

# Software Development (CS2500)

## Lecture 38: Inner Classes

M.R.C. van Dongen

January 26, 2010

## Contents

1	Outline	1
2	The Graphics2D Class	1
3	Drawing on Demand	3
3.1	Drawing Two Widgets . . . . .	3
3.2	Finishing the Application . . . . .	4
4	Inner Classes	5
4.1	One Event Listener Class . . . . .	6
4.2	Two External Listener Classes . . . . .	6
4.3	Two Inner Listener Classes . . . . .	8
5	For Friday	10

## 1 Outline

This lecture continues our study of event handlers and GUIs. We start with an application that has several widgets in a JFrame. We implement the application by introducing *inner classes*

## 2 The Graphics2D Class

Before we start with our multi-widget application, let's have a look at some graphics widgets.

Creating a drawing widget is easy: you extend JPanel and override `paintComponent()`. The method `paintComponent()` is called when the JPanel is redrawn. Don't call it yourself: it's called automatically. The following is an example.

```

import java.awt.*;
import java.swing.*;

public class MyDrawPanel extends JPanel {
    @Override
    public void paintComponent( Graphics g ) {
        g.setColor( Color.green );
        g.fillRect( 20, 50, 100, 100 );
    }
}

```

The following shows a class that draws a circle with a random colour. You're invited to look up the methods `fillRect`, `fillOval`, and the `Color` constructor in the online [Java API documentation](#).

```

import java.awt.*;
import javax.swing.*;
import java.util.Random;

public class MyRandomPanel extends JPanel {
    private static final Random rand = new Random( );

    @Override
    public void paintComponent( Graphics g ) {
        g.fillRect( 0, 0, this.getWidth( ), this.getHeight( ) );
        int redPart  = rand.nextInt( 255 );
        int greenPart = rand.nextInt( 255 );
        int bluePart  = rand.nextInt( 255 );
        g.setColor( new Color( redPart, greenPart, bluePart ) );
        g.fillOval( 70, 70, 100, 100 );
    }
}

```

The following is a concrete class that draws randomly coloured circle.

```
import javax.swing.*;

public class RandomCircle {
    public static void main( String[] args ) {
        JFrame frame = new JFrame( "A Randomly Coloured Circle" );
        JPanel panel = new MyRandomPanel( );
        frame.getContentPane( ).add( panel );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        frame.setSize( 300, 300 );
        frame.setVisible( true );
    }
}
```

The type of the argument of `paintComponent( )` is `Graphics`. The type of the object referenced by the argument is actually `Graphics2D`. This should tell you that `Graphics2D` extends `Graphics`.

The class `Graphics2E` can do more than the class `Graphics`. For example, it lets you fill a shape with a gradient blend. Inside the `paintComponent( )` you use casting to create a `Graphics2E` reference from a `Graphics` reference. The following reminds you how this works.

```
public void paintComponent( Graphics g ) {
    Graphics2E g2e = (Graphics2E)g;
    GradientPaint grad = new GradientPaint( 70, 70, Color.blue,
                                           150, 150, Color.red );

    g2e.setPaint( grad );
    g2e.fillOval( 70, 70, 100, 100 );
}
```

You're invited to look up the `GradientPaint` constructor and the method `setPaint( )`. These methods are not examinable.

### 3 Drawing on Demand

This section demonstrates how to draw graphics on demand. We create a frame with two widgets: a drawing panel that draws randomly coloured circles, and a button. We create a listener that registers with the button. The listener waits until the user clicks the button. When the user clicks the button, this creates an action event. This triggers the listener's action even handler, `actionPerformed( )`. The action event handler simply repaints the frame. Repainting the frame repaints each of its widgets. As a result we get a new random colour for the circle.

#### 3.1 Drawing Two Widgets

For our application we need to draw two widgets. We know how to draw one:

```
frame.getContentPane( ).add( button );
```

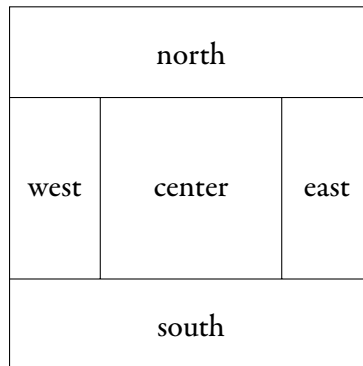


Figure 1: The five different regions of JFrame's content pane.

But how do we draw two? Figure 1 shows a key to solving the problem. As you can see, the frame's content pane has several regions.

When adding a widget to the frame's content pane you can use an overloaded version of the `add( )` method. This version takes an additional parameter which determines the region for the widget's position. The following shows how it works. (The additional regions work similarly.)

```
frame.getContentPane( ).add( BorderLayout.CENTER, button );
```

Java

### 3.2 Finishing the Application

The following shows how to finish our application. For simplicity the import statements have been omitted.

```
public class RandomColours implements ActionListener {
    JFrame frame;

    public static void main( String[] args ) {
        RandomColours rc = new RandomColours( );
    }

    @Override
    public void actionPerformed((ActionEvent event) {
        frame.repaint( );
    }

    // Constructor in next listing
}
```

Java

The `actionPerformed( )` method calls `frame.repaint( )`. This calls `paintComponent( )` on every widget in the frame. The `paintComponent( )` of the panel draw a circle with a random colour, so this

gives us (more than likely) a different colour each time the user clicks the button.

The following is the constructor.

```
private RandomColours( ) {  
    frame = new JFrame( "Random Colours" );  
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );  
  
    JButton button = new JButton( "Change Colour" );  
    button.addActionListener( this );  
  
    // The class MyRandomPanel is listed on Page 2.  
    MyRandomPanel panel = new MyRandomPanel( );  
    Container contentPane = frame.getContentPane( );  
  
    contentPane.add( BorderLayout.SOUTH, button );  
    contentPane.add( BorderLayout.CENTER, panel );  
  
    frame.setSize( 300, 300 );  
    frame.setVisible( true );  
}
```

## 4 Inner Classes

In this section we shall study three techniques which may be used to implement *several* event listeners. For the purpose of this section we shall only consider two even listeners.

Let's change our last application. We keep the button and the panel. However, we add one more button and a JLabel. The first time the user clicks the new button the JLabel's text will change.

So, how do we do that? There are three possible techniques:

- One event listener which is registered with both buttons. Since there's only one listener, it has to handle two kinds of event. The listener determines the event source: Button 1 or 2? And then handles it.
- We have two separate *external* event listener classes. Each class has its own listener. Since each class has its *own* listener, there's no need to determine the source of events. Since the classes are external, they don't have access to the frame and widget attributes of the main class. Therefore, they have to encapsulate their own widget.
- Two *inner* event listener classes. The inner classes are *part* of the main class. They have access to frame and widget attributes. Therefore, the inner classes don't have to encapsulate their own widgets. As with the second option, there's no need to determine the source of events.

The remainder of this section demonstrates each of these three techniques.

## 4.1 One Event Listener Class

The following is the core code which is needed to implement the first technique. The main application has four attributes. The `JFrame` is needed to call `repaint( )` in the event handler `actionPerformed( )`. Likewise, the `JLabel` is needed for changing the text. Finally, the `JButtons` are needed to determine the event source in the event handler.

```
public class OptionOne implements ActionListener {
    // Instance attributes needed.
    private final JFrame frame;
    private final JLabel label;
    private final JButton colourButton;
    private final JButton labelButton;

    public static void main( String[] args ) {
        new OptionOne( ); // No assignment needed.
    }

    @Override
    public void actionPerformed((ActionEvent event) ) {
        if (event.getSource( ) == colourButton) {
            frame.repaint( );
        } else if (event.getSource( ) == labelButton) {
            label.setText( "New Text" );
        }
    }

    // Constructor omitted.
}
```

This option is not very clean because we may have to change the implementation of the event handler if we change the implementation of the class. For example, if we decide to put the `JButtons` in an array then we have to change the event handler.

## 4.2 Two External Event Listener Classes

The second option is cleaner. Here we have two event listener classes, each of which is implemented in its own file. There is one class listening to the colour button event and one listening to the label button event. The first class encapsulates the `JFrame` and the second class encapsulates the `JLabel`. *What can the class “see?”* Each class may encapsulate its own button, but is not required.

The following shows a possible implementation for the first outer class. The class is only meant to serve as an example, which should explain why name of the class is not very imaginative. The second class may be implemented in a similar way.

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

public class Outer1 implements ActionListener {
    private final JFrame frame;

    public Outer1( JFrame frame ) {
        this.frame = frame;
    }

    @Override
    public void actionPerformed((ActionEvent event) ) {
        frame.repaint( );
    }
}
```

The details for the main class are in the listing on the next page. Note that there is no need for this class to implement the `ActionListener` interface; listening is taken care of by the two external classes. Also note that this class no longer needs instance attributes: the `main( )` sets everything up. The `import` statements have been omitted.

```

public class OptionTwo {
    // No instance attributes needed.
    public static void main( String[] args ) {
        JFrame frame = new JFrame( "OptionTwo" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        JLabel label = new JLabel( "Initial Text" );

        JButton colourButton = new JButton( "Change Colour" );
        JButton labelButton = new JButton( "Change Label" );
        colourButton.addActionListener( new Outer1( frame ) );
        labelButton.addActionListener( new Outer2( label ) );

        MyRandomPanel panel = new MyRandomPanel( );
        Container contentPane = frame.getContentPane( );

        contentPane.add( BorderLayout.SOUTH, colourButton );
        contentPane.add( BorderLayout.CENTER, panel );
        contentPane.add( BorderLayout.WEST, label );
        contentPane.add( BorderLayout.EAST, labelButton );

        frame.setSize( 500, 300 );
        frame.setVisible( true );
    }
}

```

This option is cleaner than the first option but we have to implement two separate external classes which have attributes which are also known outside the class. As a consequence we have to write special constructors for the classes, which, as we shall see in a few moments, can be avoided.

### 4.3 Two Inner Event Listener Classes

The third and last option is the cleaner option. We simply create the event listener classes inside the main class. Java allows this and the resulting classes are called *inner classes*. Inner classes are almost as flexible as normal classes and you can encapsulate them inside another class.

Since they are inside their “parent” class they can see all any attribute which is owned by the parent class. The following shows the main details of a possible implementation. The `import` statements have been omitted. Notice that this time instance attributes are needed because they are needed by the inner classes. Carefully compare this to the implementation of `OptionTwo`.



```

public class OptionThree {
    // Instance attributes needed.
    private final JFrame frame;
    private final JLabel label;

    public static void main( String[] args ) {
        new OptionThree( );
    }

    private OptionThree( ) {
        frame = new JFrame( "OptionThree" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        label = new JLabel( "Initial Text" );

        JButton colourButton = new JButton( "Change Colour" );
        JButton labelButton = new JButton( "Change Label" );
        colourButton.addActionListener( new Inner1( ) );
        labelButton.addActionListener( new Inner2( ) );

        MyRandomPanel panel = new MyRandomPanel( );
        Container contentPane = frame.getContentPane( );

        contentPane.add( BorderLayout.SOUTH, colourButton );
        contentPane.add( BorderLayout.CENTER, panel );
        contentPane.add( BorderLayout.WEST, label );
        contentPane.add( BorderLayout.EAST, labelButton );

        frame.setSize( 500, 300 );
        frame.setVisible( true );
    }

    // Inner classes in next listing
}

```

The following are the inner classes. Notice how simple they are. Arguably this is the better of the three options.

```

public class OptionThree {
    private final JFrame frame;
    private final JLabel label;

    // Constructor and main omitted.

    private class Inner1 implements ActionListener {
        @Override
        public void actionPerformed((ActionEvent event) {
            frame.repaint( );
        }
    }

    private class Inner2 implements ActionListener {
        @Override
        public void actionPerformed((ActionEvent event) {
            label.setText( "Text Changed" );
        }
    }
}

```

## 5 For Friday

Study the lecture notes, carefully study the differences between the three different options in Section 4, and study Chapter 11 (up to Page 381).