A Quick Java Swing Tutorial

1

Introduction

- Swing – A set of GUI classes
  - Part of the Java's standard library
  - Much better than the previous library: AWT
    - Abstract Window Toolkit
- Highlights
  - A rich set of widgets
    - Widget: Any GUI element (also called: components)
  - Contents and shape are separated (MVC support)
  - Fine-grained control over the behavior and look and feel
  - Platform independent
    - Isolates the programmer from the operating system's GUI

2

Swing Components

- Containers
  - Contain and manage other components.
  - Top Level/Internal
  - Examples: `JFrame` (Top Level), `JScrollPane`, `JPanel`.
- Basic controls
  - Atomic components
  - Used for showing ouput and/or getting some input
  - Inherits JComponent
  - Examples: `JButton, JLabel, JTextArea, JTable, Jlist`
- Usually every Swing class extends the corresponding AWT class
  - For backward-compatibility reasons

3

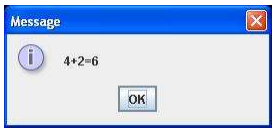
My First Swing Program

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

public class First {
  public static void main(String[] args) {
    JFrame frame = new JFrame("My First Frame");

    // operation to do when the window is closed.
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.getContentPane().setLayout(new BorderLayout());
    frame.getContentPane().add(new JLabel("I Love Swing"),
       BorderLayout.CENTER);
    frame.pack();
    frame.setVisible(true);
  }
}
```

4

Top Level Containers: JDialog

- **`javax.swing.JDialog`**:
  - More simple and limited than frames
  - Typically used for showing a short message on the screen
  - Also has a border and a title bar
  - May have an owner
    - If the owner is invisible the dialog will also be invisible
  - Use the static method of JoptionPane to show standard dialog boxes:
    `JOptionPane.showMessageDialog(null, "4+2=6");`



5

Top Level Containers: JFileChooser

- **`javax.swing.JFileChooser`**:
  - Allows the the user to choose a file
  - Supports "open" and "save": `showOpenDialog()`, `showSaveDialog()`

```
JFileChooser fc = new JFileChooser();
int returnVal = fc.showOpenDialog(null);
if(returnVal == JFileChooser.APPROVE_OPTION)
   System.out.println("File: " + fc.getSelectedFile());
```

6

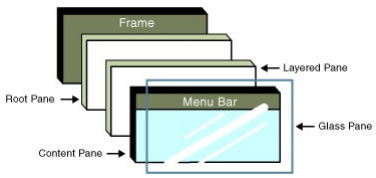
Top Level Containers: JFrame

- **`javax.swing.JFrame`**:
  - Top-level window with a title and a border.
  - Usually used as a program's main window

7

More on JFrame

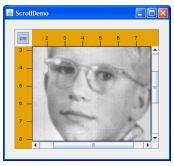
- Made of several layers
- Widgets are added to the Content Pane layer.
  - Use `getContentPane()` to obtain it
- Other layers are used for customizing the window's appearence

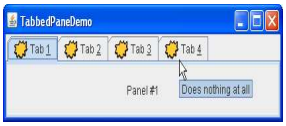


8

Internal Containers

- Not Top level containers
- Can contain other non-top level components
- Examples:
  - `JScrollPane`: Provides a scrollable view of its components
  - `JSplitPane`: Separates two components
  - `JTabbedPane`: User chooses which component to see





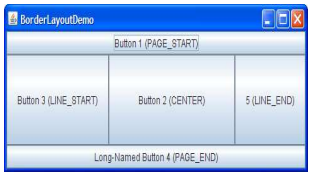
9

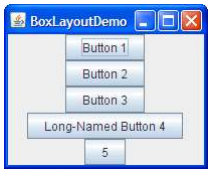
Containers - Layout

- Each container has a layout manager
  - Determines the size, location of contained widgets.
- Setting the current layout of a container:
  `void setLayout(LayoutManager lm)`
- *`LayoutManager`* implementing classes:
  - `BorderLayout`
  - `BoxLayout`
  - `FlowLayout`
  - `GridLayout`

10

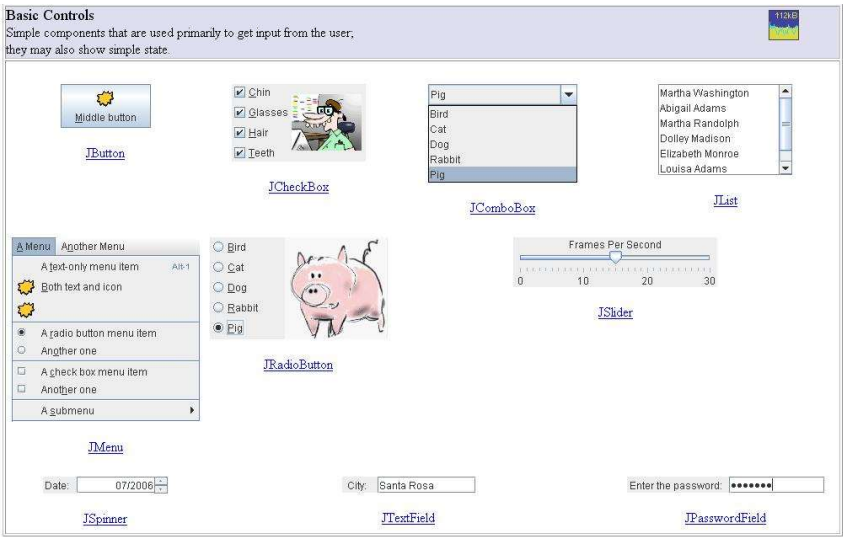
Containers - Layout





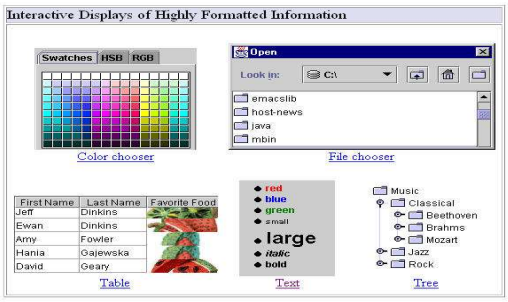
11

Swing Components



12

Swing Components



First Swing Program Revisited

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

public class First {
  public static void main(String[] args) {
    JFrame frame = new JFrame("My First Fra

    // operation to do when the window is cl
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.getContentPane().setLayout(new BorderLayout());
    frame.getContentPane().add(new JLabel("I Love Swing"),
       BorderLayout.CENTER);
    frame.pack();
    frame.setVisible(true);
  }
}
```

Create a frame

Choose the border layout

Create a text label

Add the label to the content pane

Specify CENTER as the layout position

Interface

- Interface – similar to Abstract class
- No methods are implemented
- Syntax:

```
public interface MyInterface{
    public String name="hello";
    public void sayHello();
}
```

Implementing an Interface

- Before using an interface, it needs to be implemented
- Why do we need interfaces?
  - A contract that guarantees that listed methods will be implemented
  - Helps to 'generalize' code
- Syntax:

```
public class MyInterfaceImpl implements
MyInterface{
    public void sayHello(){…}
}
```

Interface Instances

- Once interface is implemented, you can create instances of it:

```
MyInterfaceImpl myInterface = new MyInterfaceImpl();
myInterface.sayHello();
```

- Why not use abstract classes instead?
  - In Java, a subclass can have only one parent class
  - A class can implement multiple interfaces

Shortcut

- Interfaces can be implemented locally:

```
MyInterface interface = new MyInterface(){
   public void sayHello(){…}
}
interface.sayHello();
```