5COSC001W - OBJECT ORIENTED PROGRAMMING
Lecture 6: Graphical User Interfaces using Swing

Dr Dimitris C. Dracopoulos

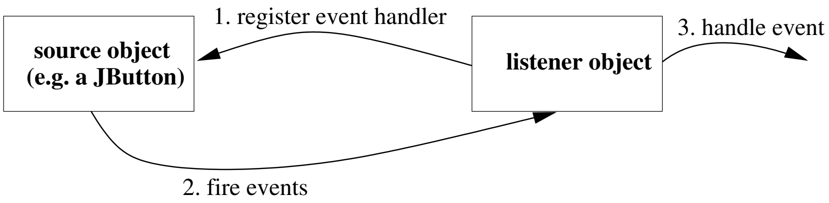
AWT (Abstract Window Toolkit)

- Create portable GUIs (run in every platform without the need of recompilation).
- The native library behind the Java run time library is specific for each platform. Differences can be seen by running the same code in different platforms.
- Event driven mode. The GUI generates events through its interaction with the user (e.g. pressing a button). The GUI passes the generated events to code which can take appropriate action.

Event Handling

Event programming is done using the following steps:

- A main loop waits some user input.
- When the user clicks the mouse, an event is generated.
- The event is passed to some user written method of a listener object (the event handler) which is registered to handle the specific type of event (e.g. clicking a Swing button) and "listens" for such events.



Swing vs AWT

The main advantage of Swing (which is part of JFC-the Java Foundation Classes) over AWT is:

- Unlike AWT, Swing is not using native libraries and thus running the same code in different platforms does not reveal the differences in the native libraries.

JFC includes among other things:

- A pluggable look and feel which make application look in the same way whether they run in Windows, Unix, Mac or another operating system.
- The Java 2D API.
- The Swing components. These replace many of the components that AWT offer (scrollbars, buttons, labels, textfields, etc.)
- A drag and drop library.

Swing uses the same event driven model as AWT.

Example:

The simplest Swing program which just creates a new frame on the screen is:

```
import javax.swing.*;

public class SimpleSwingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("SimpleSwingExample");
        frame.setSize(400, 400);
        frame.setVisible(true);
    }
}
```

Adding an event to Swing code

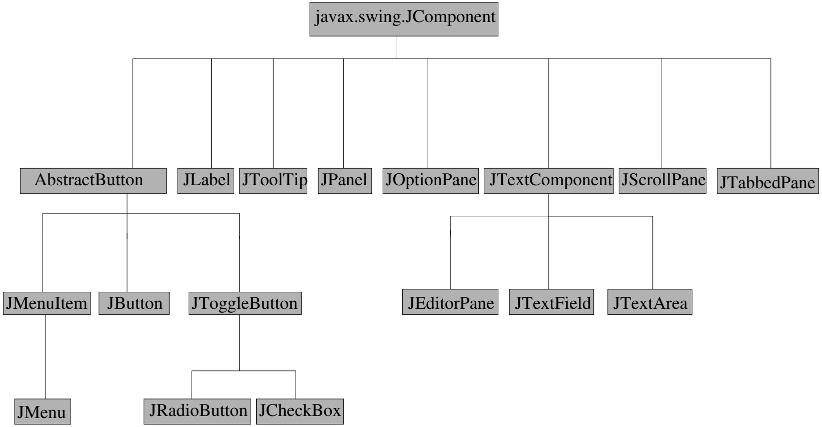
```java
import javax.swing.*;
import java.awt.event.*;

// window event Handler class
class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.out.println("Closing window!");
        System.exit(0);
    }
}

public class SimpleSwingExampleWithEvent {
    public static void main(String[] args) {
        JFrame frame = new JFrame("SimpleSwingExample");

        // register an event handler for frame events
        frame.addWindowListener(new MyWindowListener());
        // frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setSize(400, 400);
        frame.setVisible(true);
    }
}
```

Swing Components (JComponents)

All Swing components are subclasses of `JComponent` which provides the following features among others:

- size
- double buffering
- support for accessibility and internationalisation
- tooltips (pop-up help when the cursor is placed over a component)
- support for keyboard control instead of the mouse

Swing Components (JComponents) — cont'ed



Adding Components to a Container

To add a component within a container (e.g. `JFrame`, `JDialog` or `JApplet`):

1. The component must be added to the *content pane* of the container:

   ```
   MyContainer.getContentPane().add(myComponent);
   ```

2. Register the event handler with the component using a method `addXXXListener`. Events of `XXX` type will execute the corresponding code of the event handler.

Example:

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

// window event Handler class
class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.out.println("Closing window!");
        System.exit(0);
    }
}
```

```java
// button event handler class
class MyActionListener implements ActionListener {
    private int i=1;
    JFrame frame;
    MyActionListener(JFrame f) {
        frame = f;
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Pressed Button " + i++ + "th time!");

        if (i % 2 == 0)
            frame.getContentPane().setBackground(Color.red);
        else
            frame.getContentPane().setBackground(Color.white);
    }
}
```

```java
public class ComponentExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("ComponentExample");
        JButton button = new JButton("press me");
        JPanel jp = new JPanel();
        jp.setBackground(Color.white);

        // set the content pane to be the newly created JPanel
        frame.setContentPane(jp);

        frame.getContentPane().add(button);

        // register an event handler for frame events
        frame.addWindowListener(new MyWindowListener());

        // register an event handler for button events
        button.addActionListener(new MyActionListener(frame));

        frame.setSize(400, 400);
        frame.setVisible(true);
    }
}
```

Layout Managers - What goes where

Layout managers specify how the various components will be placed in a container.
The following are some of the layout managers:

- *FlowLayout*: components are added from left to right, centered in the container. A new line is started when it is necessary.
  This is the default layout for the `JPanel` objects.
- *GridLayout*: components are added in a $m \times n$ grid.
  For example:

```
new GridLayout(4, 2);  // a grid with 4 rows, 2 columns
```

Layout Managers - What goes where (cont'ed)

- *BorderLayout*: the default layout for `JFrame`. Four components can be placed in the four edges of the container ("North", "South", "East", "West") with a fifth component occupying the remaining space in the middle ("Center"). For example:

  ```
  // add myComponent in the right hand edge
  frame.getContentPane().add(myComponent, "East");
  ```

  This is the default layout for the content panes.
- *BoxLayout*: components can be grouped in the x or y direction.
  For example:

  ```
  JFrame frame = new JFrame("Frame Title");
  Container c = frame.getContentPane();
  // place components in the y-direction, i.e. top to bottom
  c.setLayout(new BoxLayout(c, BoxLayout.Y_AXIS));
  ```

Layout Managers - What goes where (cont'ed)

- *CardLayout*: Implement an area that contains different components (cards) at different times. Which component is displayed at each point in time, is often controlled by a combo box.
- *GridBagLayout*: the most flexible and complex layout manager. A GridBagLayout places components in a grid of rows and columns, allowing specified components to span multiple rows or columns.
  Not all rows necessarily have the same height. Similarly, not all columns necessarily have the same width. Constraints for the various components are specified using a `GridBagConstraints` object.

JPanel

`JPanel` is a generic container which must be always placed within another container (e.g. a `JFrame`).

- A typical usage of `JPanel` is to group related controls (components) together, which can then be treated as a single unit by manipulating (e.g. placing) the panel.

Creating Applications with Professional Look

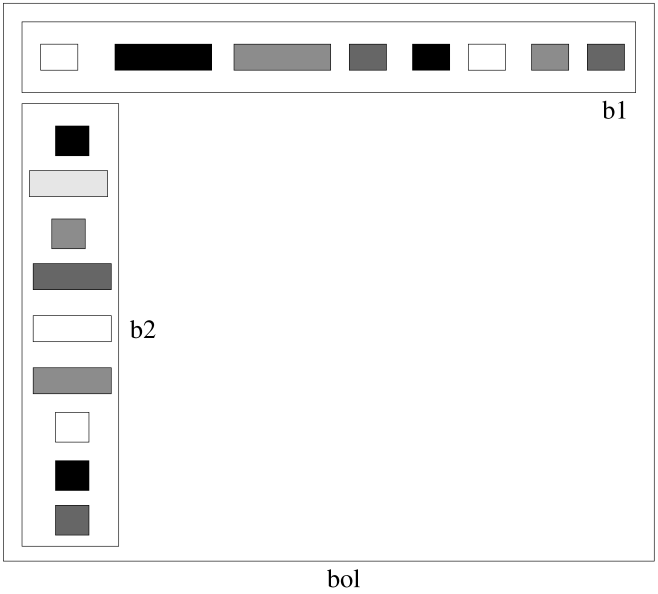
No existing layout manager is adequate to create professionally looking applications.
The solution to this is to:

- combine different layout managers together.

For example:

1. A series of components can be placed in a `JPanel` container `b1` with `BoxLayout` (x direction) as the layout manager.
2. `b1` itself can be placed in the "North" edge of a `BorderLayout bol` based container.
3. A series of components can be placed in a `JPanel` container `b2`, having `BoxLayout` (y direction) layout.
4. The new `JPanel b2` can be placed in the left ("West") edge of the `BorderLayout bol`.
5. ...

Creating Applications with Professional Look (cont'ed)



JLabel and JTextField

- *JLabel:* Display a fixed string and/or an image.
- *JTextField:* An area where a single line of text can be entered by the user.

Example:

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

// window event Handler class
class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.out.println("Closing window!");
        System.exit(0);
    }
}

// textfield event handler class
class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("You entered: " + e.getActionCommand())
    }
}
```

Example:

```
public class LabelFieldExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JLabel and JTextField Example")

        JLabel label = new JLabel("Enter your name: ");
        // create a field with 25 chars width
        JTextField field = new JTextField(25);

        // put components next to each other in the x-direction
        Container c = frame.getContentPane();
        c.setLayout(new BoxLayout(c, BoxLayout.X_AXIS));

        // add label and field in the frame
        c.add(label);
        c.add(field);

        // register an event handler for frame events
        frame.addWindowListener(new MyWindowListener());

        // register an event handler for button events
        field.addActionListener(new MyActionListener());

        frame.pack();
        frame.setVisible(true);
    }
}
```

JCheckBox and JRadioButton/ButtonGroup

- *JCheckBox*: A box which can be selected or not selected to indicate a choice of the user.
- *JRadioButton*: A series of checkboxes, out of which only one can be selected.
- *ButtonGroup*: Used to group together a number of `JRadioButtons`.

Example:

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

// window event Handler class
class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.out.println("Closing window!");
        System.exit(0);
    }
}

// radio event handler class
class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Selected: " + e.getActionCommand());
    }
}
```

```java
// checkboxes event handler class
class MyCheckBoxListener implements ItemListener {
    public void itemStateChanged(ItemEvent e) {
        JCheckBox chk = (JCheckBox) e.getItem();
        String label = chk.getText();

        if (e.getStateChange() == e.SELECTED)
            System.out.println(label + " selected");
        else
            System.out.println(label + " de-selected");
    }
}
```

```
public class ButtonGroupExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Radio Buttons and " + "
                                     CheckBoxes Example");

        // create a JPanel to hold the checkboxes
        JPanel topPanel = new JPanel();
        // JPanel has BoxLayout in x-direction
        topPanel.setLayout(new BoxLayout(topPanel,
                                          BoxLayout.X_AXIS));

        JCheckBox chk1 = new JCheckBox("GPS");
        JCheckBox chk2 = new JCheckBox("Alloys");
        JCheckBox chk3 = new JCheckBox("Power Steering");
        JCheckBox chk4 = new JCheckBox("Convertible");
        JLabel label = new JLabel("Extras: ");

        // add label and checkboxes in JPanel;
        topPanel.add(label);
        topPanel.add(chk1);
        topPanel.add(chk2);
        topPanel.add(chk3);
        topPanel.add(chk4);
```

```
// create a JPanel to hold the checkboxes
JPanel leftPanel = new JPanel();
// JPanel has BoxLayout in x-direction
leftPanel.setLayout(new BoxLayout(leftPanel,
                                  BoxLayout.Y_AXIS));

JRadioButton rd1 = new JRadioButton("CD Player");
JRadioButton rd2 = new JRadioButton("DVD Player");
JRadioButton rd3 = new JRadioButton("Cassette Player");


// group radio buttons together
ButtonGroup group = new ButtonGroup();
group.add(rd1);
group.add(rd2);
group.add(rd3);

// add radio buttons in the JPanel
leftPanel.add(rd1);
leftPanel.add(rd2);
leftPanel.add(rd3);

// add panels in the frame
frame.getContentPane().add(topPanel, "North");
frame.getContentPane().add(leftPanel, "West");
```

```
        // register an event handler for frame events
        frame.addWindowListener(new MyWindowListener());

        // register an event handler for checkboxes
        MyCheckBoxListener chkListener = new MyCheckBoxListener();
        chk1.addItemListener(chkListener);
        chk2.addItemListener(chkListener);
        chk3.addItemListener(chkListener);
        chk4.addItemListener(chkListener);

        // register an event handler for radio buttons
        ActionListener radioListener = new MyActionListener();
        rd1.addActionListener(radioListener);
        rd2.addActionListener(radioListener);
        rd3.addActionListener(radioListener);

        frame.setSize(400, 400);
        //frame.pack();
        frame.setVisible(true);
    }
}
```

Adding Scrollbars to a Component

The JScrollPane class provides a scrollable view of any lightweight component.

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

public class ScrollExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JScrollPane Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel label = new JLabel(new ImageIcon("peppers.png"));

        JScrollPane jsp = new JScrollPane(label);

        // add label in the frame
        frame.getContentPane().add(jsp);

        frame.setSize(200, 200);
        frame.setVisible(true);
    }
}
```

Other JComponents

- *JTextArea*: A multi-line area in which plain text can be entered.
- *JEditorPane*: An area which can be used to display and edit formatted text. Among other formats, it supports HTML and RTF.
- *JOptionPane*: A pane used to pop up warning and error messages.
- *JTabbedPane*: A series of tabs associated with components on a large rectangular area. Selecting a different tab, displays the associated component. This is used to save screen space.

Creating Tables in Swing — JTable

A JTable can be created using one of 2 constructors, the simplest one is the one accepting two arguments:

1. A two dimensional array, containing the data of the table.
2. A one dimensional array, containing the column names.

```
JTable(Object[][] rowData, Object[] columnNames)
```

- You can place a JTable in a JScrollPane which is then placed in a container.
- if you don't place a table in a scrollpane then you must get the table header component and place it in the container:

  ```
  container.setLayout(new BorderLayout());
  container.add(table.getTableHeader(),BorderLayout.PAGE_START);
  container.add(table, BorderLayout.CENTER);
  ```

Example:

```
import javax.swing.*;

public class JTableExample extends JPanel {
    public JTableExample() {
        String[] columnNames = {"First Name",
                                "Last Name",
                                "Position",
                                "Age",
                                "Salary"};

        Object[][] data = {
            {"John", "Smith", "Manager",
              new Integer(35), new Integer(40000)},
            {"Tom", "Bubble", "Developer",
              new Integer(22), new Integer(22000)},
            {"Helen", "Hitchcock", "Project Leader",
              new Integer(30), new Integer(34000)},
            {"Kate", "Silva", "Receptionist",
              new Integer(20), new Integer(18000)},
            {"Susie", "White", "Developer",
              new Integer(25), new Integer(25000)}
        };
        final JTable table = new JTable(data, columnNames);
        table.setFillsViewportHeight(true);


        //Create the scroll pane and add the table to it.
        JScrollPane scrollPane = new JScrollPane(table);

        //Add the scroll pane to this panel.
        add(scrollPane);
    }

    private static void createAndShowGUI() {
        // Create the frame window
        JFrame frame = new JFrame("JTableExample");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create and set up the content pane.
        JTableExample newContentPane = new JTableExample();
        newContentPane.setOpaque(true); // content panes must be o
        frame.setContentPane(newContentPane);

        // Display the window.
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        createAndShowGUI();
    }
}
```

The Event Dispatching Thread

Updates to the graphical components should ONLY be done inside the event dispatching thread!

- Methods of event handlers such as `ActionListener.actionperformed()`, etc. are executed in the event dispatching thread.
- If you attempt to update the graphical components outside the event dispatching thread, they might not be updated and you will get inconsistent results!

Solution: If you would like to run code (which is located outside the event listener methods) as part of the event dispatching thread use:

```
javax.swing.SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        // ... code to run in the event dispatching thread
    }
});
```