Java y Herramientas de Desarrollo

Sesión 5: Introducción a los clientes ricos

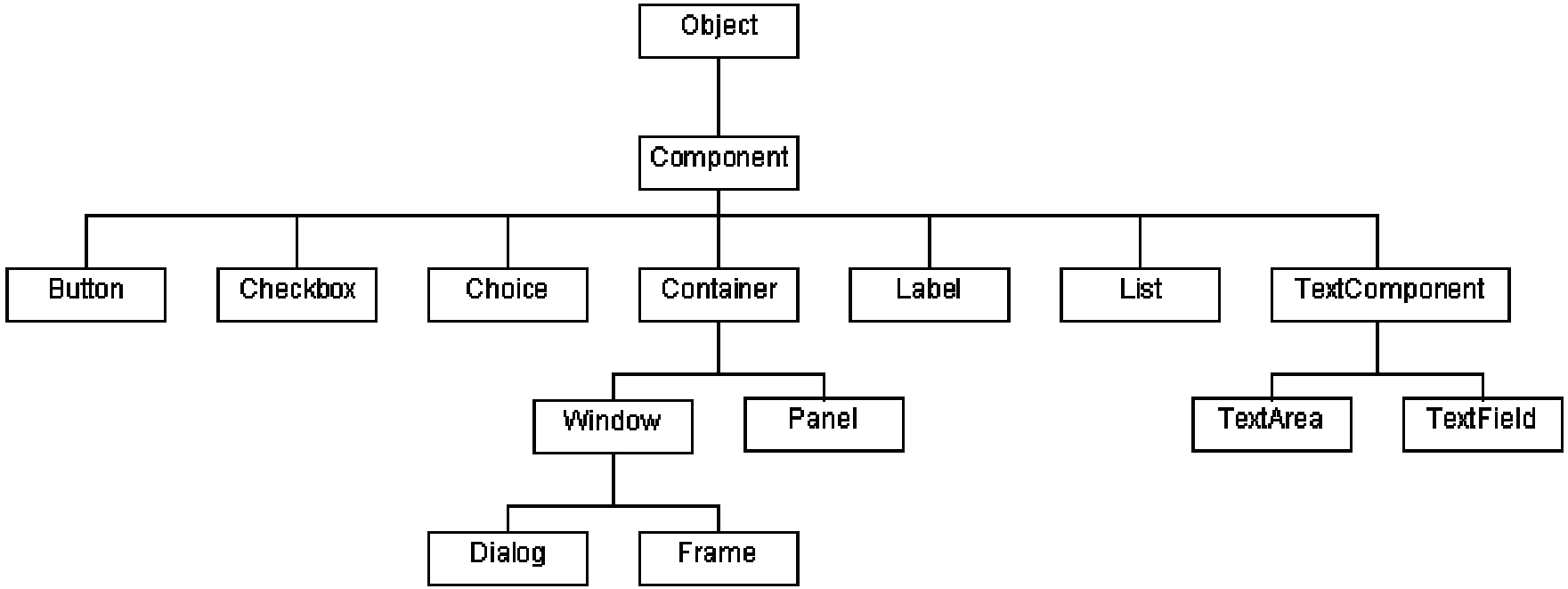
Puntos a tratar

- Introducción a AWT
- Componentes AWT
- Introducción a Swing
- Componentes Swing
- Gestores de disposición
- Modelo de eventos de Java

Introducción a AWT

- AWT (*Abstract Windows Toolkit*) es la librería original de Java para construir aplicaciones gráficas
- Disponible con las primeras versiones de Java, aunque en la 1.1 sufrió un cambio notable
- Todos los controles de esta librería se encuentran en el paquete **java.awt**

Esquema de componentes AWT



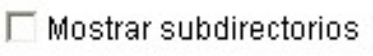
Componentes simples

Botones

```
Button btn = new Button(“Pulsame”);
```

Etiquetas

```
Label lbl = new Label(“Etiqueta”);
```



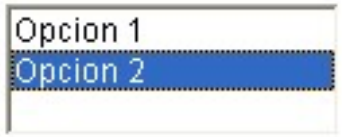
Casillas de verificación

```
Checkbox cb = new Checkbox(“Mostrar subdirectorios”);
System.out.println (“Marcado: “ + cb.getState());
```

Componentes simples

Listas desplegables:

```
Choice ch = new Choice();
ch.addItem("Opcion 1");
ch.addItem("Opcion 2");
String s = (String)(ch.getSelectedItem());
```



Listas fijas:

```
List lst = new List(3, true);
lst.addItem("Opcion 1");
lst.addItem("Opcion 2");
String s = (String)(lst.getSelectedItem());
```

Componentes simples

Cuadros de texto de una línea:

```
TextField txt = new TextField(“Hello”);
System.out.println (“Texto: “ + txt.getText());
txt.setText(“Otro texto”);
```

Cuadros de texto de varias líneas:

```
TextArea ta = new TextArea(5, 20);
System.out.println (“Texto: “ + ta.getText());
ta.setText(“Otro texto”);
ta.append(“ más otro texto”);
```

Contenedores

Ventanas principales:

```
Frame f = new Frame();
f.add(new Button(“Hola”));
```

Diálogos:

```
Dialog dlg = new Dialog(f);
```

Paneles internos:

```
Panel p = new Panel();
p.add(new Button(“Hola”));
p.add(new Label(“Etiqueta”));
f.add(p);
```

Menús

Barra de menú (sólo se pueden añadir a Frames):

```
MenuBar mb = new MenuBar();
Frame f = new Frame();
f.setMenuBar(mb);
```

Menús y submenús (se añaden a la barra u otros menús):

```
Menu m1 = new Menu("Menu 1");
Menu m11 = new Menu("Menu 1.1");
m1.add(m11);
mb.add(m1);
```

Opciones de menú (se añaden a menús o submenús):

```
MenuItem mi1 = new MenuItem("Item 1.1");
CheckboxMenuItem cmi1=new CheckboxMenuItem("Check");
m1.add(mi1);
m1.add(cmi1);
```

Introducción a Swing

- Swing es una librería gráfica adicional que se incorporó al núcleo de Java a partir de la versión 1.2
- Los controles se encuentran en el paquete **javax.swing**
- Presenta controles similares a AWT pero con muchas más posibilidades

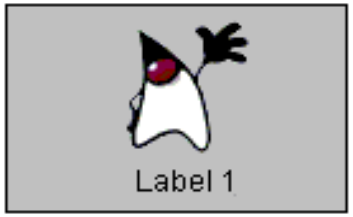
Diferencias entre AWT y Swing

- Apariencia de las aplicaciones configurable (*look & feel*)
- Los componentes Swing ofrecen más capacidades que los de AWT:
  - Posibilidad de mostrar iconos
  - Modificación de la apariencia de los componentes
  - Uso de bordes
  - Más variedad de componentes
- Cuidado al mezclar componentes Swing y AWT, puede haber solapes.

Componentes simples

Botones

```
JButton          == Button de AWT
JCheckBox        == Checkbox de AWT
JRadioButton
```



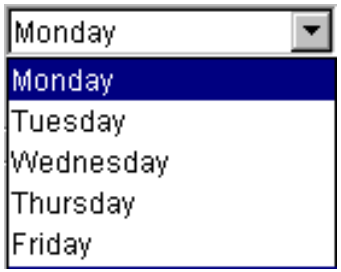
Etiquetas

```
JLabel           == Label de AWT
```

Cuadros de texto

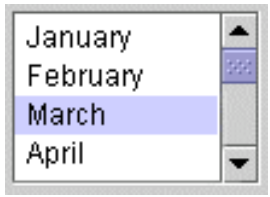
```
JTextField       == TextField de AWT
JTextArea        == TextArea de AWT
```

Componentes simples



Listas desplegables:

`JComboBox` `== Choice de AWT`



Listas fijas:

`JList` `== List de AWT`

Contenedores

Ventanas principales:

```
JFrame                                  == Frame de AWT
```

Diálogos:

```
JDialog                                 == Dialog de AWT
JFileChooser, JColorChooser, etc        == Diálogos especiales
```

Paneles internos:

```
JPanel                                  == Panel de AWT
```

Para ventanas y diálogos:

```
getContentPane()                        // Para ciertos métodos
```

Menús

Barra de menú (sólo se pueden añadir a JFrames):

```
JMenuBar mb              == MenuBar de AWT
JFrame f = new...
f.setJMenuBar(mb);
```

Menús y submenús:

```
JMenu                    == Menu de AWT
```

Opciones de menú (se añaden a menús o submenús):

```
JMenuItem                == MenuItem de AWT
JCheckBoxMenuItem        == CheckboxMenuItem de AWT
```

Gestores de disposición

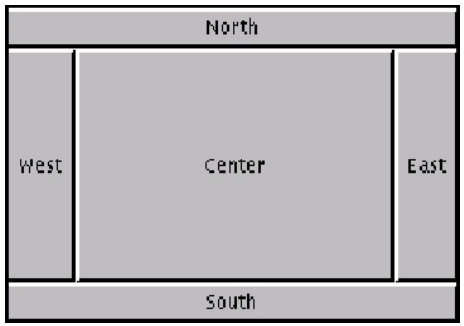
- Indican cómo colocar los componentes en un determinado contenedor (frame, diálogo o panel)
- Se establecen llamando al método *setLayout(...)* del contenedor, pasándole el tipo de gestor que se quiera:

```
JPanel p = new JPanel();
p.setLayout(new BorderLayout());
```

- Después se llama al método *add* del contenedor para añadir los componentes:

```
p.add(new Button(“Hola”));
```

Gestor *BorderLayout*



- Dividen la zona del contenedor en 5 regiones: Norte, Sur, Este, Oeste y Centro
- Al añadir componentes, se indica en qué zona colocarlos:

```
JPanel p = new JPanel();
p.setLayout(new BorderLayout());
JButton btn = new JButton (“Hola”);
p.add(btn, BorderLayout.SOUTH);
```

- No puede haber más de un componente en una zona (usar paneles para agrupar varios)
- Los componentes ocupan toda la zona en la que se añaden
- Es el gestor por defecto para Frames y Dialogs

Gestor *FlowLayout*

- Añaden los componentes conforme les llegan, bajando a la siguiente fila cuando ya no quepan más

```
JPanel p = new JPanel();
p.setLayout(new FlowLayout());
JButton btn = new JButton (“Ok”);
JButton btn2 = new JButton (“Open”);
p.add(btn);
p.add(btn2);
```

- Es el gestor por defecto para Panel
- El tamaño de los componentes es su tamaño mínimo o preestablecido

Gestor *GridLayout*

- Divide el área del contenedor en tantas filas y columnas como se le diga
- Añade los componentes por filas, de izquierda a derecha, hasta completar la malla

```
JPanel p = new JPanel();
p.setLayout(new GridLayout(2, 3));
p.add(new JButton (“1”));
p.add(new JButton (“2”));
...
```

- Todas las casillas tienen el mismo tamaño
- Los componentes ocupan todo el tamaño de la casilla

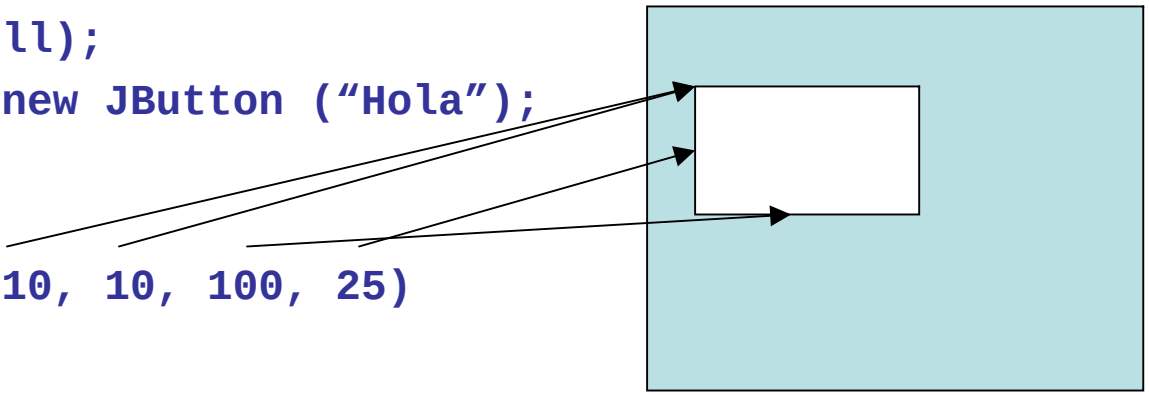
No utilizar gestor

- Podemos no utilizar gestor, e indicar nosotros dónde van los componentes y con qué tamaño
- Simplemente necesitamos hacer un *setLayout(null)* y luego utilizar el método *setBounds(...)* para indicar tamaño y ubicación

```
JPanel p = new JPanel();
p.setLayout(null);
JButton btn = new JButton (“Hola”);



btn.setBounds(10, 10, 100, 25)
p.add(btn);
```



- El tamaño de los componentes no se reajusta si redimensionamos el contenedor

Diferencias en gestores entre AWT y Swing

- En versiones anteriores a la 1.5 de Java, en las clases JFrame y JDialog se establece el gestor de forma diferente, ya que debemos llamar a su *getContentPane()* y luego establecerlo

```
JPanel p = new JPanel();
p.setLayout(new FlowLayout());

JFrame f = new JFrame();
f.getContentPane().setLayout(new BorderLayout());
```

- Para añadir componentes sobre JFrames y JDialogs, también debemos llamar antes a *getContentPane()*, y desde allí a *add()*:

```
p.add(new Button("Pulsame"));
f.getContentPane().add(new Label("Hola"));
```

- IMPORTANTE: esto SOLO pasa con JFrame y JDialog, no con JPanel, como se ve en los ejemplos, y SOLO para versiones anteriores a la 1.5

Modelo de eventos en Java

- Un EVENTO es una acción o cambio que ocurre en una aplicación y que permite que ésta emita una respuesta
- El modelo de eventos de Java se compone de:
  - FUENTE: elemento que genera el evento (normalmente, un componente de la aplicación)
  - OYENTE: elemento que espera el evento para producir la respuesta
- Para gestionar un evento se debe definir un manejador de eventos, un elemento que actúe de oyente sobre el control o controles que necesitamos vigilar.
- El modelo que veremos a continuación se aplica igual para AWT que para Swing

Oyentes

- Cada tipo de evento tiene asignada una interfaz
- Para definir el manejador de un evento, hay que hacer una clase que implemente la interfaz asociada al evento
- Tipos de eventos más comunes:
  - ***ActionListener***: para eventos de acción (pulsar un botón, pulsar Intro, etc)
  - ***ItemListener***: para cambios de estado de componentes
  - ***KeyListener***: para eventos de teclado (pulsar una tecla, soltarla, etc)
  - ***MouseListener***: para eventos estáticos de ratón (hacer click, ratón dentro, ratón fuera, etc)
  - ***MouseMotionListener***: para eventos dinámicos de ratón (mover el ratón, arrastrar el ratón)
  - ***WindowListener***: para eventos de ventana (cerrar ventana, redimensionarla, etc)

Modos de definir un oyente

- Ejemplo: hacer algo al pulsar un botón

1. Que la propia clase que utiliza el control implemente el oyente:

```
Class MiClase extends JFrame implements ActionListener
{
   public MiClase()
   {
        ...
        Button btn = new Button("Boton");
        btn.addActionListener(this);
        ...
   }
   public void actionPerformed(ActionEvent e)
   {
        ... // Codigo del evento del botón
   }
   ...
```

Modos de definir un oyente

2. Definir otra clase que implemente el oyente:

```
Class MiClase extends JFrame {
    public MiClase()
    {
          ...
          Button btn = new Button("Boton");
          btn.addActionListener(new MiOyente());
          ...
    }
}
Class MiOyente implements ActionListener {
    public void actionPerformed(ActionEvent e)
    {
          ... // Codigo del evento del botón
    }
}
```

Modos de definir un oyente

3. Definir una instancia interna del oyente:

```
Class MiClase extends JFrame {
    public MiClase()
    {
          ...
          Button btn = new Button(“Boton”);
          btn.addActionListener(new ActionListener()
          {
                  public void actionPerformed(ActionEvent e)
                  {
                          ... // Codigo del evento del botón
                  }
          });
          ...
    }
}
```

Uso de adapters

- Algunos *listeners* de Java tienen varios métodos para implementar (por ejemplo, *MouseListener*)
- Si sólo queremos definir una acción para uno de esos métodos, deberíamos dejar los demás vacíos, pero ocupando líneas de código y tiempo de programación:

```
JPanel p = new JPanel();
p.addMouseListener(new MouseListener()
{
   public void mouseClicked(MouseEvent e) {
        // ... Código del evento
   }
   public void mouseEntered  (MouseEvent e) { // Inutil }
   public void mouseExited   (MouseEvent e) { // Inutil }
   public void mousePressed  (MouseEvent e) { // Inutil }
   public void mouseReleased (MouseEvent e) { // Inutil }
});
```

Uso de adapters

- En lugar de eso, podemos utilizar la clase *adapter* asociada al *listener*, y sólo definir los métodos que necesitemos:

```
JPanel p = new JPanel();
p.addMouseListener(new MouseAdapter()
{
   public void mouseClicked(MouseEvent e) {
        // ... Código del evento
   }
});
```