CSEN401 – Computer Programming Lab

Topics:

Graphical User Interface
Window Interfaces using Swing

Prof. Dr. Slim Abdennadher

22.3.2015

Swing

AWT versus Swing

- Two basic sets of components to implement a **graphical user Interface (GUI)**:
  - **Abstract Window Toolkit** (AWT)
  - **Swing**
- **Swing** can be viewed as an improved version of AWT.
- Swing implements a set of GUI components that **build on** AWT technology.
- We will use classes from both Swing and AWT.

Event-Driven Programming

- A **widget** (component of a window) can get an event and can call the corresponding processing program.
- **Main implementation issues**:
  - How to pass an event to widget?
  - How to specify the event-processing program that will be called?
  - How the event processing program can get details about the event that called it?
- GUIs are **event driven**:
  - Generate events when user interacts with GUI, e.g. mouse click, mouse movement, typing in a text field, ...
  - Event information stored in object that extends `AWTEvent`.

Creating a Simple Window

```
import javax.swing.*;
public class FirstWindow {
     public static void main(String[] args) {
          JFrame myWindow = new JFrame();
          myWindow.setSize(400,100);
          myWindow.setVisible(true); }
}
```

- `import` says that the program uses the **Swing library**.
- The object `myWindow` is an object of the class `JFrame`.
- A `JFrame` swing is a very simple window but with a set of features, e.g. close-window button, ...
- `setSize` is a method of the class `JFrame` and sets the size of the window.
- `setVisible` makes the window visible on the scree.

Adding Text to the Window

```
import javax.swing.*;
public class FirstWindow {
     public static void main(String[] args) {
        JFrame myWindow = new JFrame();
        myWindow.setSize(400,100);
        JLabel myLabel = new JLabel("My first window");
        myWindow.getContentPane().add(myLabel);
        myWindow.setVisible(true); } }
```

- The object `myLabel` is an object of the class `JLabel`.
- `Jlabel` is a special kind of text that can be added to a `JFrame` or to any of a number of other kinds of objects.
- `getContent` is a method of the class `JFrame` that add produces the **content pane** of the `JFrame`.
- Every `JFrame` has its **content pane**, e.g. inside of the `JFrame`.
- Using the `add` method, the label `myLabel` is added to the content pane of `myWindow`.

Event Firing and Event Listener

- What should happen when the user clicks the close-window button?
- The window should **fire** an event and send it to a **listener object**.
- A **listener object** should have methods that should specify what should happen when events of various kinds are sent to the listener.
- These methods are called **event handlers**.
- The programmer has to **define** or **redefine** these event-handler methods.

Adding Events to the Window

```
import javax.swing.*;
public class FirstWindow {
     public static void main(String[] args) {
        JFrame myWindow = new JFrame();
        myWindow.setSize(400,100);
        JLabel myLabel = new JLabel("My first window");
        myWindow.getContentPane().add(myLabel);
        WindowDestroyer myListener = new WindowDestroyer();
        myWindow.addWindowListener(myListener);
        myWindow.setVisible(true); } }
```

- The **listener** object is `WindowDestroyer`.
- The object `myListener` should be associated with the object (the window) `mywindow`, so that `myListener` will receive any event fired by the object `mywindow`:

  ```
  myWindow.addWindowListener(myListener);
  ```

- A listener class should be **defined**.

A Listener Class for Window Events

```
import javax.swing.*;
import java.awt.event.*;
public class WindowDestroyer extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
       System.exit(0); } }
```

- A **window listener** class for a GUI is often a derived class of the class `WindowAdapter`.
- The methods **inherited** from `WindowAdapter` class responds automatically to a different kind of event.
- Normally no new methods are added. Methods are **redefined**.
- The method that handles events that a window should be closed is `windowClosing`.
- The method `windowClosing` is redefined. The command ends the program and thus closes the window.
- The second `import` statement tells the compiler where the definitions for the `WindowAdapter` and for event handling are located.

Methods in the Class `WindowAdapter`

- `public void windowOpened(WindowEvent e)`

  Invoked when a window has been opened

- `public void windowClosed(WindowEvent e)`

  Invoked when a window has been closed

- `public void windowClosing(WindowEvent e)`

  Invoked when a window is in the process of being closed.

- `public void windowIconified(WindowEvent e)`

  Invoked when a window is iconified.

- `public void windowDeinconified(WindowEvent e)`

- `public void windowDeactivated(WindowEvent e)`

- `public void windowActivated(WindowEvent e)`

  Invoked when a window is activated, e.g. when you click in a window

- ...

Better Version of Our First Swing Program

```
import javax.swing.*;
public class FirstWindow extends JFrame
{
    public FirstWindow() {
        super();
        setSize(400,100);
        JLabel myLabel = new JLabel("My first window");
        getContentPane().add(myLabel);
        WindowDestroyer myListener = new WindowDestroyer();
        addWindowListener(myListener); } }
```

- This is the **style** you should follow in writing your own GUIs.
- You define a subclass of `JFrame` to define a **window interface**.
- The **base class** gives some basic window facilities.
- The **derived class** adds whatever additional features you want in your window interface.

Program that uses the Class `FirstWindow`

```
import javax.swing.*;
public class FirstWindowDemo  {
    public static void main(String[] args)  {

        FirstWindow window1 = new FirstWindow();
        window1.setVisible(true);

        FirstWindow window2 = new FirstWindow();
        window2.setVisible(true);
    }
}
```

A Window with Title and Color

Inherited Methods from the class `JFrame`:

- To give **title** to a window:

  ```
  setTitle("Second Window");
  ```

- To give the window a **background color**:

  ```
  getContentPane().setBackground(Color.BLUE);
  ```

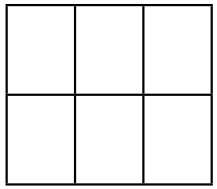
- The class `Color` contains constants for many of the common colors, e.g. `YELLOW`, `MAGENTA`, `BLACK` , . . .

- The class `Color` is in the AWT package.

Layout Manager

- We can add more than one label to the content pane.
- How are the labels arranged?
- The arranging is done by a special kind of object known as a **layout manager**.
- Different layout managers follow different **rules**.
  - `FlowLayout` is the simplest layout manager that arranges the components one after the other, going from the left to the right.
  - `BorderLayout` is a layout manager that places labels into the five regions `NORTH`, `SOUTH`, `EAST`, `WEST` and `CENTER`.
  - `GridLayout` is a layout manager that arranges components in rows and columns. The `add` method has only one argument, items are placed from left to right.

```
getContentPane().setLayout(new GridLayout(2,3));
```



Layout Manager – Example

```
import javax.swing.*;
import java.awt.*;
public class ThirdWindow extends JFrame {
    public ThirdWindow() {
      super();
      setSize(400,100);
      getContentPane().setLayout(new BorderLayout());
      JLabel label1 = new JLabel("My name is");
      getContentPane().add(label1, BorderLayout.NORTH);
      JLabel label2 = new JLabel("Slim");
      getContentPane().add(label2, BorderLayout.SOUTH);
      setTitle("Second Window");
      getContentPane().setBackground(Color.BLUE);
      WindowDestroyer myListener = new WindowDestroyer();
      addWindowListener(myListener); } }
```

Adding Buttons

```
import javax.swing.*;
import java.awt.*;
public class ButtonD extends JFrame
{
    public static void main(String[] args)  {
       ButtonD buttonGUI = new ButtonD();
       buttonGUI.setVisible(true); }

    public ButtonD() {
      super();
      setSize(400,100);
      JButton button = new JButton("Red");
      getContentPane().add(button);
      setTitle("Second Window");
      getContentPane().setBackground(Color.BLUE);
      WindowDestroyer myListener = new WindowDestroyer();
      addWindowListener(myListener); }
}
```

Adding Action Listeners

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ButtonDemo extends JFrame implements ActionListener
{
 ...
  public ButtonDemo() {
    ...
    getContentPane().setLayout(new FlowLayout());
    JButton button1 = new JButton("Red");
    button1.addActionListener(this);
    getContentPane().add(button1);
    JButton button2 = new JButton("Black");
    button2.addActionListener(this);
    getContentPane().add(button2);
    ...
}
```

Action Listeners and Action Events

- `button1.addActionListener(this);`

  registers `this` (`ButtonDemo`) as listener to receive events from the button called `button1`.

- An **action listener** is an object of type `ActionListener`. It is not A class but it is a property (interface).

- To make a class into an `ActionListener`, we need
  - add the statement `implements ActionListener` to the beginning of the class definition.
  - define a method named `actionPerformed`.

actionPerfomed Method

- In order to be an action listener a class must have a method named `actionPerformed`.
- The `actionPerformed` method is the only method required by the interface `ActionListener`.
- The code is typically a branching statement.
- Often the branching depends on `getActionCommand()`.

```
public void actionPerformed(ActionEvent e) {
 if (e.getActionCommand().equals("Red"))
            getContentPane().setBackground(Color.RED);
        else if (e.getActionCommand().equals("Black"))
            getContentPane().setBackground(Color.BLACK);
}
```

Mouse Events

- The mouse listeners allow you to receive events to process:
  - Button clicks, presses, or releases by the left, middle, or right buttons.
  - Moves and drags.
  - Which Modifier keys (shift, control, alt) were down when the event occurred.
  - Notification when the mouse enters or exits the component.
  - Scroll wheel movements.
- **Normally handled for you**.
- Sometimes used with graphics. If you are are drawing your own graphics (eg, on a `JComponent` or `JPanel`) and need to know where the user clicks, then you need to know about mouse events. You can easily add a mouse listener to a `JComponent` or `JPanel`.

The actions that a `MouseListener` catches:

- **press**: one of the mouse buttons is pressed.
- **release**: one of the mouse buttons is released.
- **click**: a mouse button was pressed and released without moving the mouse. This is perhaps the most commonly used.
- **enter**: mouse cursor enters the component. Often used to change cursor.
- **exit**: mouse cursor exits the component. Often used to restore cursor.

To listen for these events you will use `addMouseListener.`

MouseListener Interface and Mouse Events

- To implement a `MouseListener` interface, you must define the following methods. You can copy these definitions into your program and only make a meaningful body for those methods that are of interest.

```
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
```

- To get the mouse coordinates: All coordinates are relative to the upper left corner of the component with the mouse listener.

```
int getX() // returns the x coordinate of the event.
int getY() // returns the y coordinate of the event.
```

- To check for double clicks: Use the following `MouseEvent` method

```
int getClickCount() // number of mouse clicks
```

`JPanel` Class

- A GUI is often organized in a hierarchical fashion, with windowlike containers inside of other windowlike containers.
- `JPanel` is used to define subparts of a window. It is a very simple container class that does little more than grouping objects.
- A `JPanel` object is analogous to the braces used to combine a number of simpler Java statements into a single larger Java statement.
- A `JPanel` objects groups smaller objects, such as buttons and labels into a larger component.

Putting Buttons in a Panel

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class PanelDemo extends JFrame implements ActionListener
{
    public static void main(String[] args)
    {
        PanelDemo buttonGUI = new PanelDemo();
        buttonGUI.setVisible(true);
    }

    public PanelDemo()
    {
        super();
        setSize(400,100);
        WindowDestroyer myListener = new WindowDestroyer();
        addWindowListener(myListener);
        Container contentPane = getContentPane();
```

Putting Buttons in a Panel

```
        contentPane.setBackground(Color.BLUE);
        contentPane.setLayout(new BorderLayout());

        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(Color.WHITE);
        buttonPanel.setLayout(new FlowLayout());

        JButton button1 = new JButton("Red");
        button1.setBackground(Color.RED);
        button1.addActionListener(this);
        buttonPanel.add(button1);
        JButton button2 = new JButton("MAGENTA");
        button2.setBackground(Color.MAGENTA);
        button2.addActionListener(this);
        buttonPanel.add(button2);

        contentPane.add(buttonPanel, BorderLayout.SOUTH);
    }
```

Putting Buttons in a Panel

```
    public void actionPerformed(ActionEvent e)
    {
        if (e.getActionCommand().equals("Red"))
            getContentPane().setBackground(Color.RED);
        else if (e.getActionCommand().equals("MAGENTA"))
            getContentPane().setBackground(Color.MAGENTA);
    }

}
```