html	Provee soporte para componentes de texto HTML
.....	.....

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

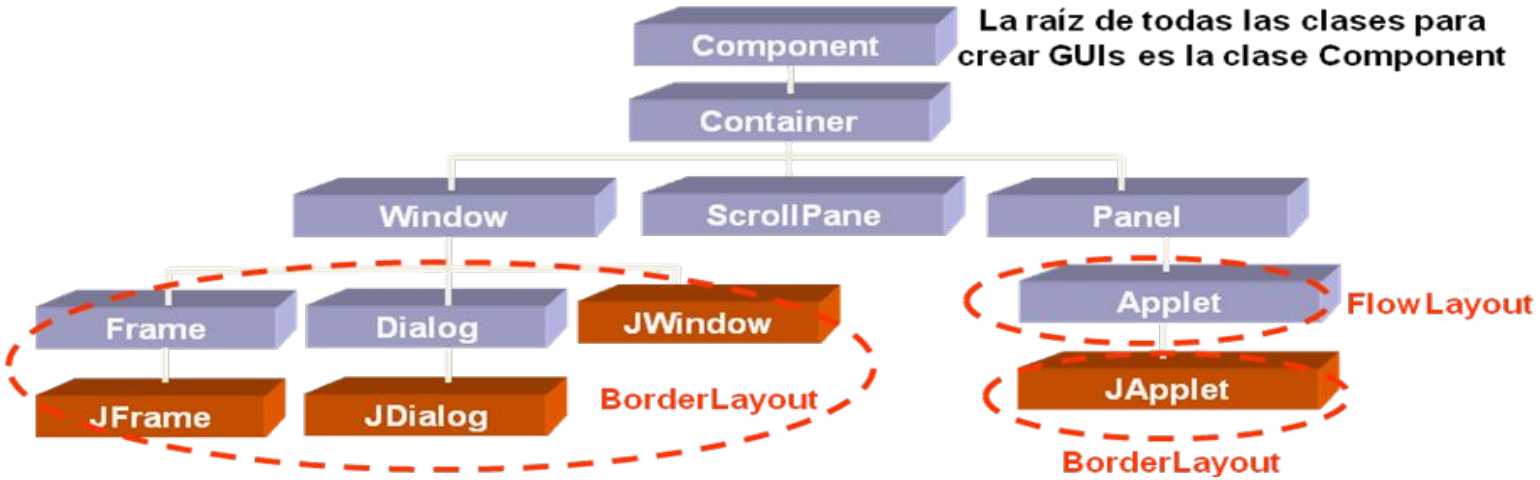
Componentes Swing



Estas clases conforman un subconjunto de las componentes Swing. Son las más comúnmente utilizadas, ya que permiten implementar objetos de interfaz de usuario

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Contenedores de Alto Nivel



✓Todo programa con una GUI Swing, debe tener al menos un contenedor Swing de alto nivel (de nivel de ventana). Este contenedor provee a las componentes Swing del soporte necesario para su pintado (provee un área donde se dibujan las componentes) y el manejo de eventos.

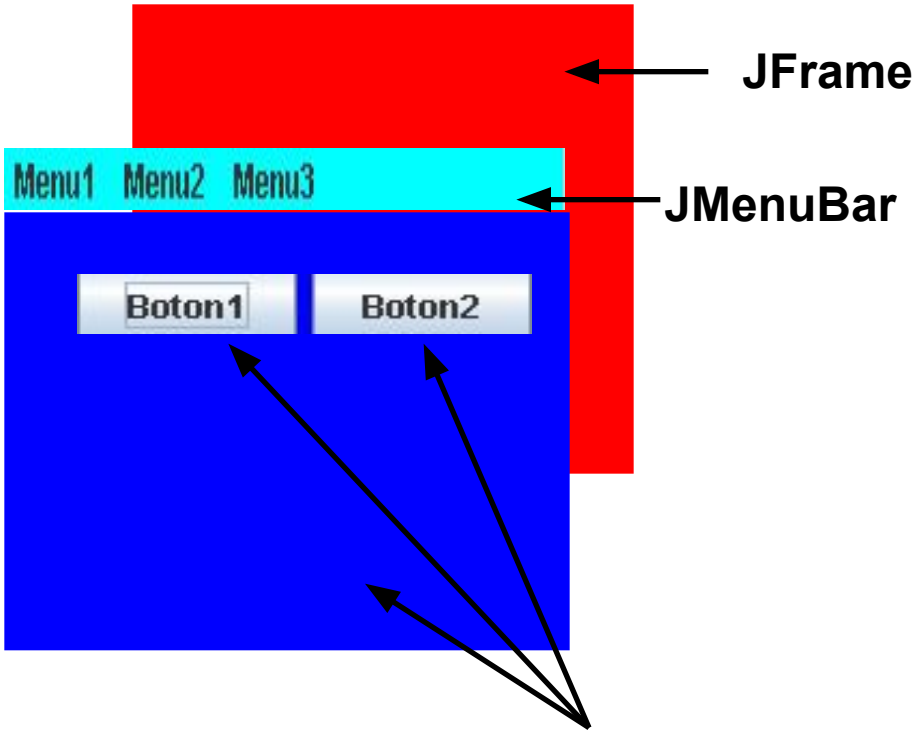
✓Los contenedores Swing de nivel de alto nivel son: **JFrame**, **JApplet**, **JDialog** y **JWindow**. Un objeto **JFrame** es una ventana principal; un objeto **JDialog** implementa una ventana secundaria, que depende de otra ventana y un objeto **JApplet** implementa el área de dibujo de un applet adentro de la ventana del navegador. **JWindow** es una ventana más simple, sin barra de título, sin bordes, sin controles para movimiento, ni botones en la barra de tareas. Tanto **JFrame** como **JWindow**, pueden usarse como ventana principal.

✓Es posible opcionalmente agregar una **barra de menúes** al contenedor de alto nivel (JFrame y JApplet), que en general está ubicada dentro de la ventana del contenedor, pero afuera del *content pane*.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

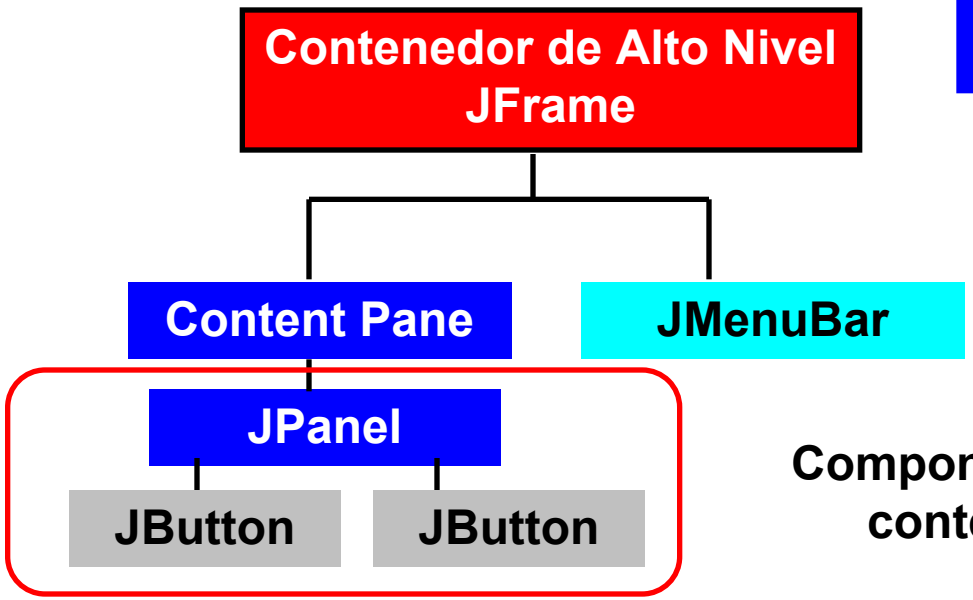
Contenedores de Alto Nivel

JFrame creado para una aplicación de escritorio Swing:



content pane con
un JPanel con 2 JButtons

Jerarquía de contención del ejemplo:



Componentes Swing en el contenedor JFrame

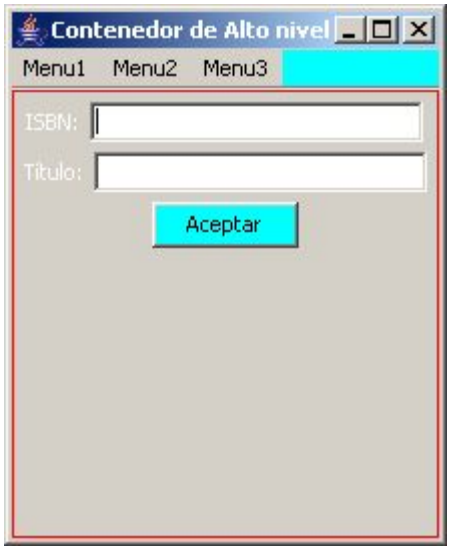
Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Agregar Componentes a un Contenedor

```
JFrame frame = new JFrame("Contenedor de Alto nivel");
JButton aceptar= new JButton("Aceptar");
JTextField inputISBN=new JTextField(20);
frame.getContentPane().add(aceptar, BorderLayout.CENTER);
frame.getContentPane().add(inputISBN, BorderLayout.LEFT);
```

El objeto **content pane** se recupera del contenedor de alto nivel, **JFrame**, invocando el método **getContentPane()**. Típicamente en el **content pane** se ubican las componentes.
El objeto **content pane** de defecto es un contenedor que es subclase de **JComponent** y utiliza un objeto **BorderLayout** como administrador de *layout* por defecto.
Es posible *customizar* el **content pane** *seteándole* un *layout manager* y agregándole un borde. Para ello es necesario *castearlo* a JComponent.

```
Container cont=frame.getContentPane();
JComponent contentpane=(JComponent) cont;
contentpane.setBorder(BorderFactory.createLineBorder(Color.red));
contentpane.setLayout( new FlowLayout());
contentpane.add(labelISBN);
contentpane.add(inputISBN);
contentpane.add(labelTit);
contentpane.add(inputTit);
contentpane.add(aceptar);
```



Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Agregar Componentes a un Contenedor

Crear una componente propia y setearla como **content pane**:

```
JPanel contentPane = new JPanel();
Border lineaRoja = BorderFactory.createLineBorder(Color.red);
title = BorderFactory.createTitledBorder(lineaRoja, "Alta de Libros");
title.setTitleJustification(TitledBorder.CENTER);
title.setTitleColor(Color.red);
contentPane.setBorder(title);
contentPane.add(labelISBN);
contentPane.add(inputISBN);
contentPane.add(labelTit);
contentPane.add(inputTit);
contentPane.add(aceptar);
frame.setContentPane(contentPane);
```

Otra opción es agregar una componente *customizada* al **content pane**, por ej. un JPanel, cubriéndolo completamente:

```
JPanel panelBlanco=new JPanel();
panelBlanco.setBackground(Color.white);
.....................
panelBlanco.add(labelISBN);
panelBlanco.add(inputISBN);
panelBlanco.add(labelTit);
panelBlanco.add(inputTit);
panelBlanco.add(aceptar);
frame.getContentPane().add(panelBlanco, BorderLayout.CENTER);
```

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Agregar Componentes a un Contenedor

Para facilitar la programación, el método **add(Component)** y sus variantes sobrecargadas así como los métodos **remove()** y **setLayout()** fueron sobreescritos de manera tal que la operación la hacen directamente sobre el contentPane.

```
frame.setLayout(new FlowLayout());
```

Se setea un objeto FlowLayout como administrador de posicionamiento del content pane.

```
frame.add(new JLabel("Hola"));
```

El JLabel se agregará en el content pane

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Agregar Barra de Menúes al Contenedor

Es posible agregarle una barra de menúes a los contenedores de alto nivel, **JFrame** y **JApplet**.
Se necesita crear un objeto **JMenuBar** y agregarle menúes y finalmente invocar al método **setJMenuBar(JMenuBar)** sobre el contenedor.

```
JMenuBar barraMenues = new JMenuBar();
barraMenues.setPreferredSize(new Dimension(200, 20));
barraMenues.add(new JMenu("Menu1"));
barraMenues.add(new JMenu("Menu2"));
barraMenues.add(new JMenu("Menu3"));
frame.setJMenuBar(barraMenues);
```


Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

La Clase JComponent

A excepción de los contenedores de alto nivel **JFrame**, **JApplet** y **JDialog**, el resto de componentes Swing, son subclases de la clase **JComponent**.
Por ejemplo **JPanel**, **JScrollPane**, **JButton**, **JTable, etc.** heredan de **JComponent**.

Las componentes Swing heredan de **JComponent** la siguiente funcionalidad:

Tooltips: el método **setToolTipText(String)** permite proveerle de ayuda al usuario de la componente, especificando un texto.

Ej: se le agrega un tooltip a un botón

```
b1.setToolTipText("Agrega un nuevo Libro");
```

Bordes y Pintado: el método **setBorder()** permite especificar el borde que desplegará la componente. Los objetos Border no son componentes Swing. Saben dibujar los bordes de las componentes Swing. Son útiles púes permiten proveer de títulos a las componentes. El método **paintComponent()** permite pintar adentro de la componente. Sobrescribiéndolo es posible pintar la componente en forma personaliza.

```
Border lineaNegra = BorderFactory.createLineBorder(Color.black);
titulo = BorderFactory.createTitledBorder(lineaNegra, "Alta de Libros");
titulo.setTitleJustification(TitledBorder.CENTER);
titulo.setTitleColor(Color.white);
panelAzul.setBorder(titulo);
```

Creo un borde Swing

Es posible crear bordes *customizados*

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

La Clase JComponent

Las componentes Swing heredan de **JComponent** la siguiente funcionalidad (continuación):

L&F: la arquitectura de Swing permite cambiar el L&F de la GUI de la aplicación. El *Look* hace referencia a la apariencia de las componentes y el *Feel* a cómo reaccionan ante eventos. Es posible elegir el L&F de las componentes Swing invocando al método UIManager.setLookAndFeel().

```
try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (UnsupportedLookAndFeelException e) {
       // TODO
    } catch (IllegalAccessException e) {
       // TODO
    } catch (ClassNotFoundException e) {
       // TODO
    } catch (InstantiationException e) {
       // TODO
    }
}
```

La implementación de SUN de la JRE provee los siguientes L&Fs:

CrossPlatformLookAndFeel: es el **Java L&F**, también llamado **Metal**. Se ve igual en todas las plataformas de ejecución y forma parte del paquete **javax.swing.plaf.metal**. Es el de defecto.

SytemLookAndFeel: la aplicación usa el L&F del sistema nativo donde se está ejecutando la aplicación. Se determina en ejecución.

Synth: permite crear nuestro propio L&F usando un archivo XML.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.