Week9

Programming Fundamentals II

Course Outline

8. Events + GUI	เสนอหัวข้อ Project
9. UI programming	Lab7
10. Exceptions	Lab8
11. Testing and debugging	Lab9
12. Generics	Lab10
13. Team project	เสนอความคืบหน้า Project
14. Concurrency	-

Final Exam Lab Exam Present Project

Week 9 **Event + Graphic**

1. Event
2. Using the Listener
3. Event Application
4. Listener Implementation
5. GUI features
6. MouseListener
7. Mouse Example
8. GUI and Graphics
9. Drawing Rectangles and Ovals
10. Colors and Filled Shapes
11. Drawing Arcs
12. Drawing with Polymorphism

Event

Programming Fundamentals II

Event

The programmer must:

1. Implement the listener, by coding its event handler methods
2. 'Link' the GUI components in their program with the implemented listener

The Java runtime system will automatically pass events to the handlers for processing.

Event

There are manyListener interfaces that can handle events from different GUI components. I'll look at:

- `ActionListener`
- `ItemListener`
- `MouseListener`
- `MouseMotionListener`

Event

- เป็นการจัดการในส่วนโต้ตอบระหว่างผู้ใช้ กับ GUI ด้วย เช่น การคลิกที่ปุ่ม Button หรือ การกด Enter บนแป้นพิมพ์ที่ Text Field เป็นต้น
- จะต้องตรวจจับเหตุการณ์ก่อนว่าผู้ใช้มีการกระทำกับคอมโพเนนต์ที่เหตุการณ์ใดบ้าง ซึ่งเป็นหน้าที่ของออบเจ็กต์ที่เป็น Event Listener
- จากนั้นนำ Event Listener ไปผูกติดกับคอมโพเนนต์ เช่น ต้องการตรวจจับเหตุการณ์ที่ปุ่ม CloseButton โดยสร้างออบเจ็กต์จากคลาส ButtonListener เพื่อให้ออบเจ็กต์ดังกล่าวทำหน้าที่เป็น Event Listener และนำไปผูกติดกับปุ่ม CloseButton

รูปแบบ

```
buttonName.addActionListener(new ButtonListener());
```

โดยที่

buttonName	เป็นชื่อออบเจ็กต์ที่ประกาศจากคลาส JButton
ButtonListener	เป็นออบเจ็กต์จากคลาส ButtonListener

Using the Listener

- การเขียนโปรแกรมในส่วนจัดการเหตุการณ์ สามารถจัดการให้อยู่ภายใน Inner Class
- โดยสร้างเป็นคลาสที่มีการ implements อินเตอร์เฟสซึ่งสอดคล้องกับเหตุการณ์ที่เกิดขึ้น และ
- กำหนดหน้าที่การทำงานให้กับเมธอดในอินเตอร์เฟสนั้นๆ

รูปแบบ

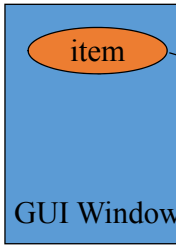
```
private class ButtonListener implements ActionListener  {
        public void actionPerformed(ActionEvent e)  {
                if(e.getSource()==buttonName)  {
                }
        }
}
```

โดยที่

ButtonListener	เป็นออบเจ็กต์จากคลาส ButtonListener
ActionListener	เป็นเหตุการณ์ที่ต้องการตรวจจับ
actionPerformed	เป็นชื่อเมธอด
buttonName	เป็นชื่อออบเจ็กต์ที่ประกาศจากคลาส JButton

Using the Listener

The GUI component must be 'linked' to code which implements the method in the listener.



the 'link' which sends an event e

```
public class Foo implements
                ActionListener
{
  public void actionPerformed(
              ActionEvent e)
    { // do something with e
      System.out.println("Ouch");
  }
}
```


Using the Listener

We write a single listener to handle all the events triggered in the program

- `implements the ActionListener interface` ①
- `defines an actionPerformed() method` ②

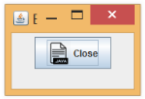
We must register the listener with each component ③

- `component.addActionListener(listener)`

Close Button

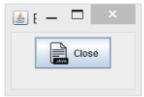
```
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  public class Lec09_CloseEvent extends JFrame
7  {
8      private JPanel p;  Icon ani;  JButton b;
9      public Lec09_CloseEvent(String title)
10     {
11         super(title);
12         p = new JPanel();
13         String str1 = "E:/CPE/Lecture/Year2016/CODE/JAVA/w08_workspace_java/Lec8_GUI/src/P81/java_icon01.png";
14         ani = new ImageIcon(str1);
15         b = new JButton("Close", ani);
16         b.addActionListener(new ButtonListener());
17         p.add(b);
18         add(p);
19     }
20     private class ButtonListener implements ActionListener
21     {
22         public void actionPerformed(ActionEvent e)
23         {
24             if (e.getSource() == b)
25             {
26                 JOptionPane.showMessageDialog(null, "See You Again !!!");
27                 System.exit(0);
28             }
29         }
30     }
31     public static void main(String[] args)
32     {
33         Lec09_CloseEvent b = new Lec09_CloseEvent("Button Test");
34         b.setSize(170, 120);
35         b.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
36         b.setVisible(true);
37
38     }
39 }
```

③ ① ②

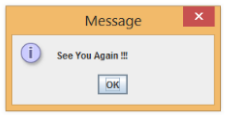


Close Button

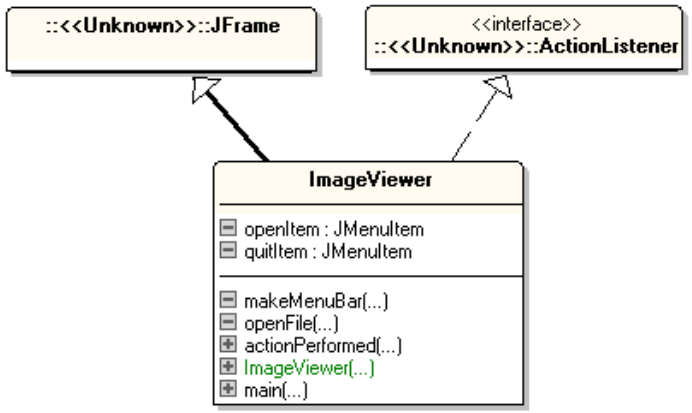
```
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  public class Lec09_CloseEvent extends JFrame
7  {
8      private JPanel p;  Icon ani;  JButton b;
9      public Lec09_CloseEvent(String title)
10     {
11         super(title);
12         p = new JPanel();
13         String str1 = "E:/CPE/Lecture/Year2016/CODE/JAVA/w08_workspace_java/Lec8_GUI/src/P81/java_icon01.png";
14         ani = new ImageIcon(str1);
15         b = new JButton("Close", ani);
16         b.addActionListener(new ButtonListener());
17         p.add(b);
18         add(p);
19     }
20     private class ButtonListener implements ActionListener
21     {
22         public void actionPerformed(ActionEvent e)
23         {
24             if (e.getSource() == b)
25             {
26                 JOptionPane.showMessageDialog(null, "See You Again !!!");
27                 System.exit(0);
28             }
29         }
30     }
31     public static void main(String[] args)
32     {
33         Lec09_CloseEvent b = new Lec09_CloseEvent("Button Test");
34         b.setSize(170, 120);
35         b.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
36         b.setVisible(true);
37
38     }
39 }
```



③ ① ②

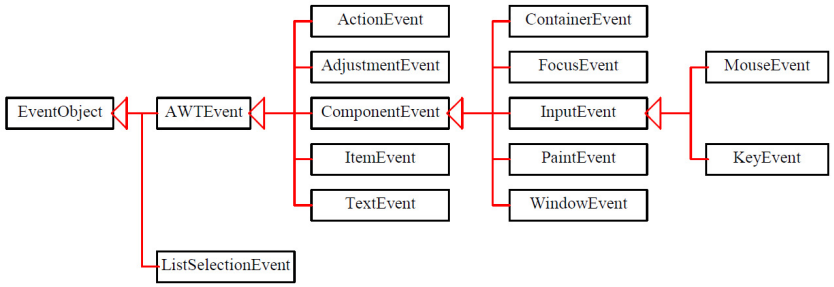


Class Diagram



multiple inheritance, using an interface

Event Class



The root class of the event classes is java.util.EventObject.

The hierarchical relationships of some event classes are shown above.

Event User Actions

User Action	Source Object	Event Type Generated
Click a button	JButton	ActionEvent
Press return on a text field	JTextField	ActionEvent
Select a new item	JComboBox	ItemEvent, ActionEvent
Select item(s)	JList	ListSelectionEvent
Click a check box	JCheckBox	ItemEvent, ActionEvent
Click a radio button	JRadioButton	ItemEvent, ActionEvent
Select a menu item	JMenuItem	ActionEvent
Move the scroll bar	JScrollBar	AdjustmentEvent
Window opened, closed, etc.	Window	WindowEvent
Mouse pressed, released, etc.	Component	MouseEvent
Key released, pressed, etc.	Component	KeyEvent
Component added or removed etc.	Container	ContainerEvent
Component moved, etc.	Component	ComponentEvent
Component gained or lost focus	Component	FocusEvent

☞Table lists external user actions, source objects, and event types generated.

Event Method

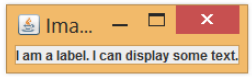
ประเภทเหตุการณ์	อินเตอร์เฟส	เมธอด
ActionEvent	ActionListener	actionPerformed(ActionEvent)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
ComponentEvent	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
ContainerEvent	ContainerListener	componentAdded(Container Event) componentRemoved(Container Event)
FocusEvent	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)

Ex. Event Application

```
7  //public class Event01ImageViewer
8  public class Event01ImageViewer extends JFrame implements ActionListener
9  {
10   private JMenuItem openItem, quitItem;
11   public Event01ImageViewer()
12   {
13       super("ImageViewer version 0.XX");
14       Container c = getContentPane();
15       JLabel label = new JLabel("I am a label. I can display some text.");
16       c.add(label);
17       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18       setSize(300, 200);
19       setVisible(true);
20   }
21   public void actionPerformed(ActionEvent event)
22   { // the event handler code
23
24   }
25
26   public static void main(String[] args)
27   {
28       new Event01ImageViewer();
29   }
30 }
```

Ex. Event Application

```
7  //public class Event01ImageViewer
8  public class Event01ImageViewer extends JFrame implements ActionListener
9  {
10   private JMenuItem openItem, quitItem;
11   public Event01ImageViewer()
12   {
13       super("ImageViewer version 0.XX");
14       Container c = getContentPane();
15       JLabel label = new JLabel("I am a label. I can display some text.");
16       c.add(label);
17       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18       setSize(300, 200);
19       pack();
20       setVisible(true);
21   }
22   public void actionPerformed(ActionEvent event)
23   { // the event handler code
24
25   }
26
27   public static void main(String[] args)
28   {
29       new Event01ImageViewer();
30   }
31 }
```

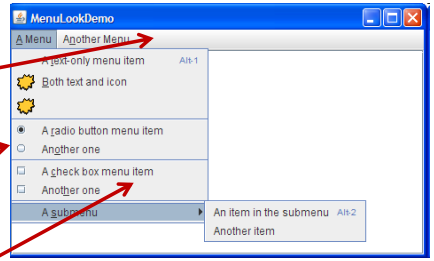


Ex. Event Application : Adding Menus

1 `JMenuBar`
- contains the menus

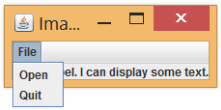
2 `JMenu`
- contains the menu items

3 `JMenuItem`
- individual items in a menu



Ex. Event Application

```
7  //public class Event01ImageViewer
8  public class Event01ImageViewer extends JFrame implements ActionListener
9  {
10   private JMenuItem fileMenu, openItem, quitItem;
11   public Event01ImageViewer()
12   {
13       super("ImageViewer version 0.XX");
14       makeMenuBar();
15       Container c = getContentPane();
16       JLabel label = new JLabel("I am a label. I can display some text.");
17       c.add(label);
18       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19       setSize(300, 200);
20       pack();
21       setVisible(true);
22   }
23
24   private void makeMenuBar()
25   {
26     JMenuBar menubar = new JMenuBar();
27     setJMenuBar(menubar); // add to 'menu area' of JFrame
28
29     // create the File menu
30     fileMenu = new JMenu("File");
31     menubar.add(fileMenu);
32
33     openItem = new JMenuItem("Open");
34     fileMenu.add(openItem);
35
36     quitItem = new JMenuItem("Quit");
37     fileMenu.add(quitItem);
38   }
```



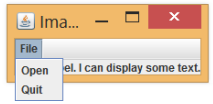
Listener Implementation

There are three ways to implement a listener:

1. have the GUI class implement the **listener itself**
2. implement the listener as an **inner class**
3. implement multiple listeners as **anonymous (inner)classes**

EX. 1. listener itself

```
24  private void makeMenuBar()
25  {
26    JMenuBar menubar = new JMenuBar();
27    setJMenuBar(menubar); // add to 'menu area' of JFrame
28
29    // create the File menu
30    fileMenu = new JMenu("File");
31    menubar.add(fileMenu);
32
33    openItem = new JMenuItem("Open");
34    openItem.addActionListener(this);
35    fileMenu.add(openItem);
36
37    quitItem = new JMenuItem("Quit");
38    quitItem.addActionListener(this);
39    fileMenu.add(quitItem);
40  }
41
42  public void actionPerformed(ActionEvent event)
43  {
44     Object src = event.getSource();
45
46     if (src == openItem)
47        openFile();
48     else if (src == quitItem)
49        System.exit(0);
50     else
51        System.out.println("Cannot process action event for " + event.getActionCommand());
52  }
53
54  private void openFile()
55  {
56    System.out.println("open file");
57  }
```



Listener Implementation

There are three ways to implement a listener:

1. have the GUI class implement the **listener itself**
2. implement the listener as an **inner class**
3. implement multiple listeners as **anonymous (inner)classes**

2. Inner Class

An inner class is defined inside another class:

```
public class Enclosing
{
     // fields, methods

     class Inner
     {
        // fields, methods
     } // end of Inner class

} // end of Enclosing class
```

EX. 2. Inner Class

```
8  public class Event02ImageViewer extends JFrame
9  {
10   private JMenuItem fileMenu, openItem, quitItem;
11
12   public Event02ImageViewer()
13   {
14       super("ImageViewer version 0.XX");
15       makeMenuBar();
16       Container c = getContentPane();
17       JLabel label = new JLabel("I am a label. I can display some text.");
18       c.add(label);
19       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20       setSize(300, 200);
21       pack();
22       setVisible(true);
23   }
24
25   private void makeMenuBar()
26   {
27     JMenuBar menubar = new JMenuBar();
28     setJMenuBar(menubar); // add to 'menu area' of JFrame
29
30     // create the File menu
31     fileMenu = new JMenu("File");
32     menubar.add(fileMenu);
33
34     MenuHandler mh = new MenuHandler();
35     openItem = new JMenuItem("Open");
36     openItem.addActionListener( mh );
37     fileMenu.add(openItem);
38
39     quitItem = new JMenuItem("Quit");
40     quitItem.addActionListener( mh );
41     fileMenu.add(quitItem);
42   }
```


EX. 2. Inner Class

```
44   class MenuHandler implements ActionListener
45   {
46     public void actionPerformed(ActionEvent event)
47     {
48       Object src = event.getSource();
49       if (src == openItem)
50         openFile();
51       else if (src == quitItem)
52         System.exit(0);
53       else
54         System.out.println("Cannot process action event for " + event.getActionCommand());
55     }
56     private void openFile()
57     {
58         System.out.println("open file");
59     }
60   }
61
62   public static void main(String[] args)
63   {
64       new Event02ImageViewer();
65   }
66 }
```


Listener Implementation

There are three ways to implement a listener:

1. have the GUI class implement the **listener itself**
2. implement the listener as an **inner class**
3. implement multiple listeners as **anonymous (inner)classes**

3. An Anonymous (Inner) Class

```
8  public class Event03ImageViewer extends JFrame
9  {
10   public Event03ImageViewer()
11   {
12       super("ImageViewer version 0.XX");
13       makeMenuBar();
14       Container c = getContentPane();
15       JLabel label = new JLabel("I am a label. I can display some text.");
16       c.add(label);
17       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18       setSize(300, 200);
19       pack();
20       setVisible(true);
21   }
22   private void makeMenuBar()
23   {
24     JMenuBar menubar = new JMenuBar();
25     setJMenuBar(menubar); // add to 'menu area' of JFrame
26
27     // create the File menu
28     JMenu fileMenu = new JMenu("File");
29     menubar.add(fileMenu);
30
31     JMenuItem openItem = new JMenuItem("Open");
32     openItem.addActionListener( new ActionListener() {
33         public void actionPerformed(ActionEvent e)
34         { openFile(); }
35       });
36     fileMenu.add(openItem);
37
38     JMenuItem quitItem = new JMenuItem("Quit");
39     quitItem.addActionListener( new ActionListener() {
40         public void actionPerformed(ActionEvent e)
41         { System.exit(0); }
42       });
43     fileMenu.add(quitItem);
44   }
```

```
46   private void openFile()
47   {
48     System.out.println("open file");
49   }
50
51   public static void main(String[] args)
52   {
53       new Event03ImageViewer();
54   }
55 }
```

Steps in GUI Creation

- The GUI is initialised in the class' constructor method.
- Initialisation steps:
  1. get the container for the frame
  2. set the layout manager (FlowLayout)
  3. declare the GUI components
  4. add them to the container
  5. register the components with event handlers
  6. set window properties

TextFieldTest.java

```
// The JTextField GUI in a Java app
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TextFieldTest extends JFrame
{
   private JTextField jtf;
      // global since used in actionPerformed()

   public TextFieldTest()
   {
      super( "Testing JTextField" );

      Container c = getContentPane();  ①
      c.setLayout( new FlowLayout() );  ②
            :
```


TextFieldTest.java

```
      // label and text entry field
      JLabel jl = new JLabel("Enter your name:");  ③
      jtf  = new JTextField(25); // 25 chars wide
      c.add( jl );
        c.add( jtf );  ④

      // Handle events from pressing return
⑤     jtf.addActionListener( new ActionListener() {
          public void actionPerformed(ActionEvent e)
          {  System.out.println("You entered " +
                   e.getActionCommand() );
             jtf.setText("");
       }
      } );

⑥    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     setSize(500,100);
     setVisible(true);
} // end of TextFieldTest()
```


Event Interface Example

Listener Interfaces

- ActionListener
- ItemListener
- MouseListener
- MouseMotionListener

1. Button Example
2. TextField Example
3. Check Boxes Example
4. Radio Buttons Example
5. Combo Box Example
6. Mouse Example

1. ButtonTest.java

```
public class ButtonTest extends JFrame
{  private int pressCount = 1;

   public ButtonTest()
   {
       super( "Testing JButton" );

       Container c = getContentPane();
       c.setLayout( new FlowLayout() );

       JButton jb = new JButton( "Press me" );
       c.add( jb );

       jb.addActionListener( new ActionListener() {
           public void actionPerformed(ActionEvent e)
           {  System.out.println("Pressed " +  pressCount++);
           }
       } );

       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       setVisible( true );
}
```


2. TextFieldTest.java

```
public class TextFieldTest extends JFrame
{
   private JTextField jtf;
   public TextFieldTest()
   {
       super( "Testing JTextField" );
       Container c = getContentPane();
       c.setLayout( new FlowLayout() );
       JLabel jl = new JLabel("Enter your name:");
       jtf  = new JTextField(25);   // 25 chars wide
       c.add( jl );
       c.add( jtf );

         jtf.addActionListener( new ActionListener() {
           public void actionPerformed(ActionEvent e)
           {  System.out.println("You entered " +
                     e.getActionCommand() );
              jtf.setText("");
           }
         } );

       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       setSize(500,100);
       setVisible(true);
   }
```


3. Check Boxes Example

```
public class CheckBoxTest extends JFrame {
   public CheckBoxTest(){
       super( "Testing JCheckBox" );
       Container c = getContentPane();
       c.setLayout( new FlowLayout() );

       JCheckBox jck1 = new JCheckBox("Pepperoni");
       JCheckBox jck2 = new JCheckBox("Mushroom");

       c.add( jck1 ); c.add( jck2 );

       jck1.addActionListener( new ActionListener() {
       public void actionPerformed(ActionEvent e)
       {       System.out.println("event = " + e); }
         } );
       jck2.addItemListener( new ItemListener() {
               public void itemStateChanged(ItemEvent e)
               {  if (e.getStateChange() == e.SELECTED)
                       System.out.print("selected ");
                  else System.out.print("de-selected ");
                       System.out.print("Mushroom\n");
             }
           } );

       setDefaultCloseOperation(
       JFrame.EXIT_ON_CLOSE);
       setSize(500,100);
       setVisible(true);
   }
```


GUI features: JPanel

Programming Fundamentals II

Three Step GUI

There are three main steps to creating a GUI for a Java application:

1. Declare the GUI **components;**

2. Implement the **event handlers** for the components;

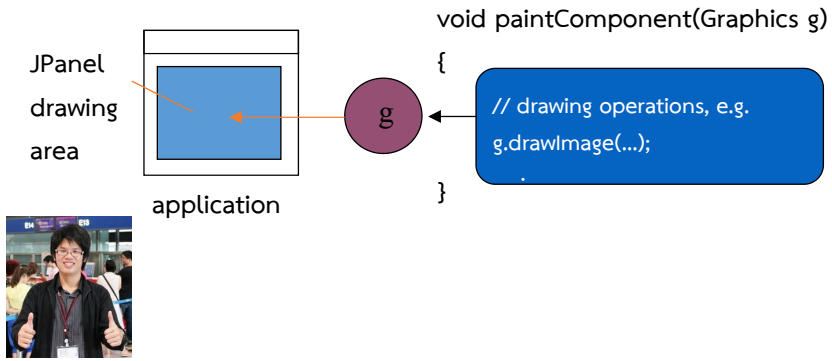
3. Position the components on the screen by using **layout managers** and/or **containers**.

Painting with JPanel

1. One of the uses of JPanel is as a 'canvas' (painting surface).
2. Its paintComponent() method can be overridden, and then draw/paint operations can be added to it.
3. Whenever the JPanel is redrawn, paintComponent() is called to draw the panel.
4. The paintComponent() declaration is:
   - public void paintComponent(Graphics g)
   - g is the graphics context for the panel

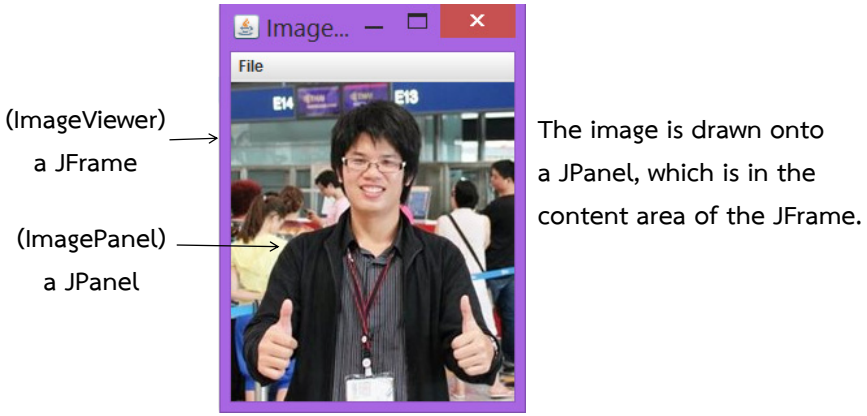
The Graphics Context

g links to the panel's drawing area on screen.



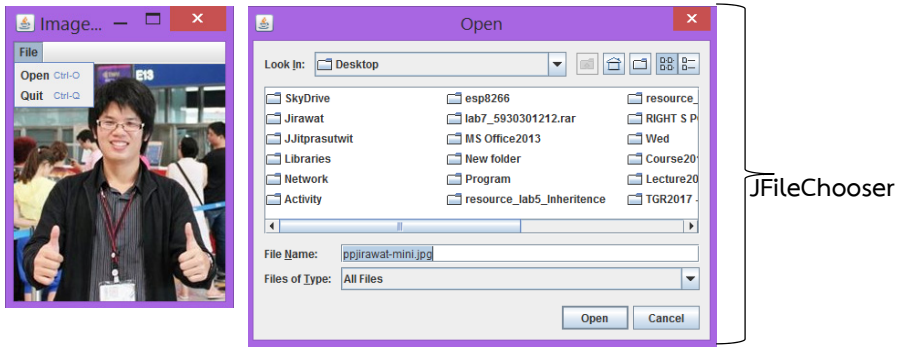
The Final ImageViewer

g links to the panel's drawing area on screen.



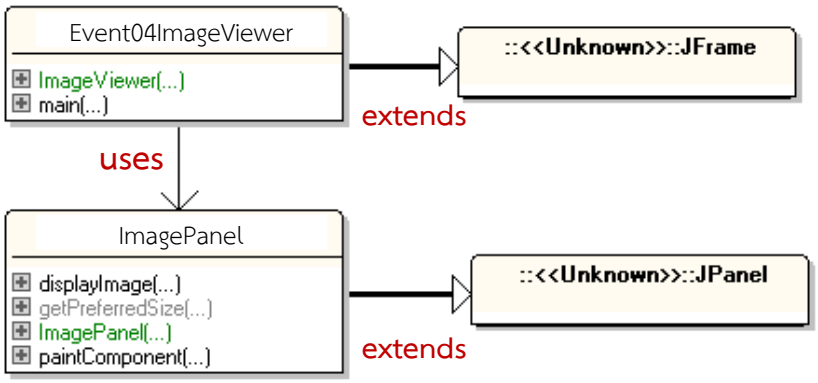
Other Features

The "open" menu item now calls JFileChooser to let the user choose an image file to load and display.



The menu items now have keyboard shortcuts:

Class Diagrams



Event04ImageViewer.java (1/3)

```
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.image.*;
6  import java.io.*;
7  import javax.imageio.*;
8  import javax.swing.*;
9
10 public class Event04ImageViewer extends JFrame
11 {
12     private JFileChooser fileChooser;
13     private ImagePanel imagePanel;
14     public Event04ImageViewer()
15     {
16         super("ImageViewer Final");
17         fileChooser = new JFileChooser( System.getProperty("user.dir") );
18         makeMenuBar();
19         Container c = getContentPane();
20         imagePanel = new ImagePanel();
21         c.add(imagePanel);
22         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23         setSize(300, 200);
24         pack();
25         setVisible(true);
26     }
27
```


Event04ImageViewer.java (2/3)

```
28     private void makeMenuBar()
29     {
30         int shortcut_mask = Toolkit.getDefaultToolkit().getMenuShortcutKeyMask();
31
32         JMenuBar menubar = new JMenuBar();
33         setJMenuBar(menubar);
34
35         JMenu fileMenu = new JMenu("File");
36         menubar.add(fileMenu);
37
38         JMenuItem openItem = new JMenuItem("Open");
39         openItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, shortcut_mask));
40         openItem.addActionListener(new ActionListener() {
41             public void actionPerformed(ActionEvent e)
42             { File f = chooseImage();
43               if (f != null)
44                   imagePanel.displayImage(f);
45               pack();   // triggers resizing to fit image
46             }
47           });
48           fileMenu.add(openItem);
49
50           JMenuItem quitItem = new JMenuItem("Quit");
51           quitItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q,shortcut_mask));  // ctrl-q
52           quitItem.addActionListener(new ActionListener() {
53             public void actionPerformed(ActionEvent e)
54             { System.exit(0); }
55           });
56           fileMenu.add(quitItem);
57     }
```

Event04ImageViewer.java (3/3)

```
58
59      private File chooseImage()
60      {
61          int returnVal = fileChooser.showOpenDialog(null);
62          if (returnVal != JFileChooser.APPROVE_OPTION)
63          return null;  // cancelled
64          return fileChooser.getSelectedFile();
65      }
66
67      public static void main(String[] args)
68      {
69          new Event04ImageViewer();
70      }
71  }
```


ImagePanel.java (1/2)

```
12  public class ImagePanel extends JPanel
13  {
14      private int width, height; // of this panel
15      private BufferedImage panelImage;
16
17      public ImagePanel()
18      {
19          width = 360;
20          height = 240;
21          panelImage = null;
22          setBackground(Color.WHITE);
23      }
24
25      public void displayImage(File f)
26      {
27          BufferedImage image = loadImage(f);
28          if (image != null)
29          {
30              width = image.getWidth();
31              height = image.getHeight();
32              panelImage = image;
33              invalidate();
34              repaint();
35          }
36      }
```


ImagePanel.java (2/2)

```
38      private BufferedImage loadImage(File imageFile)
39      {
40          try {
41              BufferedImage image = ImageIO.read(imageFile);
42              if (image == null ||(image.getWidth() < 0))
43                  return null;
44              return image;
45          }
46          catch (IOException e)
47          {  return null; }
48      }  // end of loadImage()
49
50      public Dimension getPreferredSize()
51      {
52        return new Dimension(width, height);
53      }
54
55      public void paintComponent(Graphics g)
56      {
57          super.paintComponent(g);
58          Dimension size = getSize();
59          g.clearRect(0, 0, size.width, size.height);
60          if (panelImage != null)
61              g.drawImage(panelImage, 0, 0, null);
62      }  // end of paintComponent()
63
64  }  // end of ImagePanel class
```

Redefined Methods from JPanel

Event Listener Interfaces

Listener Interfaces

- ActionListener
- ItemListener
- MouseListener
- MouseMotionListener

MouseListener

It deals with mouse clicks over GUI components.

Its interface has 5 methods:

1. public void mouseClicked(MouseEvent e)
2. public void mousePressed(MouseEvent e)
3. public void mouseReleased(MouseEvent e)
4. public void mouseEntered(MouseEvent e)
5. public void mouseExited(MouseEvent e)

MouseMotionListener

For efficiency reasons, mouse movement events are dealt with by a separate listener.

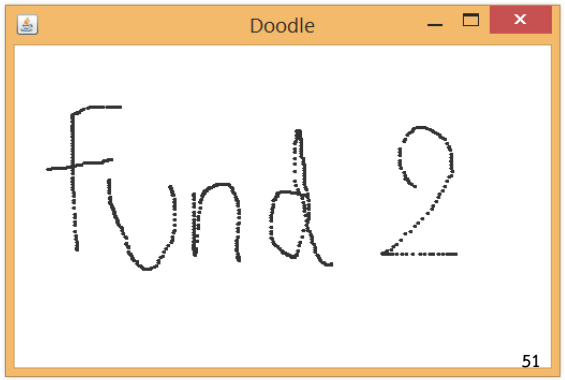
Its interface has 2 methods:

1. public void mouseDragged(MouseEvent e)
2. public void mouseMoved(MouseEvent e)

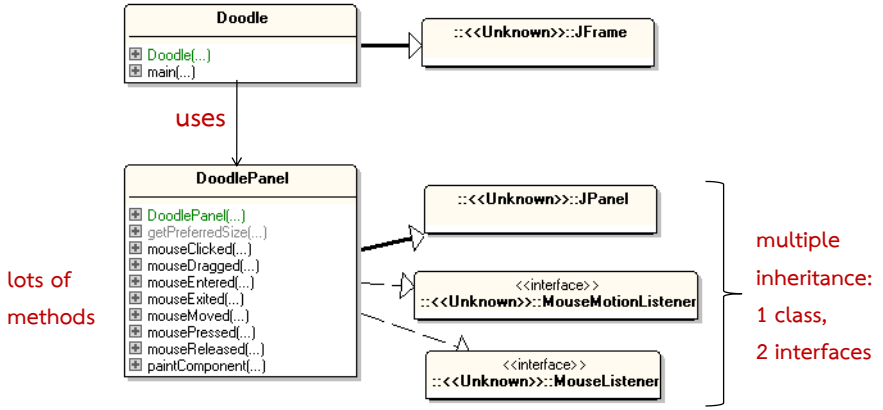
Mouse Example: Doodle

Doodle is a simple paint program for drawing onto a JPanel placed inside a JFrame.

```
Mouse pressed at (56,189)
Mouse released at (96,55)
Mouse pressed at (28,113)
Mouse released at (88,104)
Mouse pressed at (86,122)
Mouse released at (143,129)
Mouse pressed at (165,135)
Mouse released at (207,195)
Mouse pressed at (272,140)
Mouse released at (293,201)
Mouse pressed at (373,130)
Mouse released at (409,192)
```



Class Diagrams



Doodle.java

```
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Doodle extends JFrame
8  {
9      public Doodle()
10     {
11         super("Doodle");
12         Container c = getContentPane();
13         c.add( new DoodlePanel() );
14
15          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16          setSize(300, 200);
17          pack();
18          setLocationRelativeTo(null);
19          setVisible(true);
20        }
21
22     public static void main(String args[])
23     {
24         new Doodle();
25     }
26 }
```


DoodlePanel.java (1/2)

```
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class DoodlePanel extends JPanel implements MouseMotionListener, MouseListener
8  {
9      private static final int MAXPOINTS = 5000;
10     private Point[] points = new Point[MAXPOINTS];
11     private int nPoints = 0;
12
13     public DoodlePanel()
14     {
15         setBackground(Color.white);
16         addMouseMotionListener(this);
17         addMouseListener(this);
18     }
19
20     public Dimension getPreferredSize()
21     {
22         return new Dimension(500, 300);
23     }
24
25     public void paintComponent(Graphics g)
26     {
27         super.paintComponent(g);
28         for (int i = 0; i < nPoints; i++)
29           g.fillOval( points[i].x, points[i].y, 4, 4);
30     }
```

Redefined Methods from JPanel

DoodlePanel.java (2/2)

1. MouseMotionListener
2. MouseListener

```
32     public void mouseDragged( MouseEvent e)   (1)
33     {
34         if (nPoints < MAXPOINTS)
35             points[nPoints++] = new Point(e.getX(), e.getY());
36         repaint();
37     }
38
39     public void mouseMoved(MouseEvent e) {}   (1)
40
41     public void mousePressed( MouseEvent e)   (2)
42     {
43         System.out.println( "Mouse pressed at (" + e.getX() + "," + e.getY() + ")" );
44     }
45
46     public void mouseReleased( MouseEvent e)  (2)
47     {
48         System.out.println( "Mouse released at (" + e.getX() + "," + e.getY() + ")" );
49     }
50
51     public void mouseClicked(MouseEvent e) {}  // not needed
52     public void mouseEntered(MouseEvent e) {}  // not needed   (2)
53     public void mouseExited(MouseEvent e) {}   // not needed
54
55 } // end of DoodlePanel class
```


Mouse Example: A Simple Paint Program

```
1  // Fig. 12.34: PaintPanel.java
2  // Adapter class used to implement event handlers.
3  import java.awt.Point;
4  import java.awt.Graphics;
5  import java.awt.event.MouseEvent;
6  import java.awt.event.MouseMotionAdapter;
7  import java.util.ArrayList;
8  import javax.swing.JPanel;
9
10 public class PaintPanel extends JPanel
11 {
12    // list of Point references
13    private final ArrayList<Point> points = new ArrayList<>();
14
```

Fig. 12.34 | Adapter class used to implement event handlers. (Part 1 of 3.)

```
15    // set up GUI and register mouse event handler
16    public PaintPanel()
17    {
18       // handle frame mouse motion event
19       addMouseMotionListener(
20          new MouseMotionAdapter() // anonymous inner class
21          {
22             // store drag coordinates and repaint
23             @Override
24             public void mouseDragged(MouseEvent event)
25             {
26                points.add(event.getPoint());
27                repaint(); // repaint JFrame
28             }
29          }
30       );
31    }
32
33    // draw ovals in a 4-by-4 bounding box at specified locations on window
34    @Override
35    public void paintComponent(Graphics g)
36    {
37       super.paintComponent(g); // clears drawing area
38
```

Fig. 12.34 | Adapter class used to implement event handlers. (Part 2 of 3.)


```
39       // draw all points
40       for (Point point : points)
41          g.fillOval(point.x, point.y, 4, 4);
42    }
43 } // end class PaintPanel
```

Fig. 12.34 | Adapter class used to implement event handlers. (Part 3 of 3.)

`JPanel` Subclass for Drawing with the Mouse (cont.)

- Class `Point` (package `java.awt`) represents an *x*-*y* coordinate.
  - We use objects of this class to store the coordinates of each mouse drag event.
- Class `Graphics` is used to draw.
- `MouseEvent` method `getPoint` obtains the `Point` where the event occurred.
- Method `repaint` (inherited from `Component`) indicates that a `Component` should be refreshed on the screen as soon as possible.

`JPanel` Subclass for Drawing with the Mouse (cont.)

- `Graphics` method `fillOval` draws a solid oval.
  - Four parameters represent a rectangular area (called the bounding box) in which the oval is displayed.
  - The first two are the upper-left x-coordinate and the upper-left *y*-coordinate of the rectangular area.
  - The last two represent the rectangular area's width and height.
- Method `fillOval` draws the oval so it touches the middle of each side of the rectangular area.

```java
// Fig. 12.35: Painter.java
// Testing PaintPanel.
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class Painter
{
   public static void main(String[] args)
   {
      // create JFrame
      JFrame application = new JFrame("A simple paint program");

      PaintPanel paintPanel = new PaintPanel();
      application.add(paintPanel, BorderLayout.CENTER);

      // create a label and place it in SOUTH of BorderLayout
      application.add(new JLabel("Drag the mouse to draw"),
         BorderLayout.SOUTH);

      application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      application.setSize(400, 200);
      application.setVisible(true);
   }
} // end class Painter
```

Fig. 12.35 | Testing `PaintPanel`. (Part 1 of 2.)

Fig. 12.35 | Testing `PaintPanel`. (Part 2 of 2.)

GUI and Graphics

Programming Fundamentals II

4.14 (Optional) GUI and Graphics Case Study: Creating Simple Drawings

- Java's coordinate system is a scheme for identifying points on the screen.
- The upper-left corner of a GUI component has the coordinates (0, 0).
- A coordinate pair is composed of an x-coordinate (the horizontal coordinate) and a y-coordinate (the vertical coordinate).
- The *x*-coordinate is the horizontal location (from left to right).
- The *y*-coordinate is the vertical location (from top to bottom).
- The x-axis describes every horizontal coordinate, and the y-axis every vertical coordinate.
- Coordinate units are measured in pixels. The term pixel stands for "picture element." A pixel is a display monitor's smallest unit of resolution.

Reference P64 – P110

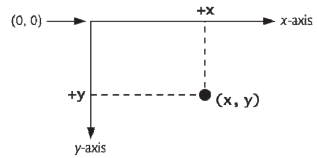


Fig. 4.17 | Java coordinate system. Units are measured in pixels.

Reference P65 – P110


```
1  // Fig. 4.18: DrawPanel.java
2  // Using drawLine to connect the corners of a panel.
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // draws an X from the corners of the panel
9     public void paintComponent( Graphics g )
10    {
11       // call paintComponent to ensure the panel displays correctly
12       super.paintComponent( g );
13
14       int width = getWidth(); // total width
15       int height = getHeight(); // total height
16
17       // draw a line from the upper-left to the lower-right
18       g.drawLine( 0, 0, width, height );
19
20       // draw a line from the lower-left to the upper-right
21       g.drawLine( 0, height, width, 0 );
22    } // end method paintComponent
23 } // end class DrawPanel
```

Import the classes `Graphics` and `JPanel` for use in this source code file.

`DrawPanel` inherits the existing capabilities of class `JPanel`

`paintComponent` *must* be displayed as shown here

This should be the first statement in method `paintComponent`

Determines the width and height of the `DrawPanel` with inherited methods

Draws a line from the top-left to the bottom-right of the `DrawPanel`

Draws a line from the bottom-left to the top-right of the `DrawPanel`

Fig. 4.18 | Using `drawLine` to connect the corners of a panel.


```
1  // Fig. 4.19: DrawPanelTest.java
2  // Application to display a DrawPanel.
3  import javax.swing.JFrame;
4
5  public class DrawPanelTest
6  {
7     public static void main( String[] args )
8     {
9        // create a panel that contains our drawing
10       DrawPanel panel = new DrawPanel();
11
12       // create a new frame to hold the panel
13       JFrame application = new JFrame();
14
15       // set the frame to exit when it is closed
16       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
17
18       application.add( panel ); // add the panel to the frame
19       application.setSize( 250, 250 ); // set the size of the frame
20       application.setVisible( true ); // make the frame visible
21    } // end main
22 } // end class DrawPanelTest
```

Imports class `JFrame` for use in this source code file

Creates a `JFrame` in which the `DrawPanel` will be displayed

Terminate application when window closes

Attach the `DrawPanel` to the `JFrame`

Sets the size of the `JFrame`

Displays the `JFrame` on the screen

Fig. 4.19 | Creating `JFrame` to display `DrawPanel`. (Part 1 of 2.)

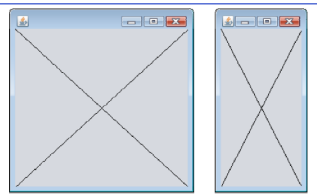


Fig. 4.19 | Creating `JFrame` to display `DrawPanel`. (Part 2 of 2.)

4.14 (Optional) GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- The keyword `extends` creates a so-called inheritance relationship.
- The class from which `DrawPanel` inherits, `JPanel`, appears to the right of keyword `extends`.
- In this inheritance relationship, `JPanel` is called the superclass and `DrawPanel` is called the subclass.

4.14 (Optional) GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- `JPanel` has a `paintComponent` method, which the system calls every time it needs to display the `JPanel`.
- The first statement in every `paintComponent` method you create should always be

```
super.paintComponent( g );
```

- `JPanel` methods `getWidth` and `getHeight` return the `JPanel`'s width and height, respectively.
- `Graphics` method `drawLine` draws a line between two points represented by its four arguments. The first two are the *x*- and *y*-coordinates for one endpoint, and the last two arguments are the coordinates for the other endpoint.

4.14 (Optional) GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- To display the `DrawPanel` on the screen, place it in a window.
- Create a window with an object of class `JFrame`.
- `JFrame` method `setDefaultCloseOperation` with the argument `JFrame.EXIT_ON_CLOSE` indicates that the application should terminate when the user closes the window.
- `JFrame`'s `add` method attaches the `DrawPanel` (or any other GUI component) to a `JFrame`.
- `JFrame` method `setSize` takes two parameters that represent the width and height of the `JFrame`, respectively.
- `JFrame` method `setVisible` with the argument `true` displays the `JFrame`.
- When a `JFrame` is displayed, the `DrawPanel`'s `paintComponent` method is implicitly called

5.10 (Optional) GUI and Graphics Case Study: Drawing Rectangles and Ovals

- `Graphics` methods `drawRect` and `drawOval`
- Method `drawRect` requires four arguments. The first two represent the *x*- and *y*-coordinates of the upper-left corner of the rectangle; the next two represent the rectangle's width and height.
- To draw an oval, method `drawOval` creates an imaginary rectangle called a **bounding rectangle** and places inside it an oval that touches the midpoints of all four sides.
- Method `drawOval` requires the same four arguments as method `drawRect`. The arguments specify the position and size of the bounding rectangle for the oval.

```
1  // Fig. 5.26: Shapes.java
2  // Demonstrates drawing different shapes.
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class Shapes extends JPanel
7  {
8     private int choice; // user's choice of which shape to draw
9
10    // constructor sets the user's choice
11    public Shapes( int userChoice )
12    {
13       choice = userChoice;
14    } // end Shapes constructor
15
```

Fig. 5.26 | Drawing a cascade of shapes based on the user's choice. (Part 1 of 2.)


```
16    // draws a cascade of shapes starting from the top-left corner
17    public void paintComponent( Graphics g )
18    {
19       super.paintComponent( g );
20
21       for ( int i = 0; i < 10; i++ )
22       {
23          // pick the shape based on the user's choice
24          switch ( choice )
25          {
26             case 1: // draw rectangles
27                g.drawRect( 10 + i * 10, 10 + i * 10,
28                   50 + i * 10, 50 + i * 10 );
29                break;
30             case 2: // draw ovals
31                g.drawOval( 10 + i * 10, 10 + i * 10,
32                   50 + i * 10, 50 + i * 10 );
33                break;
34          } // end switch
35       } // end for
36    } // end method paintComponent
37 } // end class Shapes
```

Fig. 5.26 | Drawing a cascade of shapes based on the user's choice. (Part 2 of 2.)


```
1  // Fig. 5.27: ShapesTest.java
2  // Test application that displays class Shapes.
3  import javax.swing.JFrame;
4  import javax.swing.JOptionPane;
5
6  public class ShapesTest
7  {
8     public static void main( String[] args )
9     {
10       // obtain user's choice
11       String input = JOptionPane.showInputDialog(
12          "Enter 1 to draw rectangles\n" +
13          "Enter 2 to draw ovals" );
14
15       int choice = Integer.parseInt( input ); // convert input to int
16
```

Fig. 5.27 | Obtaining user input and creating a `JFrame` to display `Shapes`. (Part 1 of 4.)


```
17       // create the panel with the user's input
18       Shapes panel = new Shapes( choice );
19
20       JFrame application = new JFrame(); // creates a new JFrame
21
22       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
23       application.add( panel ); // add the panel to the frame
24       application.setSize( 300, 300 ); // set the desired size
25       application.setVisible( true ); // show the frame
26    } // end main
27 } // end class ShapesTest
```

Fig. 5.27 | Obtaining user input and creating a `JFrame` to display `Shapes`. (Part 2 of 4.)

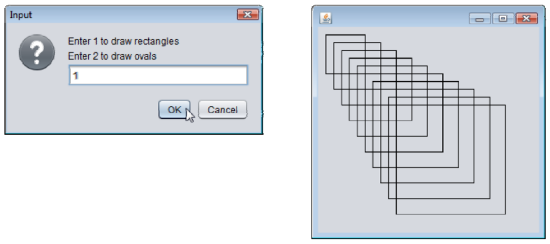


Fig. 5.27 | Obtaining user input and creating a `JFrame` to display `Shapes`. (Part 3 of 4.)

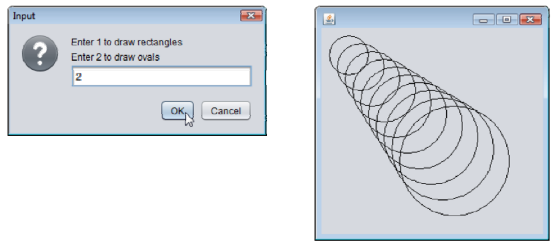


Fig. 5.27 | Obtaining user input and creating a `JFrame` to display `Shapes`. (Part 4 of 4.)

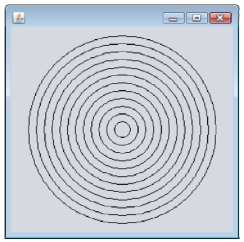


Fig. 5.28 | Drawing concentric circles.

6.13 (Optional) GUI and Graphics Case Study: Colors and Filled Shapes

- Colors displayed on computer screens are defined by their red, green, and blue components (called RGB values) that have integer values from 0 to 255.
- The higher the value of a component color, the richer that shade will be in the final color.
- Java uses class `Color` in package `java.awt` to represent colors using their RGB values.
- Class `Color` contains 13 predefined `static Color` objects—`BLACK`, `BLUE`, `CYAN`, `DARK_GRAY`, `GRAY`, `GREEN`, `LIGHT_GRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `WHITE` and `YELLOW`.

6.13 (Optional) GUI and Graphics Case Study: Colors and Filled Shapes (Cont.)

- Class `Color` also contains a constructor of the form:
  - `public Color( int r, int g, int b )`
- so you can create custom colors by specifying the red, green and blue component values.
- `Graphics` methods `fillRect` and `fillOval` draw filled rectangles and ovals, respectively.
- `Graphics` method `setColor` sets the current drawing color.


```
1  // Fig. 6.11: DrawSmiley.java
2  // Demonstrates filled shapes.
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class DrawSmiley extends JPanel
8  {
9     public void paintComponent( Graphics g )
10    {
11       super.paintComponent( g );
12
13       // draw the face
14       g.setColor( Color.YELLOW );
15       g.fillOval( 10, 10, 200, 200 );
16
17       // draw the eyes
18       g.setColor( Color.BLACK );
19       g.fillOval( 55, 65, 30, 30 );
20       g.fillOval( 135, 65, 30, 30 );
21
22       // draw the mouth
23       g.fillOval( 50, 110, 120, 60 );
24
```

Fig. 6.11 | Drawing a smiley face using colors and filled shapes. (Part 1 of 2.)


```
25       // "touch up" the mouth into a smile
26       g.setColor( Color.YELLOW );
27       g.fillRect( 50, 110, 120, 30 );
28       g.fillOval( 50, 120, 120, 40 );
29    } // end method paintComponent
30 } // end class DrawSmiley
```

Fig. 6.11 | Drawing a smiley face using colors and filled shapes. (Part 2 of 2.)


```
1  // Fig. 6.12: DrawSmileyTest.java
2  // Test application that displays a smiley face.
3  import javax.swing.JFrame;
4
5  public class DrawSmileyTest
6  {
7     public static void main( String[] args )
8     {
9        DrawSmiley panel = new DrawSmiley();
10       JFrame application = new JFrame();
11
12       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13       application.add( panel );
14       application.setSize( 230, 250 );
15       application.setVisible( true );
16    } // end main
17 } // end class DrawSmileyTest
```

Fig. 6.12 | Creating `JFrame` to display a smiley face. (Part 1 of 2.)

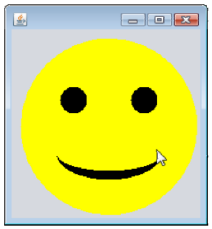


Fig. 6.12 | Creating `JFrame` to display a smiley face. (Part 2 of 2.)

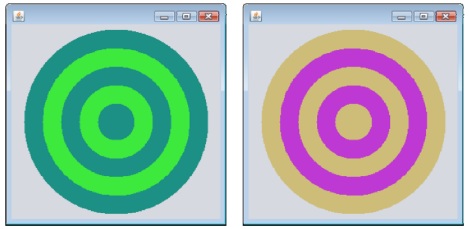


Fig. 6.13 | A bull's-eye with two alternating, random colors.

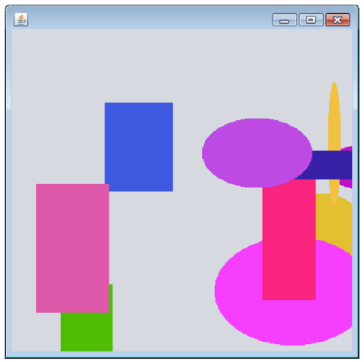


Fig. 6.14 | Randomly generated shapes.


```
1  // Fig. 7.25: DrawRainbow.java
2  // Demonstrates using colors in an array.
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class DrawRainbow extends JPanel
8  {
9     // define indigo and violet
10    private final static Color VIOLET = new Color( 128, 0, 128 );
11    private final static Color INDIGO = new Color( 75, 0, 130 );
12
13    // colors to use in the rainbow, starting from the innermost
14    // The two white entries result in an empty arc in the center
15    private Color[] colors =
16       { Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
17         Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED };
18
19    // constructor
20    public DrawRainbow()
21    {
22       setBackground( Color.WHITE ); // set the background to white
23    } // end DrawRainbow constructor
24
```

Fig. 7.25 | Drawing a rainbow using arcs and an array of colors. (Part 1 of 2.)

```
25     // draws a rainbow using concentric arcs
26     public void paintComponent( Graphics g )
27     {
28        super.paintComponent( g );
29
30        int radius = 20; // radius of an arc
31
32        // draw the rainbow near the bottom-center
33        int centerX = getWidth() / 2;
34        int centerY = getHeight() - 10;
35
36        // draws filled arcs starting with the outermost
37        for ( int counter = colors.length; counter > 0; counter-- )
38        {
39           // set the color for the current arc
40           g.setColor( colors[ counter - 1 ] );
41
42           // fill the arc from 0 to 180 degrees
43           g.fillArc( centerX - counter * radius,
44              centerY - counter * radius,
45              counter * radius * 2, counter * radius * 2, 0, 180 );
46        } // end for
47     } // end method paintComponent
48  } // end class DrawRainbow
```

Fig. 7.25 | Drawing a rainbow using arcs and an array of colors. (Part 2 of 2.)


```
1  // Fig. 7.26: DrawRainbowTest.java
2  // Test application to display a rainbow.
3  import javax.swing.JFrame;
4
5  public class DrawRainbowTest
6  {
7     public static void main( String[] args )
8     {
9        DrawRainbow panel = new DrawRainbow();
10       JFrame application = new JFrame();
11
12       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13       application.add( panel );
14       application.setSize( 400, 250 );
15       application.setVisible( true );
16    } // end main
17 } // end class DrawRainbowTest
```

Fig. 7.26 | Creating `JFrame` to display a rainbow. (Part 1 of 2.)

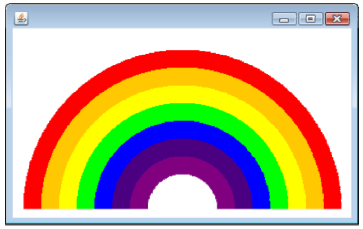


Fig. 7.26 | Creating `JFrame` to display a rainbow. (Part 2 of 2.)

7.15 (Optional) GUI and Graphics Case Study: Drawing Arcs

- Drawing arcs in Java is similar to drawing ovals—an arc is simply a section of an oval.
- `Graphics` method fillArc draws a filled arc.
- Method `fillArc` requires six parameters.
  - The first four represent the bounding rectangle in which the arc will be drawn.
  - The fifth parameter is the starting angle on the oval, and the sixth specifies the sweep, or the amount of arc to cover.
  - Starting angle and sweep are measured in degrees, with zero degrees pointing right.
  - A positive sweep draws the arc counterclockwise.
- Method drawArc requires the same parameters as `fillArc`, but draws the edge of the arc rather than filling it.
- Method setBackground changes the background color of a GUI component.

8.16 (Optional) GUI and Graphics Case Study: Using Objects with Graphics

- The next example stores information about the displayed shapes so that we can reproduce them each time the system calls `paintComponent`.
- We'll make "smart" shape classes that can draw themselves by using a `Graphics` object.
- Figure 8.18 declares class `MyLine`, which has all these capabilities.
- Method `paintComponent` in class `DrawPanel` iterates through an array of `MyLine` objects.
  - Each iteration calls the `draw` method of the current `MyLine` object and passes it the `Graphics` object for drawing on the panel.


```
1  // Fig. 8.18: MyLine.java
2  // MyLine class represents a line.
3  import java.awt.Color;
4  import java.awt.Graphics;
5
6  public class MyLine
7  {
8     private int x1; // x-coordinate of first endpoint
9     private int y1; // y-coordinate of first endpoint
10    private int x2; // x-coordinate of second endpoint
11    private int y2; // y-coordinate of second endpoint
12    private Color myColor; // color of this shape
13
14    // constructor with input values
15    public MyLine( int x1, int y1, int x2, int y2, Color color )
16    {
17       this.x1 = x1; // set x-coordinate of first endpoint
18       this.y1 = y1; // set y-coordinate of first endpoint
19       this.x2 = x2; // set x-coordinate of second endpoint
20       this.y2 = y2; // set y-coordinate of second endpoint
21       myColor = color; // set the color
22    } // end MyLine constructor
23
```

Fig. 8.18 | `MyLine` class represents a line. (Part 1 of 2.)


```
24    // Draw the line in the specified color
25    public void draw( Graphics g )
26    {
27       g.setColor( myColor );
28       g.drawLine( x1, y1, x2, y2 );
29    } // end method draw
30 } // end class MyLine
```

Fig. 8.18 | `MyLine` class represents a line. (Part 2 of 2.)


```
1  // Fig. 8.19: DrawPanel.java
2  // Program that uses class MyLine
3  // to draw random lines.
4  import java.awt.Color;
5  import java.awt.Graphics;
6  import java.util.Random;
7  import javax.swing.JPanel;
8
9  public class DrawPanel extends JPanel
10 {
11    private Random randomNumbers = new Random();
12    private MyLine[] lines; // array of lines
13
14    // constructor, creates a panel with random shapes
15    public DrawPanel()
16    {
17       setBackground( Color.WHITE );
18
19       lines = new MyLine[ 5 + randomNumbers.nextInt( 5 ) ];
20
```

Fig. 8.19 | Creating random `MyLine` objects. (Part 1 of 3.)

```
21        // create lines
22        for ( int count = 0; count < lines.length; count++ )
23        {
24           // generate random coordinates
25           int x1 = randomNumbers.nextInt( 300 );
26           int y1 = randomNumbers.nextInt( 300 );
27           int x2 = randomNumbers.nextInt( 300 );
28           int y2 = randomNumbers.nextInt( 300 );
29
30           // generate a random color
31           Color color = new Color( randomNumbers.nextInt( 256 ),
32              randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );
33
34           // add the line to the list of lines to be displayed
35           lines[ count ] = new MyLine( x1, y1, x2, y2, color );
36        } // end for
37     } // end DrawPanel constructor
38
```

Fig. 8.19 | Creating random `MyLine` objects. (Part 2 of 3.)


```
39     // for each shape array, draw the individual shapes
40     public void paintComponent( Graphics g )
41     {
42        super.paintComponent( g );
43
44        // draw the lines
45        for ( MyLine line : lines )
46           line.draw( g );
47     } // end method paintComponent
48  } // end class DrawPanel
```

Fig. 8.19 | Creating random `MyLine` objects. (Part 3 of 3.)


```
1  // Fig. 8.20: TestDraw.java
2  // Creating a JFrame to display a DrawPanel.
3  import javax.swing.JFrame;
4
5  public class TestDraw
6  {
7     public static void main( String[] args )
8     {
9        DrawPanel panel = new DrawPanel();
10       JFrame application = new JFrame();
11
12       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13       application.add( panel );
14       application.setSize( 300, 300 );
15       application.setVisible( true );
16    } // end main
17 } // end class TestDraw
```

Fig. 8.20 | Creating a `JFrame` to display a `DrawPanel`. (Part 1 of 2.)

9.8 (Optional) GUI and Graphics Case Study: Displaying Text and Images Using Labels

- Labels are a convenient way of identifying GUI components on the screen and keeping the user informed about the current state of the program.
- A `JLabel` (from package `javax.swing`) can display text, an image or both.
- The example in Fig. 9.13 demonstrates several `JLabel` features, including a plain text label, an image label and a label with both text and an image.

```
1  // Fig 9.13: LabelDemo.java
2  // Demonstrates the use of labels.
3  import java.awt.BorderLayout;
4  import javax.swing.ImageIcon;
5  import javax.swing.JLabel;
6  import javax.swing.JFrame;
7
8  public class LabelDemo
9  {
10    public static void main( String[] args )
11    {
12       // Create a label with plain text
13       JLabel northLabel = new JLabel( "North" );
14
15       // create an icon from an image so we can put it on a JLabel
16       ImageIcon labelIcon = new ImageIcon( "GUItip.gif" );
17
18       // create a label with an Icon instead of text
19       JLabel centerLabel = new JLabel( labelIcon );
20
21       // create another label with an Icon
22       JLabel southLabel = new JLabel( labelIcon );
23
```

Fig. 9.13 | `JLabel` with text and with images. (Part 1 of 3.)


```
24       // set the label to display text (as well as an icon)
25       southLabel.setText( "South" );
26
27       // create a frame to hold the labels
28       JFrame application = new JFrame();
29
30       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
31
32       // add the labels to the frame; the second argument specifies
33       // where on the frame to add the label
34       application.add( northLabel, BorderLayout.NORTH );
35       application.add( centerLabel, BorderLayout.CENTER );
36       application.add( southLabel, BorderLayout.SOUTH );
37
38       application.setSize( 300, 300 ); // set the size of the frame
39       application.setVisible( true ); // show the frame
40    } // end main
41 } // end class LabelDemo
```

Fig. 9.13 | `JLabel` with text and with images. (Part 2 of 3.)

Fig. 9.13 | `JLabel` with text and with images. (Part 3 of 3.)

9.8 (Optional) GUI and Graphics Case Study: Displaying Text and Images Using Labels (Cont.)

- An `ImageIcon` represents an image that can be displayed on a `JLabel`.
- The constructor for `ImageIcon` receives a `String` that specifies the path to the image.
- `ImageIcon` can load images in GIF, JPEG and PNG image formats.
- `JLabel` method `setText` changes the text the label displays.

9.8 (Optional) GUI and Graphics Case Study: Displaying Text and Images Using Labels (Cont.)

- An overloaded version of method `add` that takes two parameters allows you to specify the GUI component to add to a `JFrame` and the location in which to add it.
  - The first parameter is the component to attach.
  - The second is the region in which it should be placed.
- Each `JFrame` has a layout to position GUI components.
  - Default layout for a `JFrame` is `BorderLayout`.
  - Five regions—`NORTH` (top), `SOUTH` (bottom), `EAST` (right side), `WEST` (left side) and `CENTER` (constants in class `BorderLayout`)
  - Each region is declared as a constant in class `BorderLayout`.
- When calling method `add` with one argument, the `JFrame` places the component in the `BorderLayout`'s `CENTER` automatically.

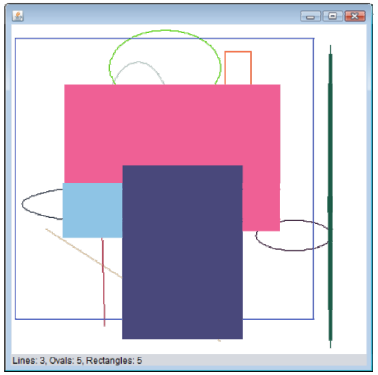


Fig. 9.14 | `JLabel` displaying shape statistics.

10.8 (Optional) GUI and Graphics Case Study: Drawing with Polymorphism

- Shape classes have many similarities.
- Using inheritance, we can "factor out" the common features from all three classes and place them in a single shape superclass.
- Then, using variables of the superclass type, we can manipulate objects of all three shape types polymorphically.
- Removing the redundancy in the code will result in a smaller, more flexible program that is easier to maintain.

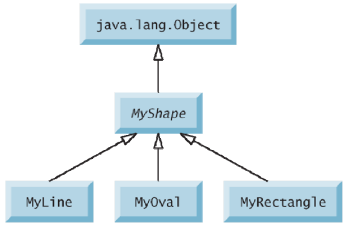


Fig. 10.17 | `MyShape` hierarchy.

10.8 (Optional) GUI and Graphics Case Study: Drawing with Polymorphism (Cont.)

- Class `MyBoundedShape` can be used to factor out the common features of classes `MyOval` and `MyRectangle`.

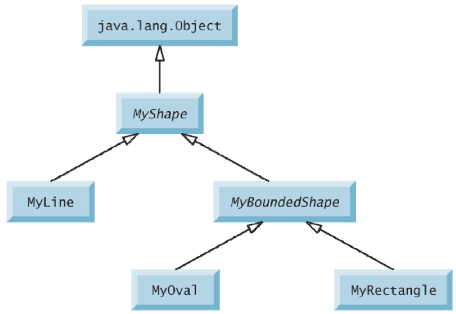


Fig. 10.18 | `MyShape` hierarchy with `MyBoundedShape`.

More Information

1. The Java tutorial "Creating a GUI with JFC/Swing".
2. Sub-topics include:
   - Getting Started with Swing
   - Swing Features and Concepts
   - Using Swing Components
   - Laying Out Components Within a Container
   - Writing Event Listeners
3. Weighing in at 1621 pages!!:
   - Graphic Java 2: Mastering the JFC Volume 2: Swing, David M. Geary, Sun MicroSystems, 1999, 3rd edition
4. The other volumes:
   - AWT (vol. 1), Advanced Swing (vol. 3) Java 2D (vol. 4)

END