2021/2022(1)
IF184301 Object Oriented Programming

Lecture #8a

Graphical User Interface (GUI)

Misbakhul Munir IRFAN SUBAKTI
司馬伊凡
Мисбакхул Мунир Ирфан Субакти

Java API

- `java.awt` – Original GUI toolkit in JDK 1.1, implemented using native GUI libraries of the operating systems
  - AWT: Abstract Window Toolkit
  - Problem: portability
- Java Foundation Classes (JFC) → Swing
  - Something WINdowing Graphics  … kidding, Swing is not an acronym.
  - Represents the collaborative choice of its designers when the project was kicked off in late 1996
- `javax.swing`. Portable GUI toolkit added in Java 2, extending `java.awt`
  - Gives platform-independent operation, though it is slower
- Many other third party GUI toolkits, e.g., Standard Widget Toolkit (SWT) in Eclipse

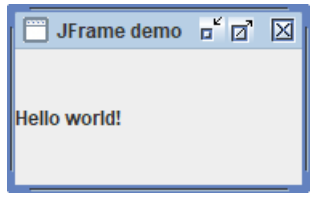
JFrame

- In many application programs one needs to import both `java.awt` and `javax.swing`
- To avoid confusion, the common class names in Swing are prefixed with the letter `J`, e.g., `JFrame`, `JApplet`, `JButton`, `JMenubar`, etc.
- *Frame* – the Java term for a GUI window

JFrame: Creating

MyJFrame.java

```
1  import java.awt.*; // BorderLayout, etc.
2  import javax.swing.*; // JFrame, JLabel, etc.
3  public class MyJFrame {
4      public static void main(String[] args) {
5          // 1. Optional: Specify who draws the window decorations
6          JFrame.setDefaultLookAndFeelDecorated(true);
7          // 2. Create the frame
8          JFrame frame = new JFrame("JFrame demo");
9          // 3. Optional: What happens when the frame closes?
10         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11         // 4. Create components and put them in the frame, e.g., a label
12         JLabel label = new JLabel("Hello world!");
13         frame.getContentPane().add(label, BorderLayout.CENTER);
14         // 5. Size the frame
15         frame.pack();
16         // 6. Show it
17         frame.setVisible(true);
18     }
19 }
```



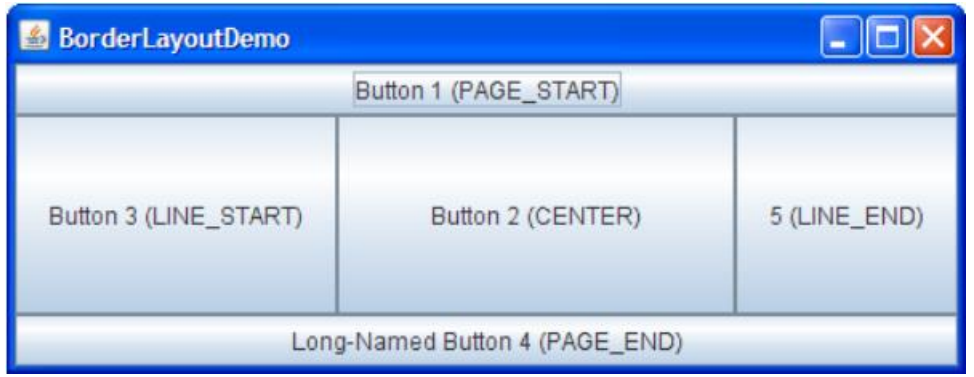
JFrame: Creating – commentary

1. Asks that the frame should be decorated with borders etc.
2. Creates an instance of the `JFrame` class
3. Specifies that the program should exit when the frame is closed.
   - If we don't put this, the program might hang
4. Adds something called `label` as a GUI component to the frame
5. Asks that the frame be automatically sized based on its contents and their preferred sizes
6. Make the frame visible
   - Otherwise, it would be hidden by default

Components and containers

- The things that we can display in GUI windows are called components
  - E.g., buttons, text-fields, tables, etc.
- Some of the components are containers
  - They can contain other components inside them – which might again be containers, etc.
- A JFrame is a container of a specialised sort
  - It has several panes as its components, the important one being its *contentPane*

contentPane



- The `contentPane` is a container – so, we can add components to it

MyContentPane.java

```
1 import java.awt.*; // BorderLayout, Container, etc.
2 import javax.swing.*; // JFrame, JLabel, etc.
3 public class MyContentPane {
4     public static void main(String[] args) {
5         JFrame.setDefaultLookAndFeelDecorated(true);
6         JFrame frame = new JFrame("JFrame demo");
7         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8         Container pane = frame.getContentPane();
9         JLabel labelOnePiece = new JLabel("One piece world");
10        labelOnePiece.setIcon(new ImageIcon("res/one_piece_world.gif"));
11        pane.add(labelOnePiece, BorderLayout.PAGE_START);
12        pane.add(new JLabel("Hello world!"));
13        pane.add(new JButton("Click me"), BorderLayout.PAGE_END);
14        frame.pack();
15        frame.setVisible(true);
16    }
17 }
```

- Or, we can create a new `contentPane`:

MyContentPane2.java

```
1 import java.awt.*; // BorderLayout, etc.
2 import javax.swing.*; // JFrame, JLabel, etc.
3 public class MyContentPane2 {
4     public static void main(String[] args) {
5         JFrame.setDefaultLookAndFeelDecorated(true);
6         JFrame frame = new JFrame("JFrame demo");
7         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8         JPanel pane = new JPanel(new BorderLayout());
9         JLabel labelOnePiece = new JLabel("One piece world");
10        labelOnePiece.setIcon(new ImageIcon("res/one_piece_world.gif"));
11        pane.add(labelOnePiece, BorderLayout.PAGE_START);
12        pane.add(new JLabel("Hello world!"));
13        pane.add(new JButton("Click me"), BorderLayout.PAGE_END);
14        frame.setContentPane(pane);
15        frame.pack();
16        frame.setVisible(true);
17    }
18 }
```

JPanel and JApplet

- *Panels* are simple lightweight containers
- They support graphics, in addition to user interface components
- *Applets* are frames that can be drawn inside web pages, included using a HTML tag such as

MyApplet.html

```
<HTML>
    <HEAD>
        <TITLE>Applet demo</TITLE>
    </HEAD>
    <BODY>
        <APPLET code="MyContentPane.class" width="400" height="300">
        Applet demo - only for Java-enable browser!
        </APPLET>
    </BODY>
</HTML>
```

- But, it's has been deprecated in HTML 4.0

GUI: the components

- Display elements: text-label, image-label
- Basic controls: text-field, button, radio-button, check-box, option-list, combo-box (drop-down list), menu, slider
- Containers: panel, scroll-pane, split-pane, tabbed-pane, toolbar
- Fancy displays: table, tree-display
- Fancy controls: file-chooser, colour-chooser

Inheritance: Defining panels

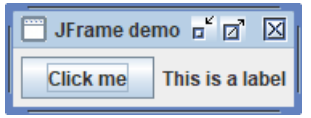
- Good design practice to define panels as classes – extending the JPanel class

MyPanel.java

```
1  import javax.swing.*;
2  public class MyPanel extends JPanel {
3      int count = 0; // The state
4      JButton button = new JButton("Click me");
5      JLabel label = new JLabel("This is a label");
6      public MyPanel() {
7          add(button);
8          add(label);
9      }
10 }
```

MyPanelTest.java

```
1  import javax.swing.*;
2  public class MyPanelTest {
3      public static void main(String[] args) {
4          JFrame.setDefaultLookAndFeelDecorated(true);
5          JFrame frame = new JFrame("JFrame demo");
6          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7          MyPanel panel = new MyPanel();
8          frame.setContentPane(panel);
9          frame.pack();
10         frame.setVisible(true);
11     }
12 }
```



- It is even necessary when we need to include graphics painting

Panels: Input

- Users enter text in text-field components
- Reading the value in a text-field involves four changes to what we have seen:

MyPanel2.java

```
1  import javax.swing.*;
2  import java.awt.event.*; // 1
3  public class MyPanel2 extends JPanel
4      implements ActionListener { // 2
5      JTextField t = new JTextField(15);
6      public MyPanel2() {
7          // ...
8          add(t);
9          t.addActionListener(this); // 3
10     }
11     public void actionPerformed(ActionEvent e) {
12         // ...
13         t.getText(); // 4
14         // ...
15     }
16 }
```

Panels: Input (continued)

- `import java.awt.event.*` must appear verbatim (the event-handling library)
- `implements ActionListener` must appear verbatim (to say that it handles action events)

  `JTextField` variable, say `t`, is declared as instance variable
- In the constructor, in addition to calling `add`, call `t.addActionListener(this);` to say that this object will handle the action events from `t`
- Define `actionPerformed` method
  - Called when user press the enter/return key in the text-field
  - Use `getText` method of `JTextField` class to get the `String` contained in `t`

MyPanel2.java

```
1  import javax.swing.*;
2  import java.awt.event.*; // 1
3  public class MyPanel2 extends JPanel
4      implements ActionListener { // 2
5      JTextField t = new JTextField(15);
6      public MyPanel2() {
7          // ...
8          add(t);
9          t.addActionListener(this); // 3
10     }
11     public void actionPerformed(ActionEvent e) {
12         // ...
13         t.getText(); // 4
14         // ...
15     }
16 }
```

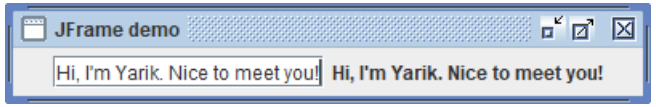
Panels: Input (continued)

MyPanel3.java

```
1  import javax.swing.*;
2  import java.awt.event.*; // 1
3  public class MyPanel3 extends JPanel
4      implements ActionListener {
5      JTextField t = new JTextField(15);
6      JLabel l;
7      public MyPanel3() {
8          add(t);
9          t.addActionListener(this);
10         l= new JLabel("Write something on text-field & enter");
11         add(l);
12     }
13     public void actionPerformed(ActionEvent e) {
14         l.setText(t.getText());
15     }
16 }
```

MyPanel3Test.java

```
1  import javax.swing.*;
2  public class MyPanel3Test {
3      public static void main(String[] args) {
4          JFrame.setDefaultLookAndFeelDecorated(true);
5          JFrame frame = new JFrame("JFrame demo");
6          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7          MyPanel3 panel = new MyPanel3();
8          frame.setContentPane(panel);
9          frame.pack();
10         frame.setVisible(true);
11     }
12 }
```



The event model

- The code above is one instance of the event model
- With the event model, the panel can respond to button clicks, mouse movements, etc.
- Can also use multiple text-fields and distinguish which one was modified

Layout

- Layout of GUI panels is a tricky business
  - Need to allow for resizing of windows
  - Need to be flexible about the text-labels, etc.
- Java library defines a number of *Layout Managers* to facilitate layout
- Or , we can define our own or import third party libraries