Object Oriented Software Design I

An overview of the Swing Library

Giuseppe Lipari
http://retis.sssup.it/~lipari

Scuola Superiore Sant'Anna – Pisa

November 20, 2011

Outline

1. Why swing
2. Containers
3. Buttons

Outline

1 Why swing

2 Containers

3 Buttons

The JFC/Swing library

- The JFC/Swing library provides a framework for building GUIs in Java
  - JFC stands for Java Foundation Classes
- Like all Java, Swing is portable across every platform
  - It provides classes for drawing windows, button, scrollbars, etc.
  - It provides ways to interact with the GUIs through the keyboard and the mouse
  - It is built on top of the AWT library, which provides a basic framework for low-level drawing on the screen
- Pros
  - It's portable and it's distributed by Oracle together with the JRE and JDK
- Cons
  - Slow, and somewhat limited

Swing framework

- Many aspects of the Swing framework are built according to well-known design patterns
- Our goal is not to learn Swing, but rather to learn a little bit of it so to identify the patterns that we have studied till now
- A more complete introduction can be found in the Swing Tutorial

Hello World in Swing

HelloWorld.java

```
import javax.swing.*;

public class HelloWorld {
    // Create the GUI and show it.
    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("HelloWorld");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Add the ubiquitous "Hello World" label.
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);

        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        createAndShowGUI();
    }
}
```

Outline

1 Why swing

2 Containers

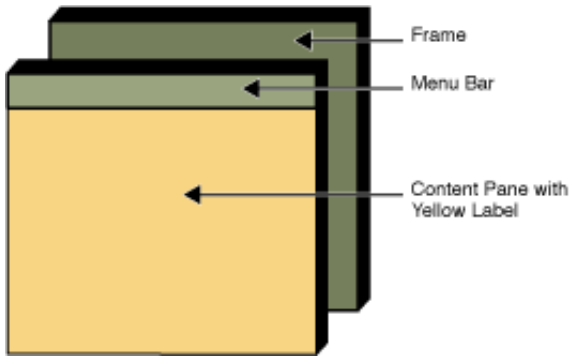
3 Buttons

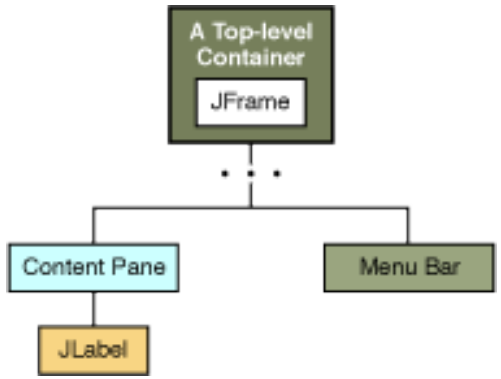
Top containers

- Inside a window, all widget are organised into “components” organised into a hierarchy
- To appear onscreen, every GUI component must be part of a containment hierarchy. A containment hierarchy is a tree of components that has a top-level container as its root.
- Each top-level container has a content pane that, generally speaking, contains (directly or indirectly) the visible components in that top-level container’s GUI.
- You can optionally add a menu bar to a top-level container. The menu bar is by convention positioned within the top-level container, but outside the content pane

Containers

- The top level container (or Frame) can be one of
  - JFrame, JDialog, JApplet
  - An example:
    `./examples/14.swing-examples/TopLevelDemo.java`
  - Here is the containement hierarchy:





Panels

- The content panel is of type `JPanel`
- It is possible to substitute the default content panel as follows:

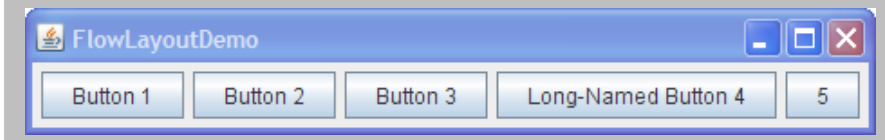
```
//Create a panel and add components to it.
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.setBorder(someBorder);
contentPane.add(someComponent, BorderLayout.CENTER);
contentPane.add(anotherComponent, BorderLayout.PAGE_END);

topLevelContainer.setContentPane(contentPane);
```

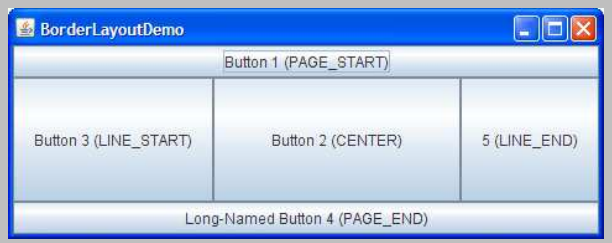
Layout

- Every panel has a layout that manages how the internal widgets are organised in the available space

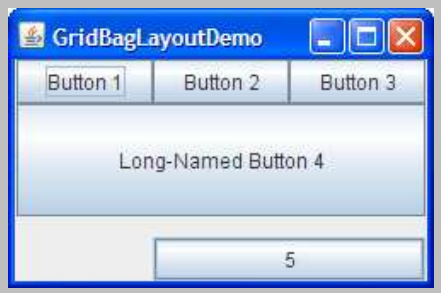
FlowLayout (default): all components in a line from left to right



Border Layout: five components in the position shown in the figure



Grid Bag Layout: all components in a grid, a component can span one or more consecutive cells



JComponent

- To add components to the panel, we use the method `add()`, which takes a `JComponent`
- All widgets (with the exception of top level containers) derive from the `JComponent` class
  - For example, `JPanel`, `JScrollPane`, `JButton`, `JTable`, `JTextArea`, etc.
- Exercise:
  - draw a UML class diagram with the class hierarchy, including `JFrame`, `JComponent`, `JPanel`, `JButton`
  - Observe the diagram: which pattern has been applied?

Outline

1. Why swing
2. Containers
3. Buttons

A button example

- Let us see how it is possible to process commands, for example clicks on buttons
- See `./examples/14.swing-examples/SwingButtonExample.java`
- **Exercise** draw the containment hierarchy for this example
- Observe the way clicks on `JButton b1` are handled:
  - The class that wants to be informed of mouse clicks on `b1` must implement interface `ActionListener`

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

  - The `ActionEvent` class describes the event type (rarely used)
  - First, the `ActionListener` object is registered through the `addActionListener()` function of `JButton`
  - When the button is clicked, method `actionPerformed()` of the registered object is called
- To install many buttons, we can create anonymous classes that just handle the event

Exercise

- Draw the UML class diagram, including JButtons, interface `ActionListener`, the Application, the anonymous classes
- Name the pattern that had been used