ACTIVIDAD GUIADA 2

Contenedores. Marcos

Contenidos

Introducción

Los usuarios de ordenadores esperan sin duda que sus programas incluyan una GUI, acepten entradas del ratón y funcionen como otros programas. Aunque algunos usuarios siguen trabajando en entornos de línea de comandos , la mayoría se confundirán si no tienen una interfaz gráfica. Vamos a crear nuestra primera interfaz gráfica de usuario (GUI) con Java.

Swing y AWT

Como Java es un leguaje multiplataforma (permite crear programas para distintos S.O.), su software de interfaz gráfica de usuario debe ser flexible. En lugar de limitarse al estilo de Windows o Mac, también debe adecuarse a otras plataformas. Con Java, el desarrollo de la interfaz de usuario de un programa se basa en **Swing** y en un conjunto de clases anterior denominado **AWT** (*Abstract Windowing Toolkit*). Estas clases le permiten crear una GUI y recibir entradas del usuario, mediante elementos como los siguientes:

- Ventanas, marcos, cuadros de diálogo, paneles y ventanas de applet.
- Botones, casillas de verificación, etiquetas y otros componentes sencillos.
- Campos de texto, reguladores y otros componentes más complejos
- Menús desplegables y emergentes.

Para crear y mostrar una interfaz, debe crear objetos, establecer sus propiedades e invocar sus métodos.

Contenedores y componentes

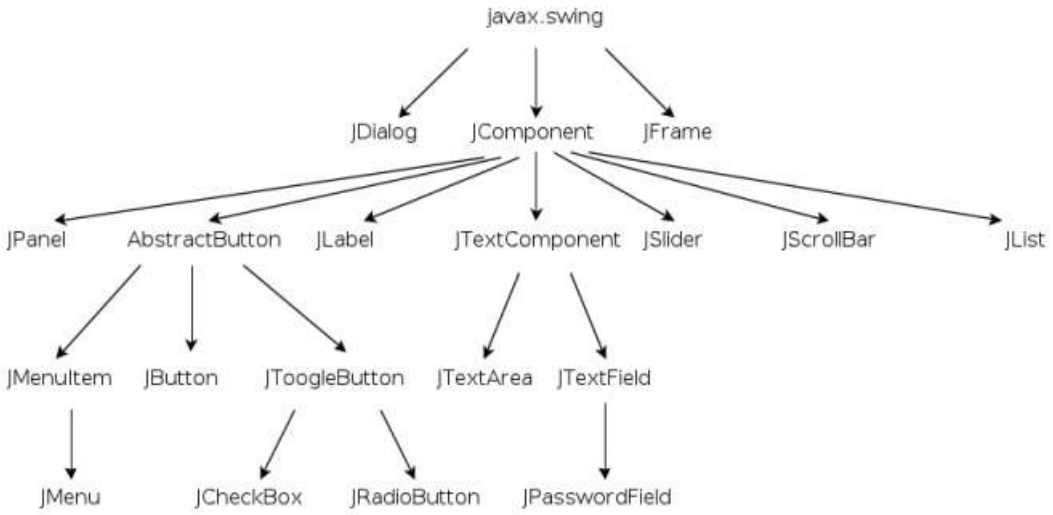
Al confeccionar una GUI, se trabaja con dos tipos de objetos: **componentes y contenedores**.

- Un **componente** es un elemento concreto de una interfaz de usuario, como un botón.
- Un **contenedor** es un componente que puede emplear para almacenar otros componentes.

El primer paso para diseñar una interfaz consiste en crear un contenedor para almacenar componentes. En una aplicación, este contenedor suele ser una ventana o un marco.

*NOTA: Se Suele utilizar la **notación húngara** para la creación de las variables que representan los elementos de la interfaz. Esto significa que el tipo de la variable forma parte del propio nombre de la misma. Por ejemplo, si tenemos una variable que tenemos pensado llamar "enviar" y representa un botón, el identificador que le daremos será `btnEnviar`. Por otra parte, si tenemos una variable que representa una caja de texto donde introduciremos un nombre llamaremos a la variable `txtNombre`.*

Este tipo de notación permite identificar fácilmente el tipo de objetos que estamos manejando.



Marcos

Los **marcos** son contenedores que se pueden mostrar en una interfaz de usuario y que albergan otros componentes. Los **marcos** incluyen las funciones de ventana habituales que los usuarios esperan al ejecutar software, como botones para cerrar, ampliar o reducir la ventana.

Este contenedor se crea con la clse **`JFrame`** de Swing. Para que el paquete de clases Swing esté disponible en un programa de Java, podemos usar la siguiente instrucción:

```
import javax.swing.*;
```

O también podemos incluir una a una las clases Java necesarias del paquete anterior. Esta aproximación no supone mejora en rendimiento en tiempo de ejecución del programa, si bien nos permite tener controladas qué clases estamos utilizando:

```
import javax.swing.JFrame;
```

Una forma de usar un marco en una aplicación de Java consiste en convertir a la aplicación en una subclase de **`JFrame`**. Su programa hereda el comportamiento que necesita para funcionar como marco. Las siguientes instrucciones crean una subclase de **`JFrame`**:

```
import javax.swing.*;

public class HolaMundoFrame extends JFrame {
public HolaMundoFrame() {
   // definir el marco
}
```

Esta clase crea un marco pero no lo configura totalmente. En el constructor, debe realizar varias acciones al crear un marco:

1. Lo primero que debe incluir el método es la **invocación de uno de los constructores** de **`JFrame`**, por medio de la instrucción:

   ```
   super();
   ```

   Esta instrucción invoca el constructor de **`JFrame`** sin argumentos. También puede invocarlo con el título del marco como argumento (que aparecerá en la barra de título del borde superior):

   ```
   super("Hola mundo");
   ```

2. Si no define el **título** de la forma anterior, puede invocar el método **`setTitle();`** del marco con una cadena como argumento:

   ```
   setTitle("Hola mundo");
   ```

3. El **tamaño del marco** se puede establecer invocando su método **`setSize()`** con dos argumentos: la anchura y la altura. La siguiente instrucción define un marco de 400 píxeles de ancho por 200 píxeles de alto:

   ```
   setSize(400,200);
   ```

   Otra forma de establecer el tamaño consiste en rellenarlo con componentes y después invocar el método **`pack()`** del marco sin argumentos:

   ```
   pack();
   ```

El método **`pack()`** establece un tamaño suficiente para albergar los componentes dentro del marco a su tamaño preferido.

4. Todo marco se muestra con un botón junto a la barra de título que sirve para **cerrarlo**. En Windows, este botón es un X en la esquina superior derecha del marco.
   Para definir lo que sucede al pulsar este botón, es necesario invocar el método **`setDefaultCloseOperation()`** del marco con una de las siguientes variables de clase **`JFrame`** como argumento:

   - **`setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`**: Salir de la aplicación cuando se cierra el marco.
   - **`setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE)`**: Cerrar el marco, borrar el objeto del marco de la memoria y mantener la aplicación ejecutándose.
   - **`setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE)`**: Mantener abierto el marco y continuar la ejecución (no hace nada).
   - **`setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE)`**: Cerrar el marco y continuar la ejecución sin liberar los recursos.

   Es importante definir este comportamiento, de lo contrario el marco seguirá ejecutándose aunque lo cerremos.

5. Una interfaz gráfica de usuario creada con Swing puede personalizar su aspecto operativo y visual por medio de un tema que controle la ubicación y el comportamiento de botones y otros componentes.

   A partir de Java 7, se presenta un aspecto operativo y visual mejorado denominado **Nimbus**, pero debe activarse para emplearlo en una clase. Para ello, se invoca el método **`setLookAndFeel()`** de la clase **`UIManager`** del paquete principal de Swing. Este método acepta un argumento: el nombre completo de la clase del aspecto operativo y visual. La siguiente instrucción establece Nimbus como aspecto operativo y visual:

```
UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
```

6. Para que el marco sea visible debe invocar su método **`setVisible()`** con **`true`**:

```
setVisible(true);
```

   De este modo se abre el marco con la anchura y altura definida. También se puede invocar con **`false`** para que el marco no se muestre.

El listado siguiente contiene el código fuente descrito anteriormente. Se ha creado una clase llamada `HolaSwingFrame`:

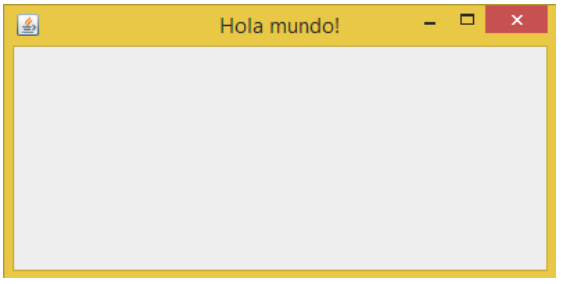
```
import javax.swing.*;

public class HolaSwingFrame extends JFrame {
    public HolaSwingFrame() {
        super("Hola mundo!");
        setLookAndFeel();
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    private void setLookAndFeel() {
        try{
      UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
        }
        catch (Exception e) {
            /* Ignoramos el error. Si no tenemos instalado Nimbus
                se mostrará el Look & Feel por defecto
            */
        }
    }
    public static void main(String[] arguments) {
        HolaSwingFrame frSaluda = new HolaSwingFrame();
    }
}
```

Se ha incluido el método **`main()`**, donde únicamente se instancia un objeto de tipo **`HolaSwingFrame`**, convirtiendo esta clase en una aplicación.

Además, se ha creado un método privado **`setLookAndFeel()`** que designa Nimbus como aspecto operativo y visual del marco a través de la clase **`UIManager`** . De este modo, al ir creando métodos, evitamos que todo el código se acumule en el constructor de la aplicación, mejorando así la legibilidad y la reutilización del mismo.

La invocación de este método se realiza en un bloque **`try-catch`**, que permite corregir los errores que se produzcan. En este caso, si se produce un error, se conserva el aspecto operativo y visual predeterminado en lugar de Nimbus.

Al ejecutar la clase, se verá el marco siguiente:



Lo único que muestra **`HolaSwingFrame`** es un título, el saludo **"Hola mundo!"**. El marco es una ventana vacía ya que todavía no tiene componentes. Parece mucho trabajo para tan poca cosa, pero al menos es un comienzo.

Para añadir componentes a un marco, debe crearlos y añadirlos al contenedor. Cada contenedor tiene un método **`add()`** que acepta como argumento el componente que se va a mostrar. Esto se verá en las siguientes actividades.