

Chapter 3 Design Pattern

Design Pattern

- Documents the solution to a problem in a very general way.
- Singleton design pattern describes a model for building a class that may only have one instance.

Dice Class: Example of Singleton Design Pattern

```
public class Dice
{
    // static reference that identifies the single instance
    private static Dice dice = null;
    private Random rnd;
    private int dice1, dice2;

    // private constructor is called by the method getDice()
    // to create a single instance of the class
    private Dice()
    {
        // create the random number generator
        rnd = new Random();
    }
}
```

Dice Class (continued)

```
// if no object currently exists, the method calls
// the private constructor to create an instance;
// in any case, the method returns the static
// reference variable dice
public static Dice getDice()
{
    if (dice == null)
    {
        dice = new Dice();
    }
    return dice;
}
```

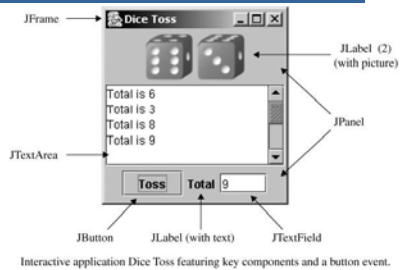
Dice Class (continued)

```
// toss the dice and update values
// for dice1 and dice2
public void toss()
{
    dice1 = rnd.nextInt(6) + 1;
    dice2 = rnd.nextInt(6) + 1;
}
// access methods getOne(), getTwo(),
// and getTotal()
. . .
}
```

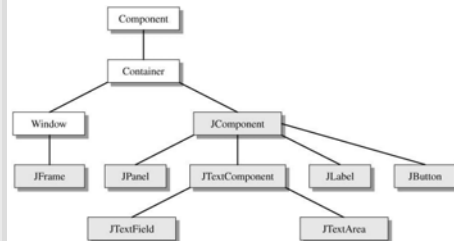
GUI Application Design

- GUI = "Graphical User Interface".
- GUI Components
 - Frame - Main application window
 - Panel - Container to which other components are added
 - Label - Displays a string and/or an image
 - Text field - Allows input/output of text line
 - Button - Responds to pressing with mouse

Key Components of a GUI Application



Inheritance Hierarchy for Java GUI Components



GUI Application Design Pattern

```

<import statements>
public class DiceToss extends JFrame
{
  < component and application variables>
  public static void main(String[] args)
  {
    DiceToss app = new DiceToss();
    app.setVisible(true);
  }
  // constructor creates components and adds them to frame
  public DiceToss()
  {
    <initialize frame attributes, create components>
  }
  // inner class defines an event handler object
  private class EventHandler implements <event interface>
  { }
}
  
```

GUI Design Pattern Import Statements

- Import statements specify the class libraries that the application will use

```

import java.awt.*; // original Java AWT
import javax.swing.*; // Swing component classes
import java.awt.event.*; // Java event classes and interfaces
  
```

GUI Design Pattern Application Variables

- Declare as application variables the GUI components and any data that will be accessed by event handlers, utility methods, and inner classes.

Dice Class Variables

```

class DiceToss extends JFrame
{
  // application variables
  private JPanel panelA, panelB;
  private JLabel dieLabel1, dieLabel2, totalLabel;
  private JTextField totalField;
  private JTextArea totalArea;
  private JButton tossButton;
  private Dice d = Dice.getDice();
  private ImageIcon[] diePict = {null,
    new ImageIcon("die1.gif"), new ImageIcon("die2.gif"),
    new ImageIcon("die3.gif"), new ImageIcon("die4.gif"),
    new ImageIcon("die5.gif"), new ImageIcon("die6.gif")};
  ...
}
  
```

GUI Application Constructor

- Initializes frame properties and loads the components.
 - Set title with `setTitle(titleStr)`.
 - `setBounds(posX, posY, width, height)` sets the size of the frame and its position on the screen.
 - `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)` closes the frame and exits the application when the user clicks the close box in the frame.

GUI Application Constructor (end)

- Components in a frame reside in a region called the *content pane*.
- Use `getContentPane()` to access the content pane and assign it to a Container reference variable.

```
Container content = getContentPane();
```
- Create and organize components and add them to the content pane.

DiceToss Constructor

```
public DiceToss()
{
    . . .
    // establish the content pane for the window
    Container content = getContentPane();
    content.setLayout(new BorderLayout());

    // panel and labels for the two die pictures
    JPanel panelA = new JPanel();
    dieLabel1 = new JLabel();
    dieLabel2 = new JLabel();
    panelA.add(dieLabel1);
    panelA.add(dieLabel2);

    // text area to store results of
    // successive dice tosses
    totalArea = new JTextArea(10, 15);
}
```

DiceToss Constructor (2)

```
// panel with toss button, total label,
// and text field for the total
// the toss button has an action listener
// (see next section)
JPanel panelB = new JPanel();
tossButton = new JButton("Toss");
tossButton.addActionListener(new TossEvent());

totalLabel = new JLabel("Total");
totalField = new JTextField(4);
panelB.add(tossButton);
panelB.add(totalLabel);
panelB.add(totalField);
```

DiceToss Constructor (end)

```
// add panels and individual components to the
// content pane using compass point constants
// for class BorderLayout.
// the text area is embedded in a JScrollPane
// to add scroll bars
content.add(panelA, BorderLayout.NORTH);
content.add(new JScrollPane(totalArea),
            BorderLayout.CENTER);
content.add(panelB, BorderLayout.SOUTH);
}
```

Event Listeners

- An event is an action such as a mouse click, a button press, or a keystroke.
- An event listener is an object that resides in a component and waits for the occurrence of an event and handles the actions required by the event.
- Add a listener to a component using `addTypeListener(...)` where Type is the type of the event.

Action Events

- An event which indicates that a component-defined action occurred. This event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every ActionListener object that registered to receive such events using the component's addActionListener method

Action Events (2)

```
void addActionListener(ActionListener handlerObj)
```

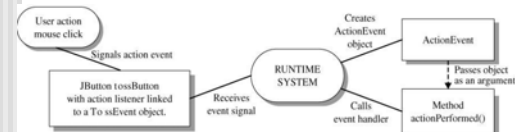
- Adds action listener to a component. The parameter is the event handler that responds to the event.
- Normally use a private inner class to define an event handler. An inner class has access to all the variables in the outer class and can update them.

Action Events (end)

- An action listener event handling class must implement the ActionListener interface. The interface has only one method

```
void actionPerformed(ActionEvent ae)
```
- Invoked when an action occurs

Action Event Sequence



DiceToss Event Handler Class

```
private class TossEvent implements ActionListener
{
    public void actionPerformed(ActionEvent ae)
    {
        d.toss();
        dieLabel1.setIcon(diePict[d.getOne()]);
        dieLabel2.setIcon(diePict[d.getTwo()]);
        totalArea.append("Total is " +
            d.getTotal() + "\n");
        totalField.setText(" " + d.getTotal());
    }
}
```