Vietnam National University of HCMC
International University
School of Computer Science and Engineering

Graphic User Interface

(IT069IU)

Le Duy Tan, Ph.D.

ldtan@hcmiu.edu.vn

leduytanit.com

Previously,

- **Polymorphism**
  - Method overriding in Inheritance
    - Zoo Example
  - **Abstraction**
    - Abstract Class
    - Abstract Method
    - Examples:
      - Zoo Example
      - Company Payroll Example
  - **Interface**
    - Interface in real life examples
    - Upgrade Company Payroll with Invoices Example
  - Abstract vs Interface vs Composition

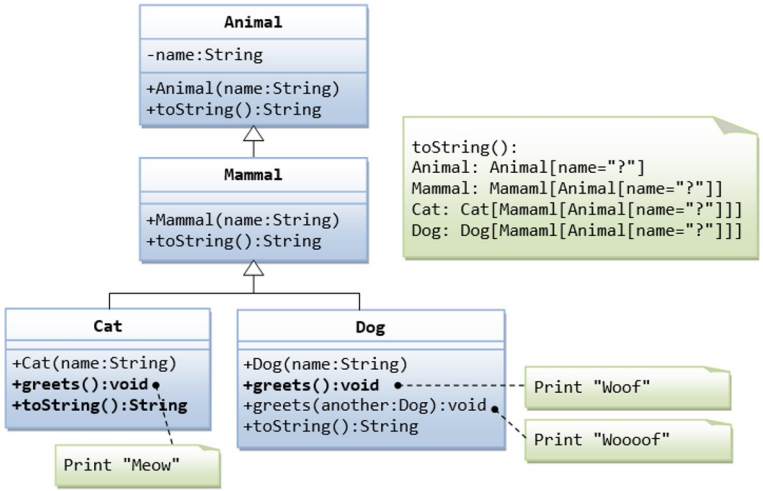
Agenda

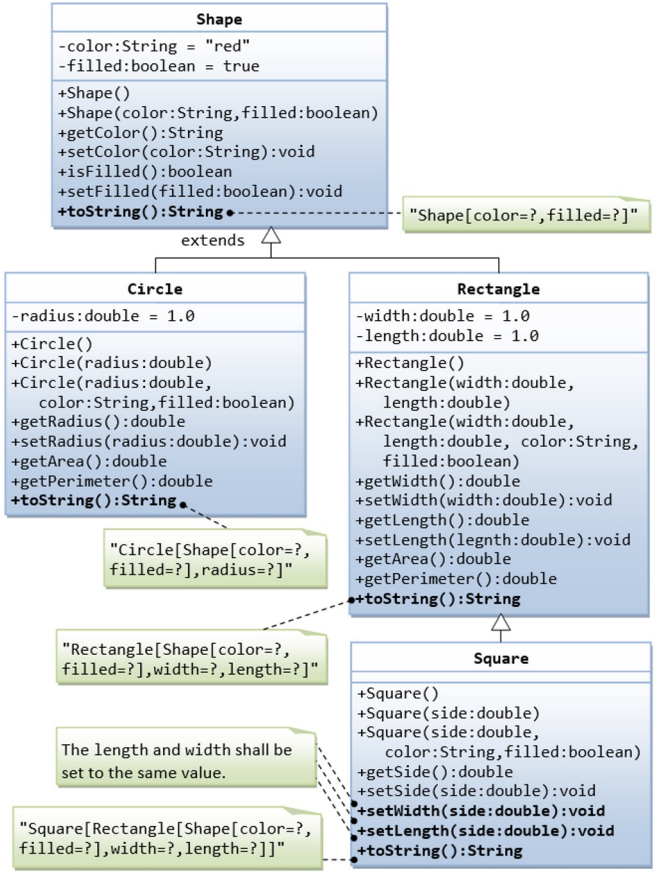
- **Mid-term Exam**
  - Possible Topics
  - Format
- **GUI** in Java
  - Swing Components
  - Swing vs JavaFX vs AWT
  - JOption-Panel (DialogBox)
  - Java 2D API
    - Java Coordinate System
    - Draw Lines
    - Draw Rectangles and Ovals
    - Draw Colors
    - Draw Polygons and Polylines
  - Text Fields, Buttons and Menu Nagivations.
  - Event Handler (Mouse & Keyboard)
  - Layout:
    - BorderLayout Frame & GridLayoutFrame

Mid-term Example

- Time: **90 minutes**
- Date: Check your EdusoftWeb
- Test Method: **Offline**, closed paper, and write on exam paper.
- Language: **You must use Java!**
- **3 Questions:**
  - **Question 1**: Theory questions
    - Three sub questions:
      - Explain keywords in Java.
      - Explain the difference between two things.
      - Your answers should be from two to four sentences.
  - **Question 2 & 3**: Coding questions
    - Question 2: Given an UML Class Diagram, main method and expected output, you need to write classes to satisfy them.
    - Question 3: Given a problem statement, you need to write classes and a test class (with main method).

Question 2: Example of Coding Question





- Given an UML diagram, a main method and expected output:
  - **Your Task:** Write two or three classes.

Question 3: Example of Problem Statement

- You need to create a system to pay employees of a company.
- There are three types of employees:
  - Developer
  - Designer
  - Manager
- Developer has a name and his salary depends on how many projects he completed. And 30% of the project reward will go to him. A developer can take a number of projects
- Designer has a name and her salary depends on how many projects she completed. And 20% of the project reward will go to her salary. A designer can take a number of projects.
- Manager has a name and his salary depends how many people he manage and their performance. His salary will be equal to all people he manages + 50% of the projects reward will go to him. A manager can manage a number of developers and managers.
- A project can be a name, cost and reward.
- **Your Task:** Write classes for Employee, Developer, Designer, Manager and Project.

Question 3: Example of Testing Your Classes

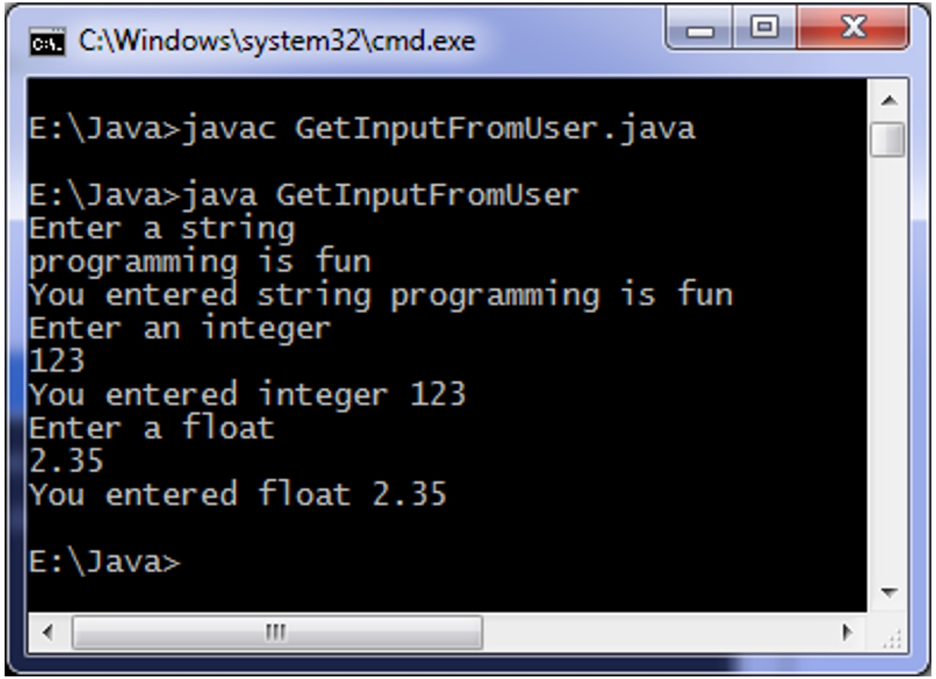
- In the previous project, write a test classes to follow the scenario:
  - Create 2 Developers Tom and Charles
  - Create 1 Designer Chloe
  - Create 1 Manager Jerry
  - Create two projects Alpha and Beta. The reward for Alpha is $500 and reward for Beta is $200.
  - Tom is responsible for Alpha, Charles is responsible for Beta
  - Chloe is responsible for designing Beta
  - Jerry manage Tom, Charles and Chloe
  - Print out salary for each person above.
- **Your Task:** Write a testing class with main method to test those scenario.

CLI vs GUI

Command Line Interface vs **Graphical User Interface**

Command Line Interface

- So far, every input and output of our Java programs is through command window (terminal/cmd/bash).

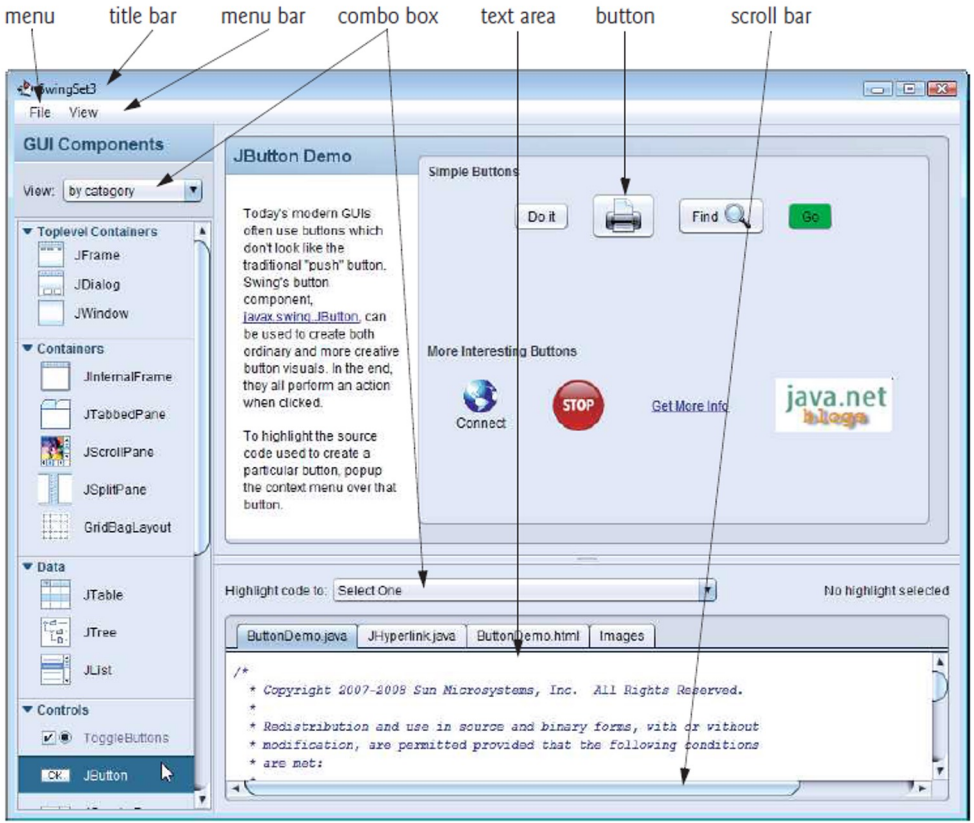


Graphic User Interface (GUI)

Graphic User Interface (GUI)

- A graphical user interface (GUI) presents a user-friendly mechanism for interacting with an application.
  - Pronounced “GOO-ee”
  - Gives an application a distinctive “look” and “feel.”
  - Consistent, intuitive user-interface components give users a sense of familiarity
  - Learn new applications more quickly and use them more productively.
  - User interacts via the mouse, the keyboard or another form of input, such as voice recognition.

Sample GUI of Java's Swing GUI APIs



SwingSet3 application demonstrates many of Java's Swing GUI components.

https://mlapshin.com/index.php/projects/swingset3/

Overview of Swing Components

Component	Description
`JLabel`	Displays *uneditable text* and/or icons.
`JTextField`	Typically *receives input* from the user.
`JButton`	Triggers an event when clicked with the mouse.
`JCheckBox`	Specifies an option that can be *selected* or *not selected*.
`JComboBox`	A *drop-down list of items* from which the *user* can make a *selection*.
`JList`	A *list of items* from which the user can make a *selection* by *clicking* on *any one* of them. *Multiple* elements *can* be selected.
`JPanel`	An area in which *components* can be *placed* and *organized*.

Java GUI library

- Java's GUI library:
  - **Abstract Window Toolkit (AWT)**:
    - Package java.awt
    - Is the very foundation of swing, it performs well but is lacking in advanced components.
    - The look of UI differs between platforms (MacOS, Windows,...).
  - **Swing technology**:
    - Mature and popular.
    - Based on AWT packages.
    - A wider range of UI components and a bigger library of GUI elements.
    - Only for desktop applications.
    - Lack consistency with MVC.
  - **JavaFX technology**:
    - New and modern.
    - The latest flagship of Java/Oracle. promising to be the facto standard in developing rich desktop or web applications.
    - Less component as compared to Swing APIs.
    - Easier to build Desktop, Website and Mobile applications.
    - Easier to add animation and special effects.
    - Rich new toolkit and expected to grow in the future.
    - Friendly with MVC pattern.

Displaying Text in a Dialog Box

- **Dialog boxes** are windows in which programs display important messages to users.
- Class **JOption-Pane** provides prebuilt dialog boxes that enable programs to display windows containing messages—such windows are called message dialogs.

Dialog Box in action

```
1  // Fig. 3.12: Dialog1.java
2  // Using JOptionPane to display multiple lines in a dialog box.
3  import javax.swing.JOptionPane;
4
5  public class Dialog1
6  {
7     public static void main(String[] args)
8     {
9        // display a dialog with a message
10       JOptionPane.showMessageDialog(null, "Welcome to Java");
11    }
12 } // end class Dialog1
```

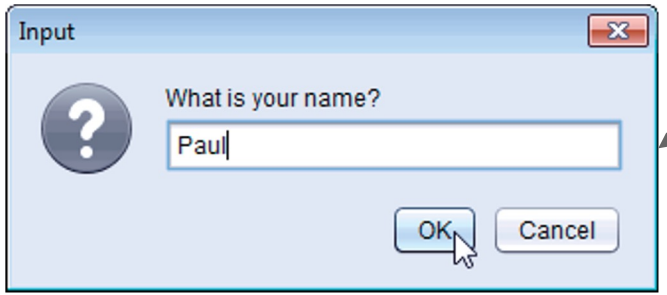
The first argument refers to that window (known as the parent window) and causes the dialog to appear centered over the app's window. If the first argument
is **null**, the dialog box is displayed at the center of your screen.
The second argument is the String to display in the dialog box.

Entering Text in a Dialog

```
1  // Fig. 3.13: NameDialog.java
2  // Obtaining user input from a dialog.
3  import javax.swing.JOptionPane;
4
5  public class NameDialog
6  {
7     public static void main(String[] args)
8     {
9        // prompt user to enter name
10       String name = JOptionPane.showInputDialog("What is your name?");
11
12       // create the message
13       String message =
14          String.format("Welcome, %s, to Java Programming!", name);
15
16       // display the message to welcome the user by name
17       JOptionPane.showMessageDialog(null, message);
18    } // end main
19 } // end class NameDialog
```

Method format works like method System.out.printf, except that format returns the formatted String rather than displaying it in a command window.

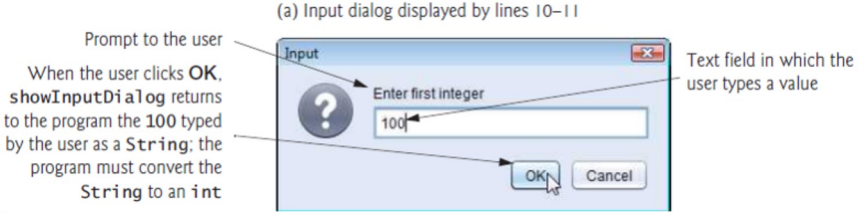
[Q&A] What happens if you click Cancel?



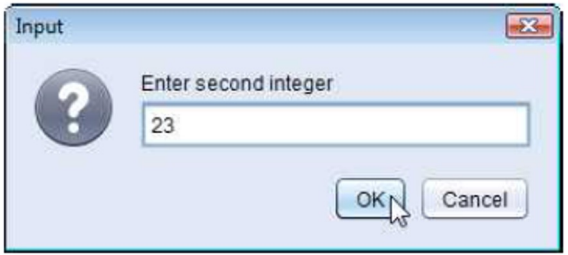
Simple GUI-Based Input/Output with JOptionPane

```
1  // Fig. 12.2: Addition.java
2  // Addition program that uses JOptionPane for input and output.
3  import javax.swing.JOptionPane;
4
5  public class Addition
6  {
7     public static void main(String[] args)
8     {
9        // obtain user input from JOptionPane input dialogs
10       String firstNumber =
11          JOptionPane.showInputDialog("Enter first integer");
12       String secondNumber =
13          JOptionPane.showInputDialog("Enter second integer");
14
15       // convert String inputs to int values for use in a calculation
16       int number1 = Integer.parseInt(firstNumber);
17       int number2 = Integer.parseInt(secondNumber);
18
19       int sum = number1 + number2;
20
21       // display result in a JOptionPane message dialog
22       JOptionPane.showMessageDialog(null, "The sum is " + sum,
23          "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE);
24    }
25 } // end class Addition
```

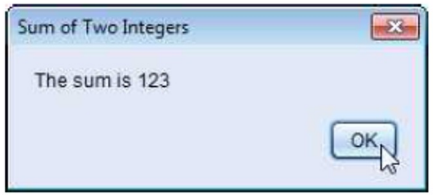
(a) Input dialog displayed by lines 10–11



(b) Input dialog displayed by lines 12–13



(c) Message dialog displayed by lines 22–23—When the user clicks **OK**, the message dialog is dismissed (removed from the screen)

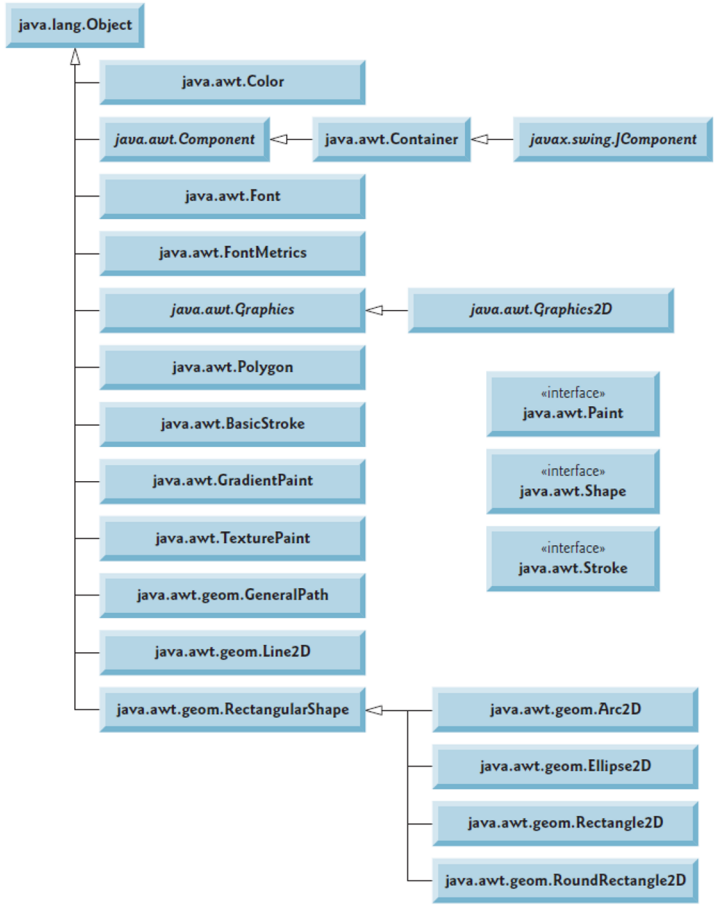


PLAIN_MESSAGE is the type of message dialog to display which does not display an icon to the left of the message.

JOptionPane static constants for message dialogs

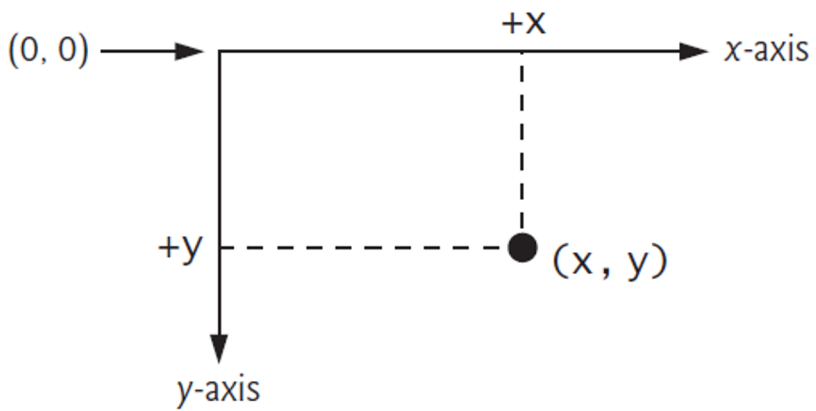
Message dialog type	Icon	Description
`ERROR_MESSAGE`		Indicates an error.
`INFORMATION_MESSAGE`		Indicates an informational message.
`WARNING_MESSAGE`		Warns of a potential problem.
`QUESTION_MESSAGE`		Poses a question. This dialog normally requires a response, such as clicking a **Yes** or a **No** button.
`PLAIN_MESSAGE`	no icon	A dialog that contains a message, but no icon.

Classes and interfaces Java's original graphics capabilities and Java 2D API.



Java Coordinate System for Drawing

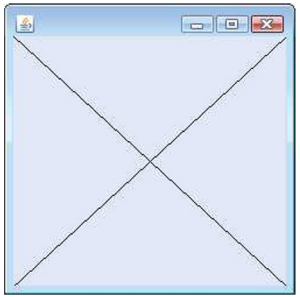
- You need to understand Java's coordinate system for identifying points on the screen.
- The upper-left corner of a GUI component has the coordinates (0, 0).
- A coordinate pair is composed of an x-coordinate (the horizontal coordinate) and a y-coordinate (the vertical coordinate).
- The x-coordinate is the horizontal location moving from left to right.
- The y-coordinate is the vertical location moving from top to bottom.
- The x-axis describes every horizontal coordinate, and the y-axis every vertical coordinate.
- Coordinates indicate where graphics should be displayed on a screen.
- Coordinate units are measured in pixels.

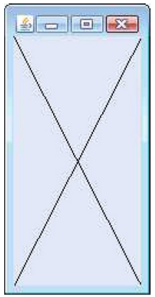


First Drawing Application

```java
// Fig. 4.18: DrawPanel.java
// Using drawLine to connect the corners of a panel.
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawPanel extends JPanel
{
   // draws an X from the corners of the panel
   public void paintComponent(Graphics g)
   {
      // call paintComponent to ensure the panel displays correctly
      super.paintComponent(g);

      int width = getWidth(); // total width
      int height = getHeight(); // total height

      // draw a line from the upper-left to the lower-right
      g.drawLine(0, 0, width, height);

      // draw a line from the lower-left to the upper-right
      g.drawLine(0, height, width, 0);
   }
} // end class DrawPanel
```

```java
// Fig. 4.19: DrawPanelTest.java
// Creating JFrame to display DrawPanel.
import javax.swing.JFrame;

public class DrawPanelTest
{
   public static void main(String[] args)
   {
      // create a panel that contains our drawing
      DrawPanel panel = new DrawPanel();

      // create a new frame to hold the panel
      JFrame application = new JFrame();

      // set the frame to exit when it is closed
      application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      application.add(panel); // add the panel to the frame
      application.setSize(250, 250); // set the size of the frame
      application.setVisible(true); // make the frame visible
   }
} // end class DrawPanelTest
```





Method paintComponent

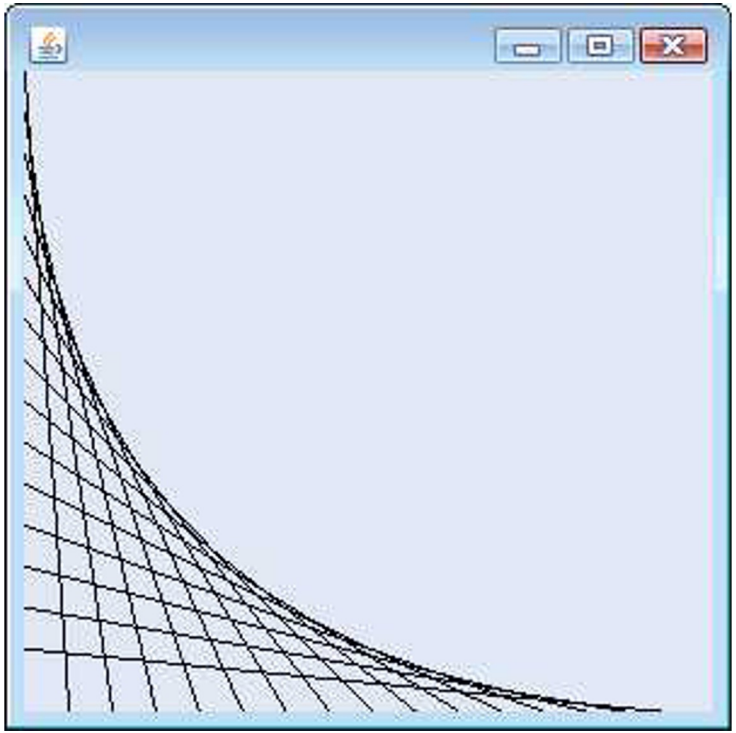
- Every JPanel, including our DrawPanel, has a paintComponent method which the system automatically calls every time it needs to display the DrawPanel.
- Method paintComponent must be declared as shown in line 9—otherwise, the system will not call it.
- This method is called when a JPanel is first displayed on the screen, when it's covered then uncovered by a window on the screen, and when the window in which it appears is resized.
- Method paintComponent requires one argument, a Graphics object, that's provided by the system when it calls paintComponent.
- The first statement in every paintComponent method you create should always be which ensures that the panel is properly rendered before we begin drawing on it:

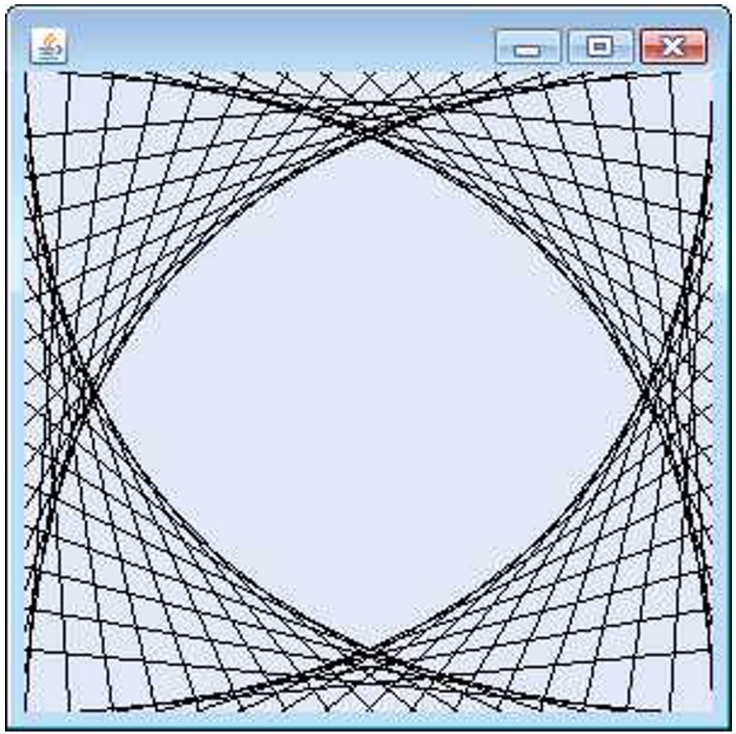
```
super.paintComponent(g);
```

Class JFrame

- You create a window with an object of class JFrame.
- JFrame method setDefaultCloseOperation with the argument JFrame.EXIT_ON_CLOSE to indicate that the application should terminate when the user closes the window.
- Class JFrame's add method to attach the DrawPanel to the JFrame.
- Method setSize takes two parameters that represent the width and height of the JFrame, respectively.
- Finally, displays the JFrame by calling its setVisible method with the argument true.
- When the JFrame is displayed, the DrawPanel's paintComponent method is implicitly called, and the two lines are drawn.
- Try resizing the window to see that the lines always draw based on the window's current width and height.

Exercise with Draw Lines (Lab)

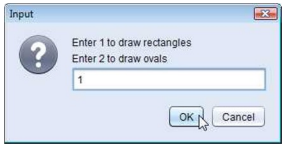


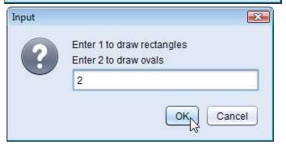


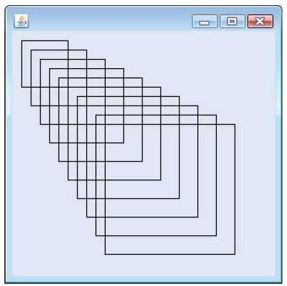
Drawing Rectangles and Ovals

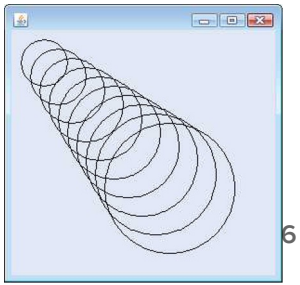
```java
// Fig. 5.27: Shapes.java
// Drawing a cascade of shapes based on the user's choice.
import java.awt.Graphics;
import javax.swing.JPanel;

public class Shapes extends JPanel
{
   private int choice; // user's choice of which shape to draw

   // constructor sets the user's choice
   public Shapes(int userChoice)
   {
      choice = userChoice;
   }

   // draws a cascade of shapes starting from the top-left corner
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      for (int i = 0; i < 10; i++)
      {
         // pick the shape based on the user's choice
         switch (choice)
         {
            case 1: // draw rectangles
               g.drawRect(10 + i * 10, 10 + i * 10,
                  50 + i * 10, 50 + i * 10);
               break;
            case 2: // draw ovals
               g.drawOval(10 + i * 10, 10 + i * 10,
                  50 + i * 10, 50 + i * 10);
               break;
         }
      }
   }
} // end class Shapes
```

```java
// Fig. 5.28: ShapesTest.java
// Obtaining user input and creating a JFrame to display Shapes.
import javax.swing.JFrame; //handle the display
import javax.swing.JOptionPane;

public class ShapesTest
{
   public static void main(String[] args)
   {
      // obtain user's choice
      String input = JOptionPane.showInputDialog(
         "Enter 1 to draw rectangles\n" +
         "Enter 2 to draw ovals");

      int choice = Integer.parseInt(input); // convert input to int

      // create the panel with the user's input
      Shapes panel = new Shapes(choice);

      JFrame application = new JFrame(); // creates a new JFrame

      application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      application.add(panel);
      application.setSize(300, 300);
      application.setVisible(true);
   }
} // end class ShapesTest
```



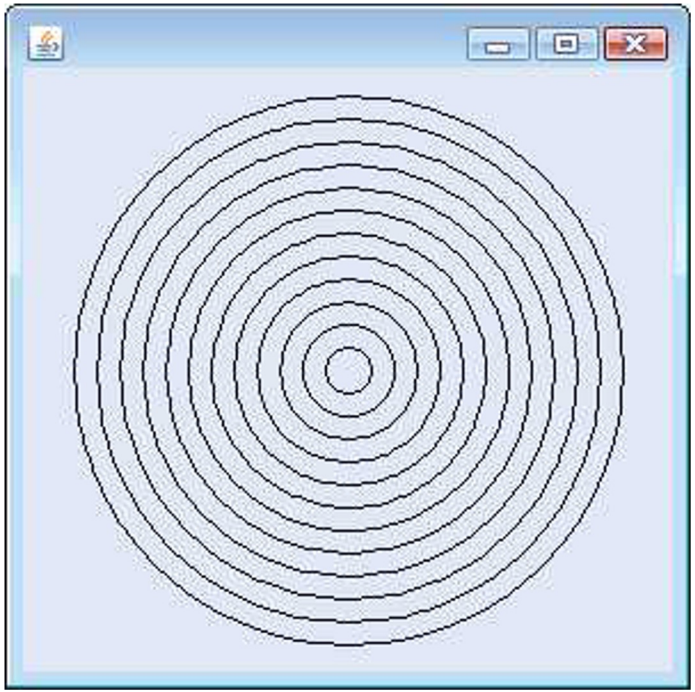




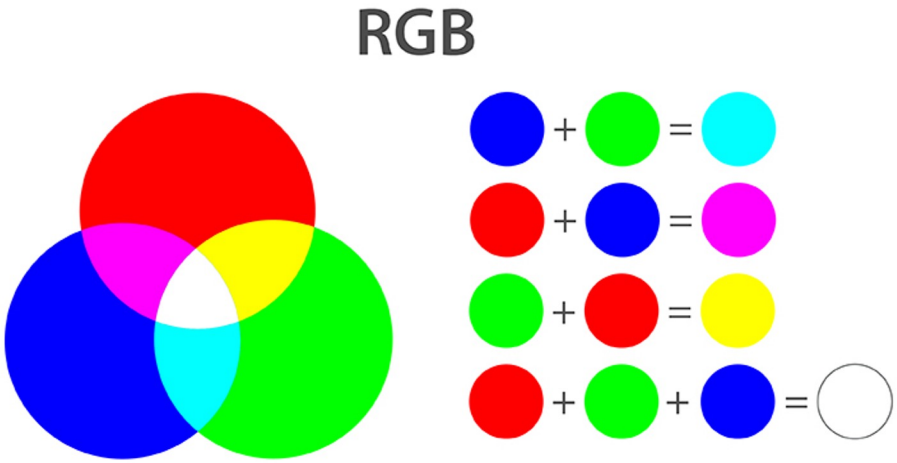


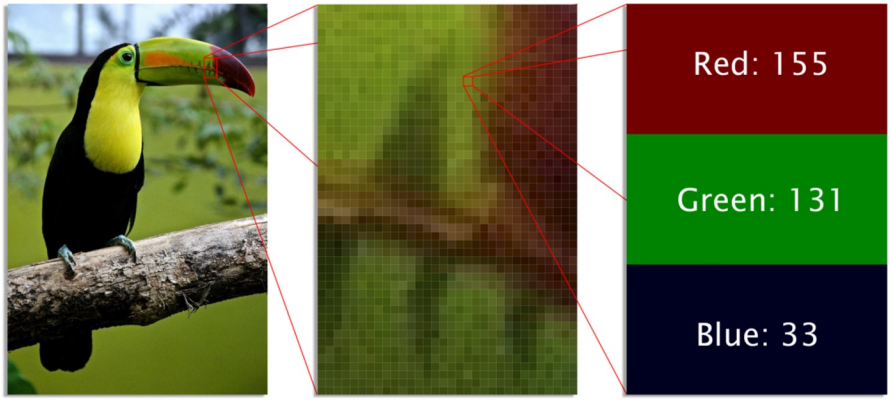
Exercise to draw concentric circles (Lab)

The innermost circle should have a radius of 10 pixels, and each successive circle should have a radius 10 pixels larger than the previous one.



Color in Computer





Color Control

Color constant	RGB value
`public static final Color RED`	255, 0, 0
`public static final Color GREEN`	0, 255, 0
`public static final Color BLUE`	0, 0, 255
`public static final Color ORANGE`	255, 200, 0
`public static final Color PINK`	255, 175, 175
`public static final Color CYAN`	0, 255, 255
`public static final Color MAGENTA`	255, 0, 255
`public static final Color YELLOW`	255, 255, 0
`public static final Color BLACK`	0, 0, 0
`public static final Color WHITE`	255, 255, 255
`public static final Color GRAY`	128, 128, 128
`public static final Color LIGHT_GRAY`	192, 192, 192
`public static final Color DARK_GRAY`	64, 64, 64

`Color` constants and their RGB values.

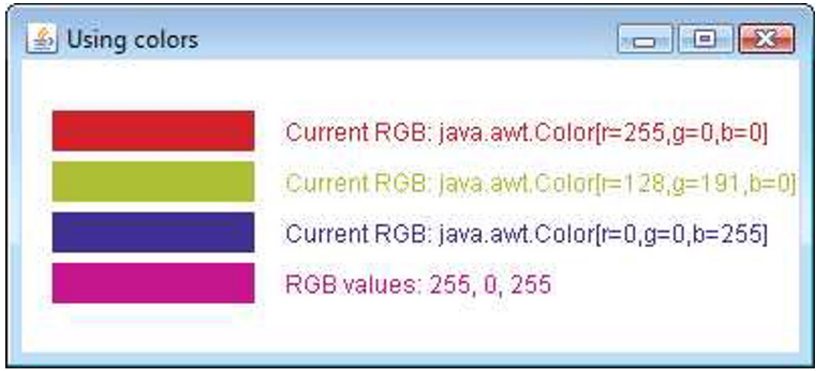
Method	Description
`Color` constructors and methods	
`public Color(int r, int g, int b)`	Creates a color based on red, green and blue components expressed as integers from 0 to 255.
`public Color(float r, float g, float b)`	Creates a color based on red, green and blue components expressed as floating-point values from 0.0 to 1.0.
`public int getRed()`	Returns a value between 0 and 255 representing the red content.
`public int getGreen()`	Returns a value between 0 and 255 representing the green content.
`public int getBlue()`	Returns a value between 0 and 255 representing the blue content.
`Graphics` methods for manipulating `Color`s	
`public Color getColor()`	Returns `Color` object representing current color for the graphics context.
`public void setColor(Color c)`	Sets the current color for drawing with the graphics context.

`Color` methods and color-related `Graphics` methods.

Draw Colors

```java
// Fig. 13.5: ColorJPanel.java
// Changing drawing colors.
import java.awt.Graphics;
import java.awt.Color;
import javax.swing.JPanel;

public class ColorJPanel extends JPanel
{
   // draw rectangles and Strings in different colors
   @Override
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      this.setBackground(Color.WHITE);

      // set new drawing color using integers
      g.setColor(new Color(255, 0, 0));
      g.fillRect(15, 25, 100, 20);
      g.drawString("Current RGB: " + g.getColor(), 130, 40);

      // set new drawing color using floats
      g.setColor(new Color(0.50f, 0.75f, 0.0f));
      g.fillRect(15, 50, 100, 20);
      g.drawString("Current RGB: " + g.getColor(), 130, 65);

      // set new drawing color using static Color objects
      g.setColor(Color.BLUE);
      g.fillRect(15, 75, 100, 20);
      g.drawString("Current RGB: " + g.getColor(), 130, 90);

      // display individual RGB values
      Color color = Color.MAGENTA;
      g.setColor(color);
      g.fillRect(15, 100, 100, 20);
      g.drawString("RGB values: " + color.getRed() + ", " +
         color.getGreen() + ", " + color.getBlue(), 130, 115);
   }
} // end class ColorJPanel
```

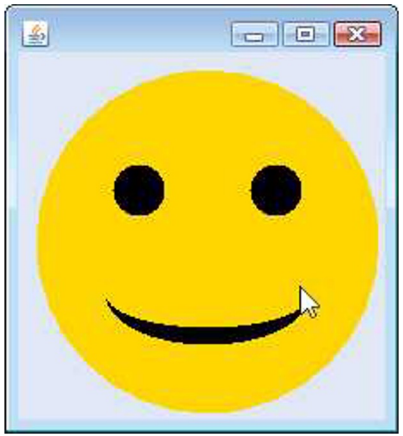
```java
// Fig. 13.6: ShowColors.java
// Demonstrating Colors.
import javax.swing.JFrame;

public class ShowColors
{
   // execute application
   public static void main(String[] args)
   {
      // create frame for ColorJPanel
      JFrame frame = new JFrame("Using colors");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      ColorJPanel colorJPanel = new ColorJPanel();
      frame.add(colorJPanel);
      frame.setSize(400, 180);
      frame.setVisible(true);
   }
} // end class ShowColors
```



Colors and Filled Shapes

```java
// Fig. 6.11: DrawSmiley.java
// Drawing a smiley face using colors and filled shapes.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawSmiley extends JPanel
{
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      // draw the face
      g.setColor(Color.YELLOW);
      g.fillOval(10, 10, 200, 200);

      // draw the eyes
      g.setColor(Color.BLACK);
      g.fillOval(55, 65, 30, 30);
      g.fillOval(135, 65, 30, 30);

      // draw the mouth
      g.fillOval(50, 110, 120, 60);

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

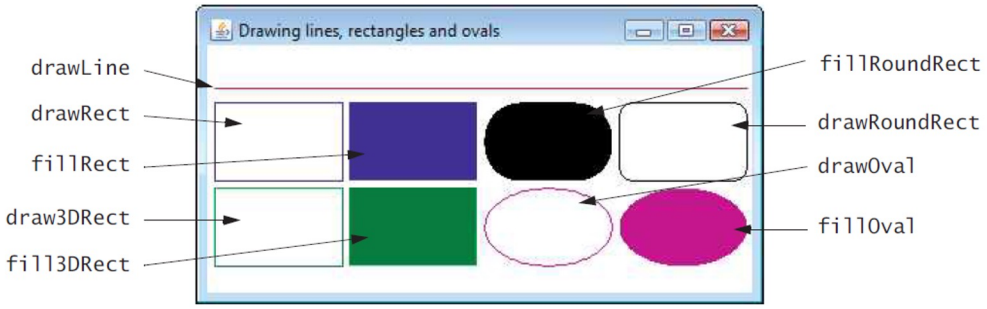
```java
// Fig. 6.12: DrawSmileyTest.java
// Test application that displays a smiley face.
import javax.swing.JFrame;

public class DrawSmileyTest
{
   public static void main(String[] args)
   {
      DrawSmiley panel = new DrawSmiley();
      JFrame application = new JFrame();

      application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      application.add(panel);
      application.setSize(230, 250);
      application.setVisible(true);
   }
} // end class DrawSmileyTest
```



Drawing Lines, Rectangles and Ovals

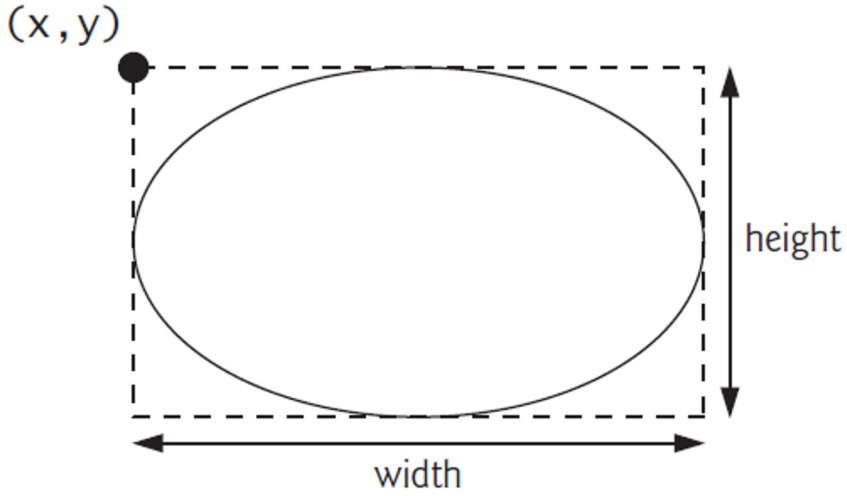
```java
// Fig. 13.18: LinesRectsOvalsJPanel.java
// Drawing lines, rectangles and ovals.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class LinesRectsOvalsJPanel extends JPanel
{
   // display various lines, rectangles and ovals
   @Override
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      this.setBackground(Color.WHITE);

      g.setColor(Color.RED);
      g.drawLine(5, 30, 380, 30);

      g.setColor(Color.BLUE);
      g.drawRect(5, 40, 90, 55);
      g.fillRect(100, 40, 90, 55);

      g.setColor(Color.CYAN);
      g.fillRoundRect(195, 40, 90, 55, 50, 50);
      g.drawRoundRect(290, 40, 90, 55, 20, 20);

      g.setColor(Color.GREEN);
      g.draw3DRect(5, 100, 90, 55, true);
      g.fill3DRect(100, 100, 90, 55, false);

      g.setColor(Color.MAGENTA);
      g.drawOval(195, 100, 90, 55);
      g.fillOval(290, 100, 90, 55);
   }
} // end class LinesRectsOvalsJPanel
```

```java
// Fig. 13.19: LinesRectsOvals.java
// Testing LinesRectsOvalsJPanel.
import java.awt.Color;
import javax.swing.JFrame;

public class LinesRectsOvals
{
   // execute application
   public static void main(String[] args)
   {
      // create frame for LinesRectsOvalsJPanel
      JFrame frame =
         new JFrame("Drawing lines, rectangles and ovals");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      LinesRectsOvalsJPanel linesRectsOvalsJPanel =
         new LinesRectsOvalsJPanel();
      linesRectsOvalsJPanel.setBackground(Color.WHITE);
      frame.add(linesRectsOvalsJPanel);
      frame.setSize(400, 210);
      frame.setVisible(true);
   }
} // end class LinesRectsOvals
```

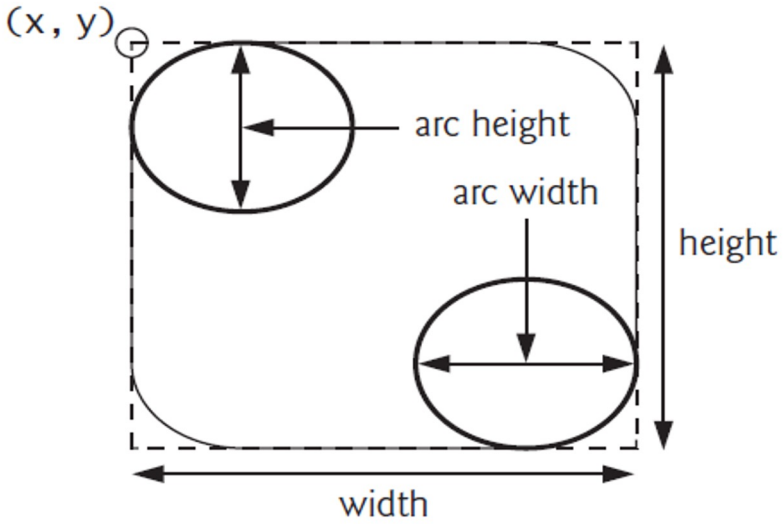


Testing LinesRectsOvalsJPanel.

Oval and Round Rectangles Explanation

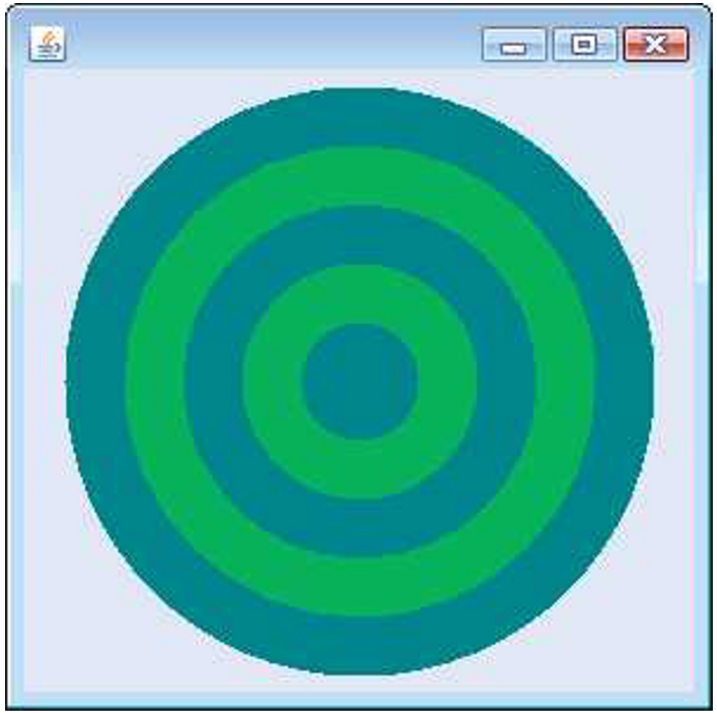


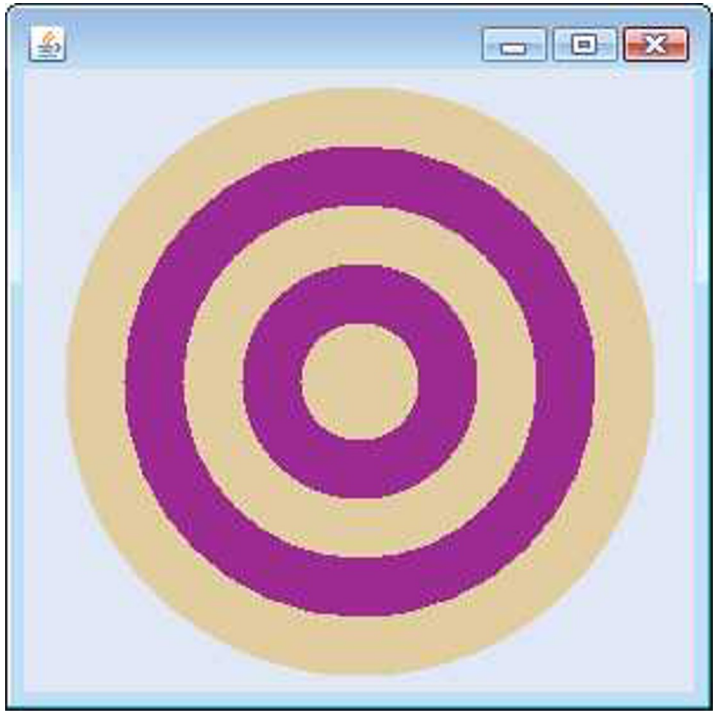
Oval bounded by a rectangle.



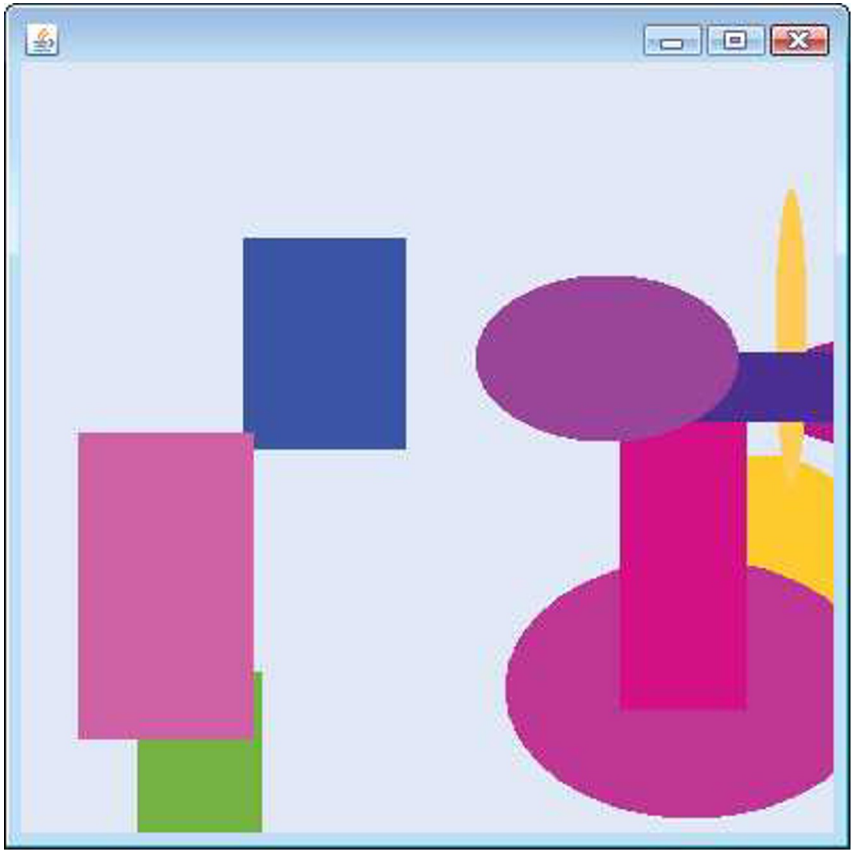
rounded rectangles.

A bulls-eye with two random colors (Lab)





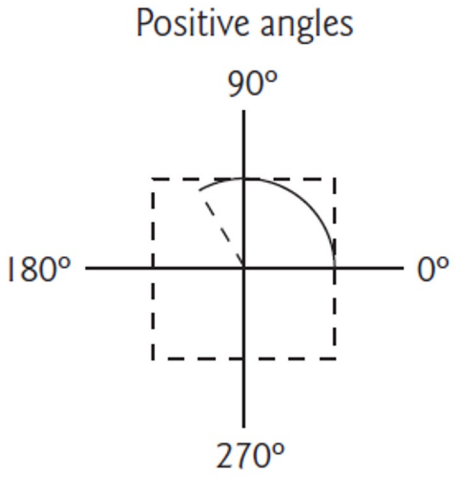
Draw an abstract painting (Lab)

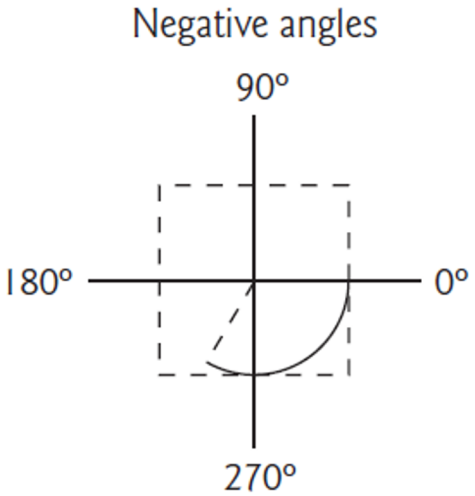


Drawing Arc

Method	Description
`public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`	Draws an arc relative to the bounding rectangle's top-left x- and y-coordinates with the specified `width` and `height`. The arc segment is drawn starting at `startAngle` and sweeps `arcAngle` degrees.
`public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`	Draws a filled arc (i.e., a sector) relative to the bounding rectangle's top-left x- and y-coordinates with the specified `width` and `height`. The arc segment is drawn starting at `startAngle` and sweeps `arcAngle` degrees.

`Graphics` methods for drawing arcs.

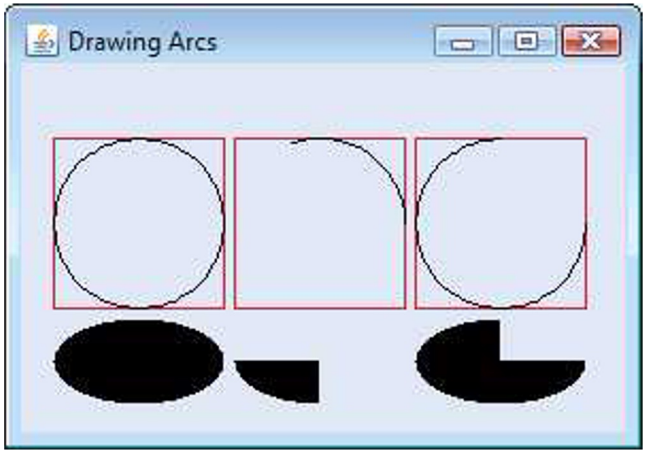




Arcs displayed with drawArc and fillArc

```java
// Fig. 13.24: ArcsJPanel.java
// Arcs displayed with drawArc and fillArc.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class ArcsJPanel extends JPanel
{
   // draw rectangles and arcs
   @Override
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      // start at 0 and sweep 360 degrees
      g.setColor(Color.RED);
      g.drawRect(15, 35, 80, 80);
      g.setColor(Color.BLACK);
      g.drawArc(15, 35, 80, 80, 0, 360);

      // start at 0 and sweep 110 degrees
      g.setColor(Color.RED);
      g.drawRect(100, 35, 80, 80);
      g.setColor(Color.BLACK);
      g.drawArc(100, 35, 80, 80, 0, 110);

      // start at 0 and sweep -270 degrees
      g.setColor(Color.RED);
      g.drawRect(185, 35, 80, 80);
      g.setColor(Color.BLACK);
      g.drawArc(185, 35, 80, 80, 0, -270);

      // start at 0 and sweep 360 degrees
      g.fillArc(15, 120, 80, 40, 0, 360);

      // start at 270 and sweep -90 degrees
      g.fillArc(100, 120, 80, 40, 270, -90);

      // start at 0 and sweep -270 degrees
      g.fillArc(185, 120, 80, 40, 0, -270);
   }
} // end class ArcsJPanel
```

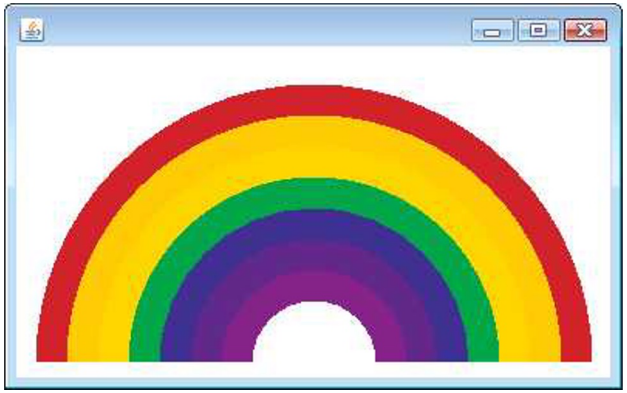
```java
// Fig. 13.25: DrawArcs.java
// Drawing arcs.
import javax.swing.JFrame;

public class DrawArcs
{
   // execute application
   public static void main(String[] args)
   {
      // create frame for ArcsJPanel
      JFrame frame = new JFrame("Drawing Arcs");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      ArcsJPanel arcsJPanel = new ArcsJPanel();
      frame.add(arcsJPanel);
      frame.setSize(300, 210);
      frame.setVisible(true);
   }
} // end class DrawArcs
```



Drawing Arcs

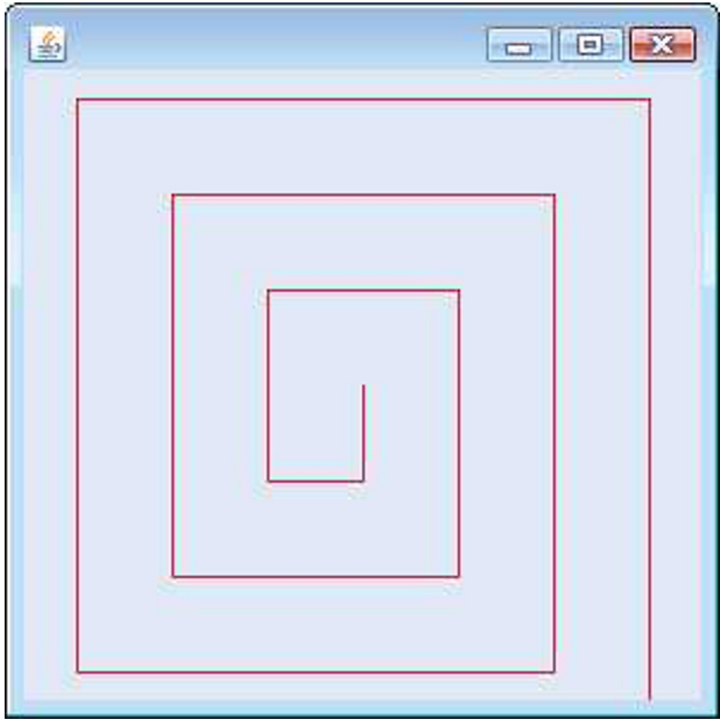
```java
// Fig. 7.25: DrawRainbow.java
// Drawing a rainbow using arcs and an array of colors.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawRainbow extends JPanel
{
   // define indigo and violet
   private final static Color VIOLET = new Color(128, 0, 128);
   private final static Color INDIGO = new Color(75, 0, 130);

   // colors to use in the rainbow, starting from the innermost
   // The two white entries result in an empty arc in the center
   private Color[] colors =
      { Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
        Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED };

   // constructor
   public DrawRainbow()
   {
      setBackground(Color.WHITE); // set the background to white
   }

   // draws a rainbow using concentric arcs
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      int radius = 20; // radius of an arc

      // draw the rainbow near the bottom-center
      int centerX = getWidth() / 2;
      int centerY = getHeight() - 10;

      // draws filled arcs starting with the outermost
      for (int counter = colors.length; counter > 0; counter--)
      {
         // set the color for the current arc
         g.setColor(colors[counter - 1]);

         // fill the arc from 0 to 180 degrees
         g.fillArc(centerX - counter * radius,
            centerY - counter * radius,
            counter * radius * 2, counter * radius * 2, 0, 180);
      }
   }
} // end class DrawRainbow
```

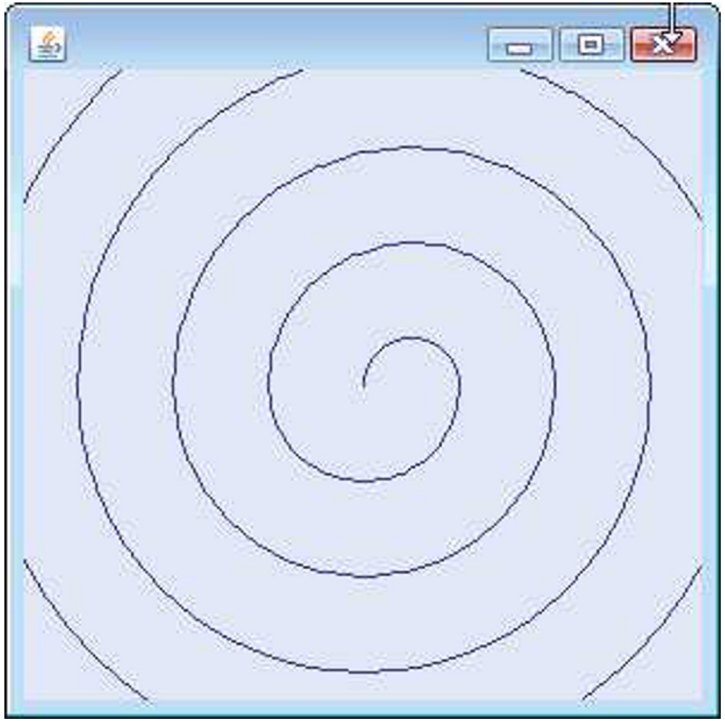
```java
// Fig. 7.26: DrawRainbowTest.java
// Test application to display a rainbow.
import javax.swing.JFrame;

public class DrawRainbowTest
{
   public static void main(String[] args)
   {
      DrawRainbow panel = new DrawRainbow();
      JFrame application = new JFrame();

      application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      application.add(panel);
      application.setSize(400, 250);
      application.setVisible(true);
   }
} // end class DrawRainbowTest
```



Drawing Spiral Paintings (Lab)

Drawing a spiral using drawLine (left) and drawArc (right).





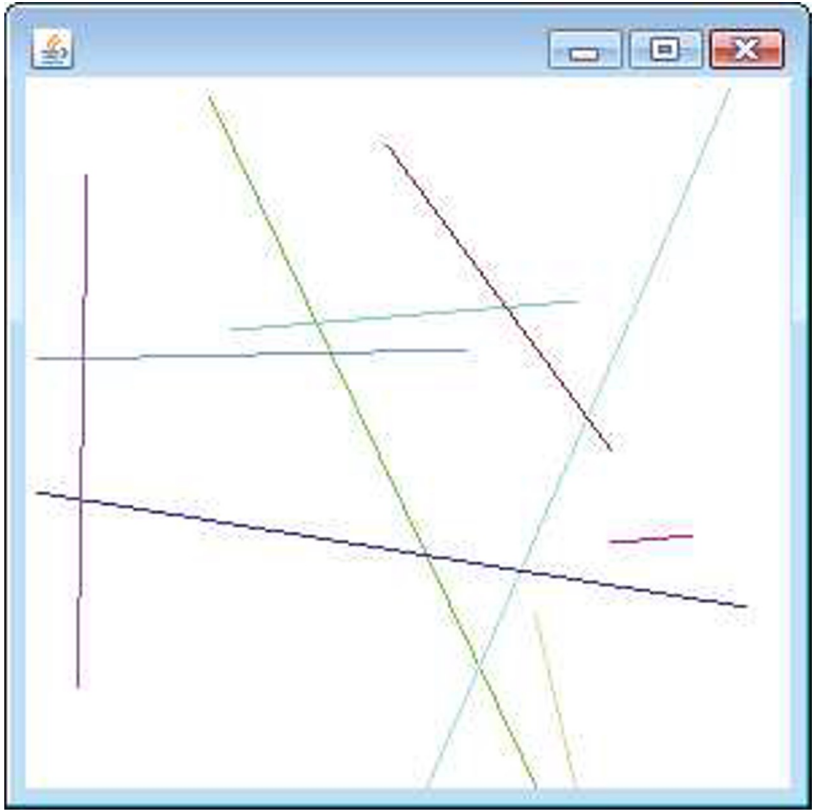
Objects with Graphics

```java
// Fig. 8.17: MyLine.java
// MyLine class represents a line.
import java.awt.Color;
import java.awt.Graphics;

public class MyLine
{
   private int x1; // x-coordinate of first endpoint
   private int y1; // y-coordinate of first endpoint
   private int x2; // x-coordinate of second endpoint
   private int y2; // y-coordinate of second endpoint
   private Color color; // color of this line

   // constructor with input values
   public MyLine(int x1, int y1, int x2, int y2, Color color)
   {
      this.x1 = x1;
      this.y1 = y1;
      this.x2 = x2;
      this.y2 = y2;
      this.color = color;
   }

   // Draw the line in the specified color
   public void draw(Graphics g)
   {
      g.setColor(color);
      g.drawLine(x1, y1, x2, y2);
   }
} // end class MyLine
```

```java
// Fig. 8.19: TestDraw.java
// Creating a JFrame to display a DrawPanel.
import javax.swing.JFrame;

public class TestDraw
{
   public static void main(String[] args)
   {
      DrawPanel panel = new DrawPanel();
      JFrame app = new JFrame();

      app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      app.add(panel);
      app.setSize(300, 300);
      app.setVisible(true);
   }
} // end class TestDraw
```

```java
// Fig. 8.18: DrawPanel.java
// Program that uses class MyLine
// to draw random lines.
import java.awt.Color;
import java.awt.Graphics;
import java.security.SecureRandom;
import javax.swing.JPanel;

public class DrawPanel extends JPanel
{
   private SecureRandom randomNumbers = new SecureRandom();
   private MyLine[] lines; // array of lines

   // constructor, creates a panel with random shapes
   public DrawPanel()
   {
      setBackground(Color.WHITE);

      lines = new MyLine[5 + randomNumbers.nextInt(5)];

      // create lines
      for (int count = 0; count < lines.length; count++)
      {
         // generate random coordinates
         int x1 = randomNumbers.nextInt(300);
         int y1 = randomNumbers.nextInt(300);
         int x2 = randomNumbers.nextInt(300);
         int y2 = randomNumbers.nextInt(300);

         // generate a random color
         Color color = new Color(randomNumbers.nextInt(256),
            randomNumbers.nextInt(256), randomNumbers.nextInt(256));

         // add the line to the list of lines to be displayed
         lines[count] = new MyLine(x1, y1, x2, y2, color);
      }
   }

   // for each shape array, draw the individual shapes
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      // draw the lines
      for (MyLine line : lines)
         line.draw(g);
   }
} // end class DrawPanel
```

Objects with Graphics



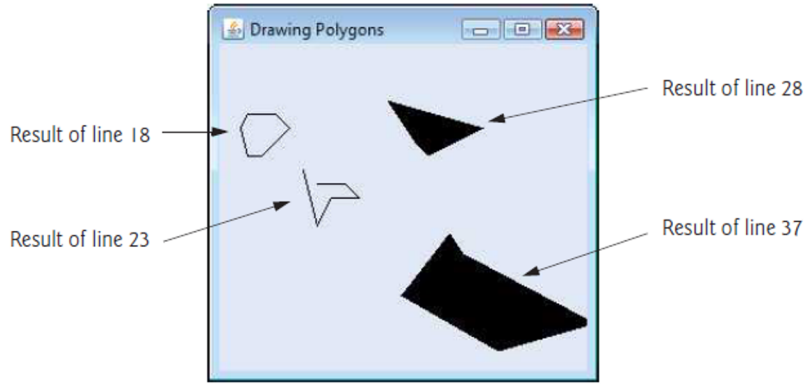
Drawing Polygons and Polylines

Method	Description
`Graphics` methods for drawing polygons	
`public void drawPolygon(int[] xPoints, int[] yPoints, int points)`	Draws a polygon. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point in the `yPoints` array. The last argument specifies the number of `points`. This method draws a *closed polygon*. If the last point is different from the first, the polygon is *closed* by a line that connects the last point to the first.
`public void drawPolyline(int[] xPoints, int[] yPoints, int points)`	Draws a sequence of connected lines. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point in the `yPoints` array. The last argument specifies the number of `points`. If the last point is different from the first, the polyline is *not* closed.
`public void drawPolygon(Polygon p)`	Draws the specified polygon.
`public void fillPolygon(int[] xPoints, int[] yPoints, int points)`	Draws a *filled* polygon. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point in the `yPoints` array. The last argument specifies the number of `points`. This method draws a *closed polygon*. If the last point is different from the first, the polygon is *closed* by a line that connects the last point to the first.
`public void fillPolygon(Polygon p)`	Draws the specified *filled* polygon. The polygon is *closed*.
`Polygon` constructors and methods	
`public Polygon()`	Constructs a new polygon object. The polygon does not contain any points.
`public Polygon(int[] xValues, int[] yValues, int numberOfPoints)`	Constructs a new polygon object. The polygon has `numberOfPoints` sides, with each point consisting of an *x*-coordinate from `xValues` and a *y*-coordinate from `yValues`.
`public void addPoint(int x, int y)`	Adds pairs of *x*- and *y*-coordinates to the `Polygon`.

Drawing Polygons and Polylines

```java
// Fig. 13.27: PolygonsJPanel.java
// Drawing polygons.
import java.awt.Graphics;
import java.awt.Polygon;
import javax.swing.JPanel;

public class PolygonsJPanel extends JPanel
{
   // draw polygons and polylines
   @Override
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      // draw polygon with Polygon object
      int[] xValues = {20, 40, 50, 30, 20, 15};
      int[] yValues = {50, 50, 60, 80, 80, 60};
      Polygon polygon1 = new Polygon(xValues, yValues, 6);
      g.drawPolygon(polygon1);

      // draw polylines with two arrays
      int[] xValues2 = {70, 90, 100, 80, 70, 65, 60};
      int[] yValues2 = {100, 100, 110, 110, 130, 110, 90};
      g.drawPolyline(xValues2, yValues2, 7);

      // fill polygon with two arrays
      int[] xValues3 = {120, 140, 150, 190};
      int[] yValues3 = {40, 70, 80, 60};
      g.fillPolygon(xValues3, yValues3, 4);

      // draw filled polygon with Polygon object
      Polygon polygon2 = new Polygon();
      polygon2.addPoint(165, 135);
      polygon2.addPoint(175, 150);
      polygon2.addPoint(270, 200);
      polygon2.addPoint(200, 220);
      polygon2.addPoint(130, 180);
      g.fillPolygon(polygon2);
   }
} // end class PolygonsJPanel
```

```java
// Fig. 13.28: DrawPolygons.java
// Drawing polygons.
import javax.swing.JFrame;

public class DrawPolygons
{
   // execute application
   public static void main(String[] args)
   {
      // create frame for PolygonsJPanel
      JFrame frame = new JFrame("Drawing Polygons");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      PolygonsJPanel polygonsJPanel = new PolygonsJPanel();
      frame.add(polygonsJPanel);
      frame.setSize(280, 270);
      frame.setVisible(true);
   }
} // end class DrawPolygons
```



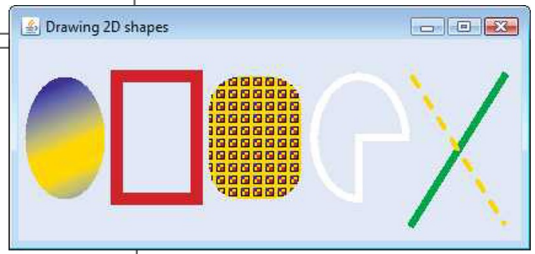
Java 2D API

- The Java 2D API provides advanced two-dimensional graphics capabilities for programmers who require detailed and complex graphical manipulations
- The API includes features for processing line art, text and images in packages java.awt, java.awt.image, java.awt.color, java.awt.font, java.awt.geom, java.awt.print and java.awt.image.renderable.
- Graphics2D is an abstract subclass of class Graphics, so it has all the graphics capabilities demonstrated earlier.
- To access Graphics2D capabilities, we must cast the Graphics reference (g) passed to paintComponent into a Graphics2D reference with a statement such as:

```
Graphics2D g2d = (Graphics2D) g;
```

```java
// Fig. 13.29: ShapesJPanel.java
// Demonstrating some Java 2D shapes.
import java.awt.Color;
import java.awt.Graphics;
import java.awt.BasicStroke;
import java.awt.GradientPaint;
import java.awt.TexturePaint;
import java.awt.Rectangle;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.awt.geom.RoundRectangle2D;
import java.awt.geom.Arc2D;
import java.awt.geom.Line2D;
import java.awt.image.BufferedImage;
import javax.swing.JPanel;

public class ShapesJPanel extends JPanel
{
   // draw shapes with Java 2D API
   @Override
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      Graphics2D g2d = (Graphics2D) g; // cast g to Graphics2D

      // draw 2D ellipse filled with a blue-yellow gradient
      g2d.setPaint(new GradientPaint(5, 30, Color.BLUE, 35, 100,
         Color.YELLOW, true));
      g2d.fill(new Ellipse2D.Double(5, 30, 65, 100));

      // draw 2D rectangle in red
      g2d.setPaint(Color.RED);
      g2d.setStroke(new BasicStroke(10.0f));
      g2d.draw(new Rectangle2D.Double(80, 30, 65, 100));

      // draw 2D rounded rectangle with a buffered background
      BufferedImage buffImage = new BufferedImage(10, 10,
         BufferedImage.TYPE_INT_RGB);

      // obtain Graphics2D from buffImage and draw on it
      Graphics2D gg = buffImage.createGraphics();
      gg.setColor(Color.YELLOW);
      gg.fillRect(0, 0, 10, 10);
      gg.setColor(Color.BLACK);
      gg.drawRect(1, 1, 6, 6);
      gg.setColor(Color.BLUE);
      gg.fillRect(1, 1, 3, 3);
      gg.setColor(Color.RED);
      gg.fillRect(4, 4, 3, 3); // draw a filled rectangle

      // paint buffImage onto the JFrame
      g2d.setPaint(new TexturePaint(buffImage,
         new Rectangle(10, 10)));
      g2d.fill(
         new RoundRectangle2D.Double(155, 30, 75, 100, 50, 50));

      // draw 2D pie-shaped arc in white
      g2d.setPaint(Color.WHITE);
      g2d.setStroke(new BasicStroke(6.0f));
      g2d.draw(
         new Arc2D.Double(240, 30, 75, 100, 0, 270, Arc2D.PIE));

      // draw 2D lines in green and yellow
      g2d.setPaint(Color.GREEN);
      g2d.draw(new Line2D.Double(395, 30, 320, 150));

      // draw 2D line using stroke
      float[] dashes = {10}; // specify dash pattern
      g2d.setPaint(Color.YELLOW);
      g2d.setStroke(new BasicStroke(4, BasicStroke.CAP_ROUND,
         BasicStroke.JOIN_ROUND, 10, dashes, 0));
      g2d.draw(new Line2D.Double(320, 30, 395, 150));
   }
} // end class ShapesJPanel
```

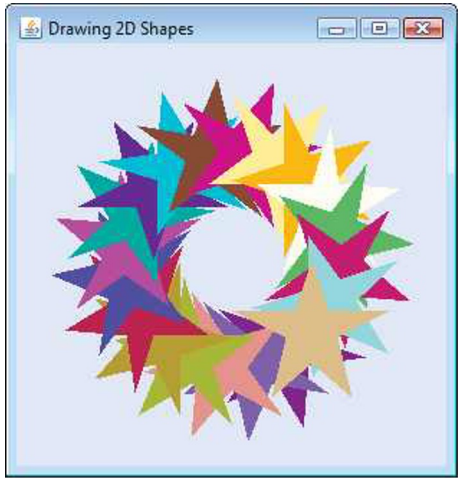
```java
// Fig. 13.30: Shapes.java
// Testing ShapesJPanel.
import javax.swing.JFrame;

public class Shapes
{
   // execute application
   public static void main(String[] args)
   {
      // create frame for ShapesJPanel
      JFrame frame = new JFrame("Drawing 2D shapes");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // create ShapesJPanel
      ShapesJPanel shapesJPanel = new ShapesJPanel();

      frame.add(shapesJPanel);
      frame.setSize(425, 200);
      frame.setVisible(true);
   }
} // end class Shapes
```



```java
// Fig. 13.31: Shapes2JPanel.java
// Demonstrating a general path.
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.GeneralPath;
import java.security.SecureRandom;
import javax.swing.JPanel;

public class Shapes2JPanel extends JPanel
{
   // draw general paths
   @Override
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      SecureRandom random = new SecureRandom();

      int[] xPoints = {55, 67, 109, 73, 83, 55, 27, 37, 1, 43};
      int[] yPoints = {0, 36, 36, 54, 96, 72, 96, 54, 36, 36};

      Graphics2D g2d = (Graphics2D) g;
      GeneralPath star = new GeneralPath();

      // set the initial coordinate of the General Path
      star.moveTo(xPoints[0], yPoints[0]);

      // create the star--this does not draw the star
      for (int count = 1; count < xPoints.length; count++)
         star.lineTo(xPoints[count], yPoints[count]);

      star.closePath(); // close the shape

      g2d.translate(150, 150); // translate the origin to (150, 150)

      // rotate around origin and draw stars in random colors
      for (int count = 1; count <= 20; count++)
      {
         g2d.rotate(Math.PI / 10.0); // rotate coordinate system

         // set random drawing color
         g2d.setColor(new Color(random.nextInt(256),
            random.nextInt(256), random.nextInt(256)));

         g2d.fill(star); // draw filled star
      }
   }
} // end class Shapes2JPanel
```

Creating Your Own Shapes with General Path

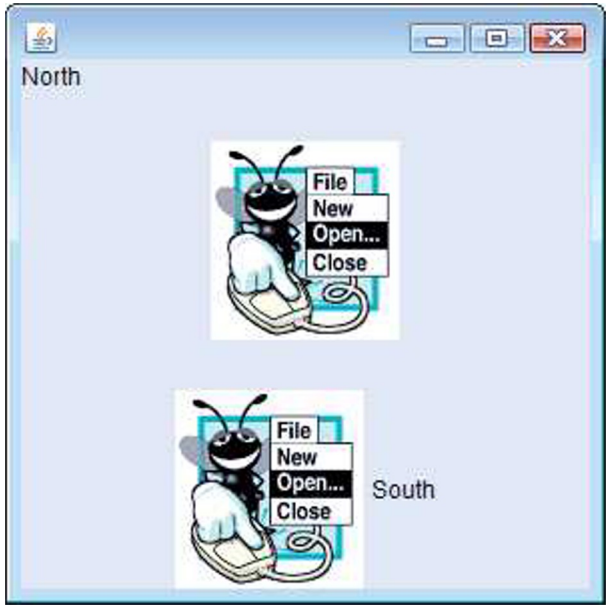
```java
// Fig. 13.32: Shapes2.java
// Demonstrating a general path.
import java.awt.Color;
import javax.swing.JFrame;

public class Shapes2
{
   // execute application
   public static void main(String[] args)
   {
      // create frame for Shapes2JPanel
      JFrame frame = new JFrame("Drawing 2D Shapes");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      Shapes2JPanel shapes2JPanel = new Shapes2JPanel();
      frame.add(shapes2JPanel);
      frame.setBackground(Color.WHITE);
      frame.setSize(315, 330);
      frame.setVisible(true);
   }
} // end class Shapes2
```



Ultimate Challenge - Captain America (Lab)

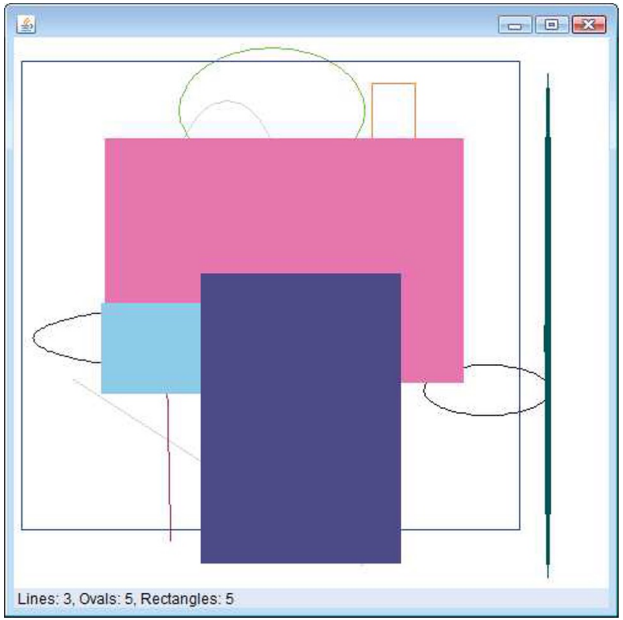
Displaying Text and Images Using Labels

```
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
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);
39       application.setVisible(true);
40    } // end main
41 } // end class LabelDemo
```



JLabel displaying shape statistics (Lab)

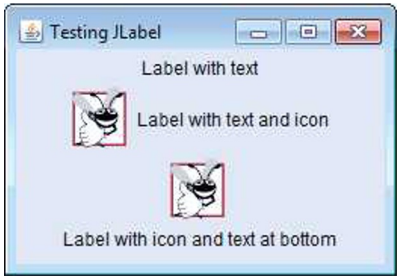
Modify GUI and Graphics to include a JLabel as a status bar that displays counts representing the number of each shape displayed. Class DrawPanel should declare a method that returns a String containing the status text. In main, first create the DrawPanel, then create the JLabel with the status text as an argument to the JLabel's constructor. Attach the JLabel to the SOUTH region of the JFrame

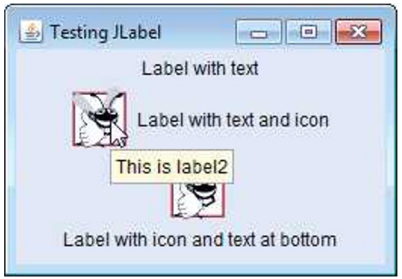


Displaying Text and Images in a Window

```java
// Fig. 12.6: LabelFrame.java
// JLabels with text and icons.
import java.awt.FlowLayout; // specifies how components are arranged
import javax.swing.JFrame; // provides basic window features
import javax.swing.JLabel; // displays text and images
import javax.swing.SwingConstants; // common constants used with Swing
import javax.swing.Icon; // interface used to manipulate images
import javax.swing.ImageIcon; // loads images

public class LabelFrame extends JFrame
{
   private final JLabel label1; // JLabel with just text
   private final JLabel label2; // JLabel constructed with text and icon
   private final JLabel label3; // JLabel with added text and icon

   // LabelFrame constructor adds JLabels to JFrame
   public LabelFrame()
   {
      super("Testing JLabel");
      setLayout(new FlowLayout()); // set frame layout

      // JLabel constructor with a string argument
      label1 = new JLabel("Label with text");
      label1.setToolTipText("This is label1");
      add(label1); // add label1 to JFrame

      // JLabel constructor with string, Icon and alignment arguments
      Icon bug = new ImageIcon(getClass().getResource( "bug1.png"));
      label2 = new JLabel("Label with text and icon", bug,
         SwingConstants.LEFT);
      label2.setToolTipText("This is label2");
      add(label2); // add label2 to JFrame

      label3 = new JLabel(); // JLabel constructor no arguments
      label3.setText("Label with icon and text at bottom");
      label3.setIcon(bug); // add icon to JLabel
      label3.setHorizontalTextPosition(SwingConstants.CENTER);
      label3.setVerticalTextPosition(SwingConstants.BOTTOM);
      label3.setToolTipText("This is label3");
      add(label3); // add label3 to JFrame
   }
} // end class LabelFrame
```

```java
// Fig. 12.7: LabelTest.java
// Testing LabelFrame.
import javax.swing.JFrame;

public class LabelTest
{
   public static void main(String[] args)
   {
      LabelFrame labelFrame = new LabelFrame();
      labelFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      labelFrame.setSize(260, 180);
      labelFrame.setVisible(true);
   }
} // end class LabelTest
```





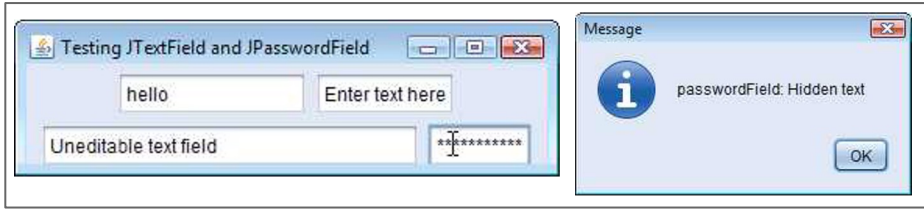
Constant	Description	Constant	Description
Horizontal-position constants		*Vertical-position constants*	
LEFT	Place text on the left	TOP	Place text at the top
CENTER	Place text in the center	CENTER	Place text in the center
RIGHT	Place text on the right	BOTTOM	Place text at the bottom

Positioning constants (static members of interface SwingConstants).

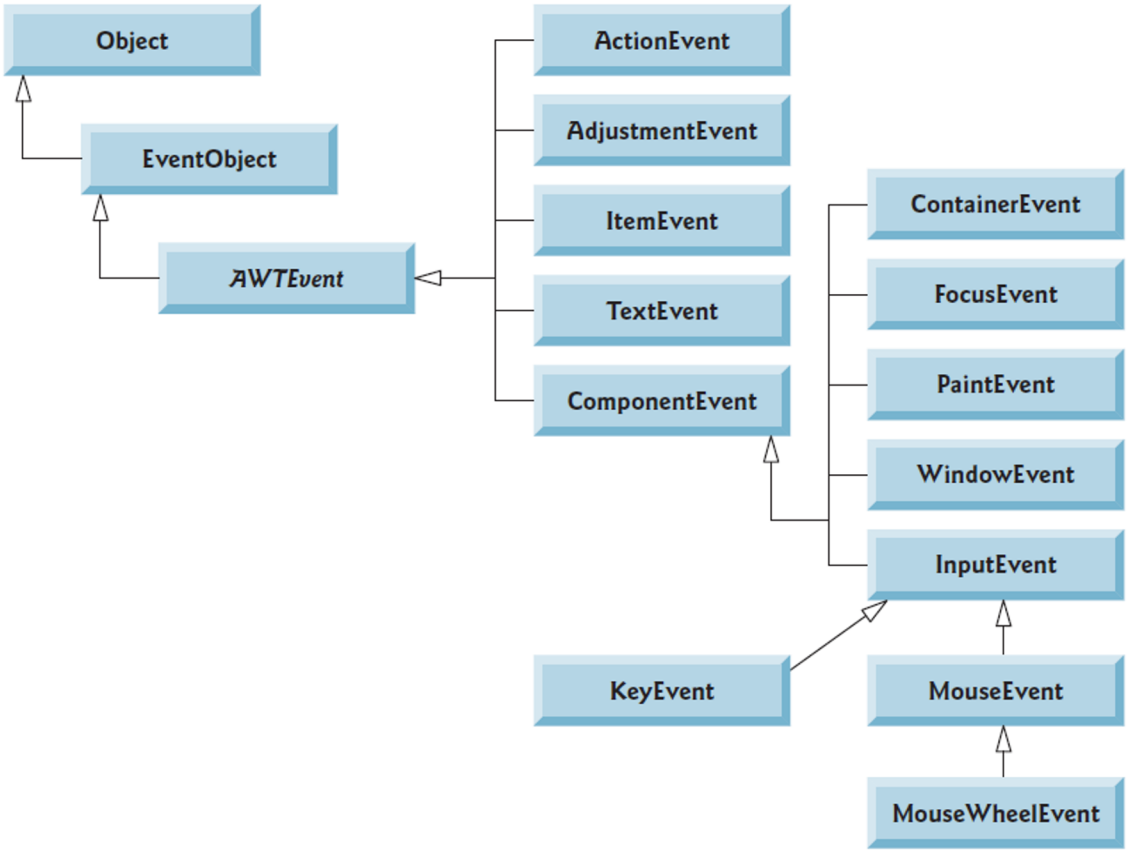
Text Fields and an Introduction to Event Handling with Nested Classes

```java
// Fig. 12.9: TextFieldFrame.java
// JTextFields and JPasswordFields.
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JOptionPane;

public class TextFieldFrame extends JFrame
{
   private final JTextField textField1; // text field with set size
   private final JTextField textField2; // text field with text
   private final JTextField textField3; // text field with text and size
   private final JPasswordField passwordField; // password field with text

   // TextFieldFrame constructor adds JTextFields to JFrame
   public TextFieldFrame()
   {
      super("Testing JTextField and JPasswordField");
      setLayout(new FlowLayout());

      // construct text field with 10 columns
      textField1 = new JTextField(10);
      add(textField1); // add textField1 to JFrame

      // construct text field with default text
      textField2 = new JTextField("Enter text here");
      add(textField2); // add textField2 to JFrame

      // construct text field with default text and 21 columns
      textField3 = new JTextField("Uneditable text field", 21);
      textField3.setEditable(false); // disable editing
      add(textField3); // add textField3 to JFrame

      // construct password field with default text
      passwordField = new JPasswordField("Hidden text");
      add(passwordField); // add passwordField to JFrame

      // register event handlers
      TextFieldHandler handler = new TextFieldHandler();
      textField1.addActionListener(handler);
      textField2.addActionListener(handler);
      textField3.addActionListener(handler);
      passwordField.addActionListener(handler);
   }

   // private inner class for event handling
   private class TextFieldHandler implements ActionListene
   {
      // process text field events
      @Override
      public void actionPerformed(ActionEvent event)
      {
         String string = "";

         // user pressed Enter in JTextField textField1
         if (event.getSource() == textField1)
            string = String.format("textField1: %s",
               event.getActionCommand());

         // user pressed Enter in JTextField textField2
         else if (event.getSource() == textField2)
            string = String.format("textField2: %s",
               event.getActionCommand());

         // user pressed Enter in JTextField textField3
         else if (event.getSource() == textField3)
            string = String.format("textField3: %s",
               event.getActionCommand());

         // user pressed Enter in JTextField passwordField
         else if (event.getSource() == passwordField)
            string = String.format("passwordField: %s",
               event.getActionCommand());

         // display JTextField content
         JOptionPane.showMessageDialog(null, string);
      }
   } // end private inner class TextFieldHandler
} // end class TextFieldFrame
```

```java
// Fig. 12.10: TextFieldTest.java
// Testing TextFieldFrame.
import javax.swing.JFrame;

public class TextFieldTest
{
   public static void main(String[] args)
   {
      TextFieldFrame textFieldFrame = new TextFieldFrame();
      textFieldFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      textFieldFrame.setSize(350, 100);
      textFieldFrame.setVisible(true);
   }
} // end class TextFieldTest
```

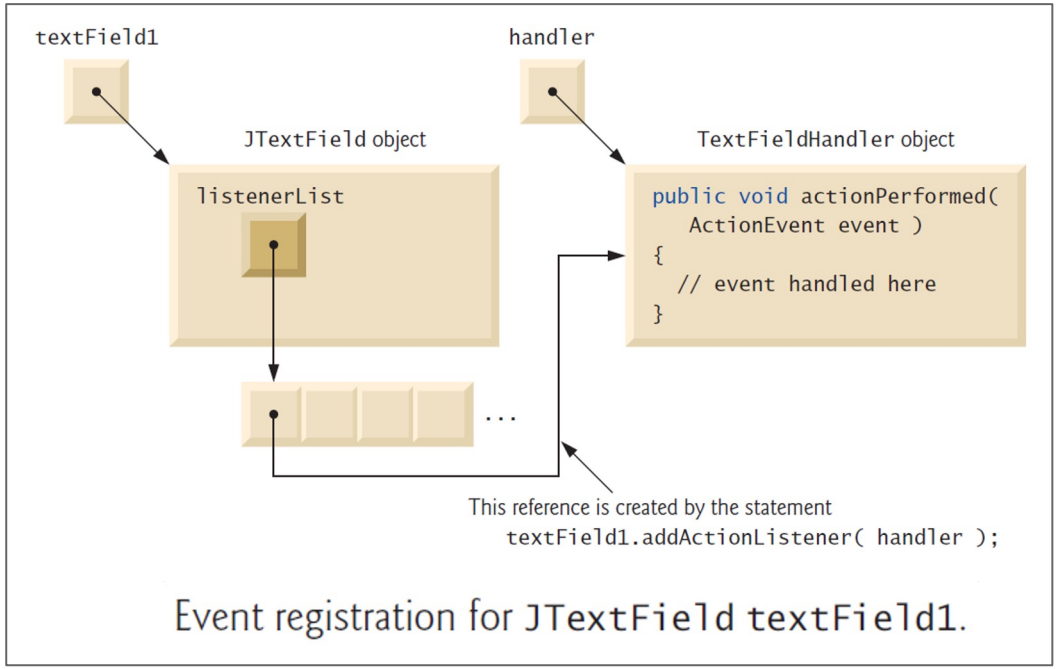


Common GUI Event Types and Listener Interfaces



Some event classes of package `java.awt.event`.

How Event Handling Works

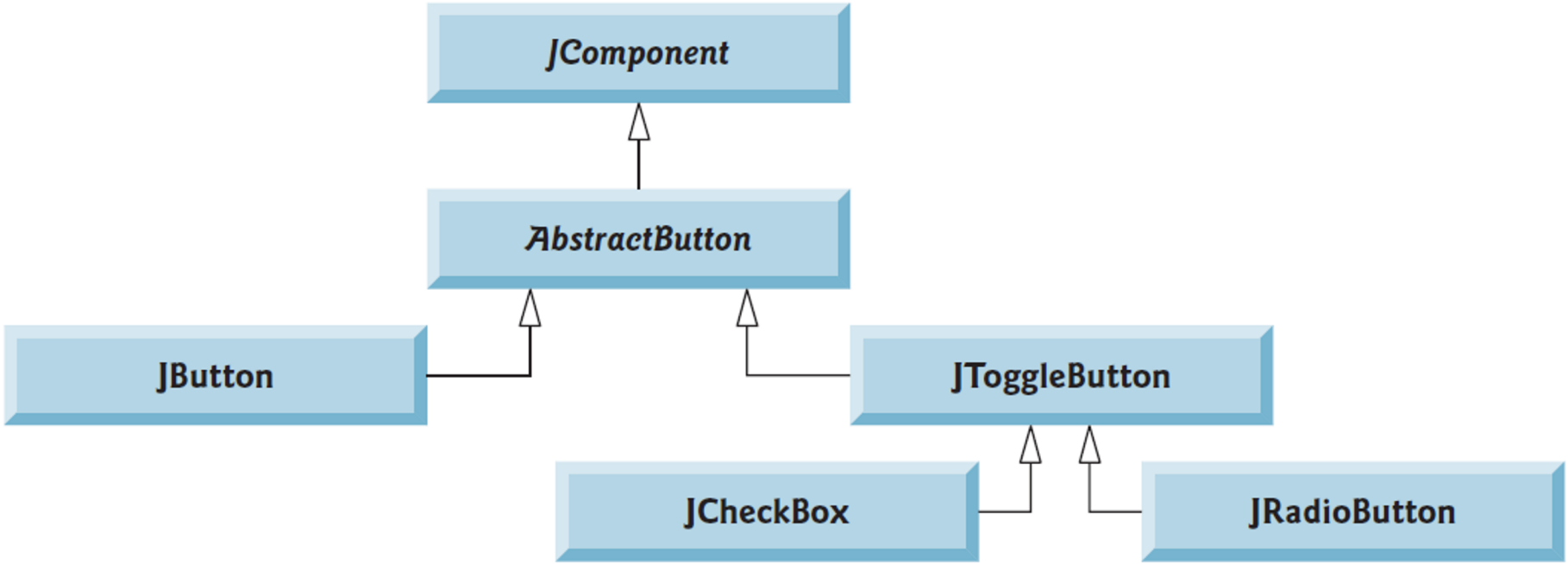


Event registration for JTextField textField1.

A new entry containing a reference to the TextFieldHandler object is placed in textField1's listenerList:

```
textField1.addActionListener(handler);
```

JButton



Swing button hierarchy.

JButton Example

```
1  // Fig. 12.15: ButtonFrame.java
2  // Command buttons and action events.
3  import java.awt.FlowLayout;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JButton;
8  import javax.swing.Icon;
9  import javax.swing.ImageIcon;
10 import javax.swing.JOptionPane;
11
12 public class ButtonFrame extends JFrame
13 {
14    private final JButton plainJButton; // button with just text
15    private final JButton fancyJButton; // button with icons
16
17    // ButtonFrame adds JButtons to JFrame
18    public ButtonFrame()
19    {
20       super("Testing Buttons");
21       setLayout(new FlowLayout());
22
23       plainJButton = new JButton("Plain Button"); // button with text
24       add(plainJButton); // add plainJButton to JFrame
25
26       Icon bug1 = new ImageIcon(getClass().getResource("bug1.gif"));
27       Icon bug2 = new ImageIcon(getClass().getResource("bug2.gif"));
28       fancyJButton = new JButton("Fancy Button", bug1); // set image
29       fancyJButton.setRolloverIcon(bug2); // set rollover image
30       add(fancyJButton); // add fancyJButton to JFrame
31
32       // create new ButtonHandler for button event handling
33       ButtonHandler handler = new ButtonHandler();
34       fancyJButton.addActionListener(handler);
35       plainJButton.addActionListener(handler);
36    }
37
38    // inner class for button event handling
39    private class ButtonHandler implements ActionListener
40    {
41       // handle button event
42       @Override
43       public void actionPerformed(ActionEvent event)
44       {
45          JOptionPane.showMessageDialog(ButtonFrame.this, String.format(
46             "You pressed: %s", event.getActionCommand()));
47       }
48    }
49 } // end class ButtonFrame
```

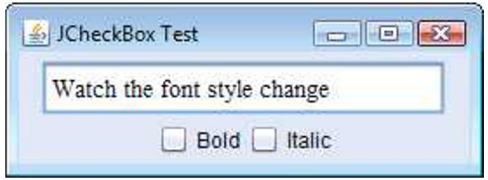
```
1  // Fig. 12.16: ButtonTest.java
2  // Testing ButtonFrame.
3  import javax.swing.JFrame;
4
5  public class ButtonTest
6  {
7     public static void main(String[] args)
8     {
9        ButtonFrame buttonFrame = new ButtonFrame();
10       buttonFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11       buttonFrame.setSize(275, 110);
12       buttonFrame.setVisible(true);
13    }
14 } // end class ButtonTest
```

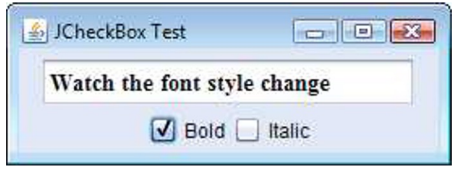
JCheckBox

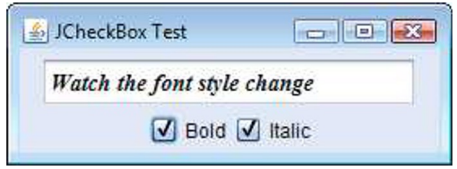
```
1  // Fig. 12.17: CheckBoxFrame.java
2  // JCheckBoxes and item events.
3  import java.awt.FlowLayout;
4  import java.awt.Font;
5  import java.awt.event.ItemListener;
6  import java.awt.event.ItemEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JTextField;
9  import javax.swing.JCheckBox;
10
11 public class CheckBoxFrame extends JFrame
12 {
13    private final JTextField textField; // displays text in changing fonts
14    private final JCheckBox boldJCheckBox; // to select/deselect bold
15    private final JCheckBox italicJCheckBox; // to select/deselect italic
16
17    // CheckBoxFrame constructor adds JCheckBoxes to JFrame
18    public CheckBoxFrame()
19    {
20       super("JCheckBox Test");
21       setLayout(new FlowLayout());
22
23       // set up JTextField and set its font
24       textField = new JTextField("Watch the font style change", 20);
25       textField.setFont(new Font("Serif", Font.PLAIN, 14));
26       add(textField); // add textField to JFrame
27
28       boldJCheckBox = new JCheckBox("Bold");
29       italicJCheckBox = new JCheckBox("Italic");
30       add(boldJCheckBox); // add bold checkbox to JFrame
31       add(italicJCheckBox); // add italic checkbox to JFrame
32
33       // register listeners for JCheckBoxes
34       CheckBoxHandler handler = new CheckBoxHandler();
35       boldJCheckBox.addItemListener(handler);
36       italicJCheckBox.addItemListener(handler);
37    }
38
39    // private inner class for ItemListener event handling
40    private class CheckBoxHandler implements ItemListener
41    {
```

```
42       // respond to checkbox events
43       @Override
44       public void itemStateChanged(ItemEvent event)
45       {
46          Font font = null; // stores the new Font
47
48          // determine which CheckBoxes are checked and create Font
49          if (boldJCheckBox.isSelected() && italicJCheckBox.isSelected())
50             font = new Font("Serif", Font.BOLD + Font.ITALIC, 14);
51          else if (boldJCheckBox.isSelected())
52             font = new Font("Serif", Font.BOLD, 14);
53          else if (italicJCheckBox.isSelected())
54             font = new Font("Serif", Font.ITALIC, 14);
55          else
56             font = new Font("Serif", Font.PLAIN, 14);
57
58          textField.setFont(font);
59       }
60    }
61 } // end class CheckBoxFrame
```

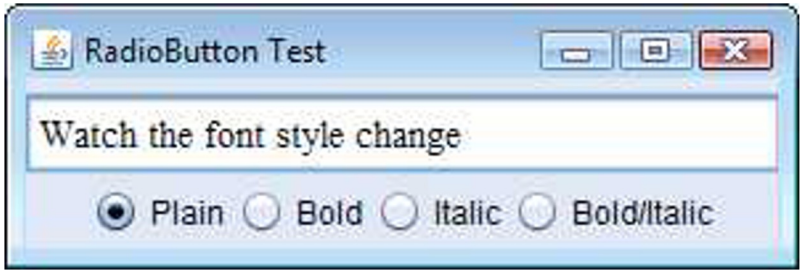
```
1  // Fig. 12.18: CheckBoxTest.java
2  // Testing CheckBoxFrame.
3  import javax.swing.JFrame;
4
5  public class CheckBoxTest
6  {
7     public static void main(String[] args)
8     {
9        CheckBoxFrame checkBoxFrame = new CheckBoxFrame();
10       checkBoxFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11       checkBoxFrame.setSize(275, 100);
12       checkBoxFrame.setVisible(true);
13    }
14 } // end class CheckBoxTest
```

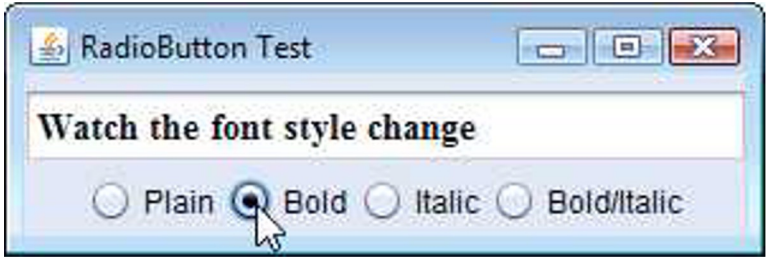




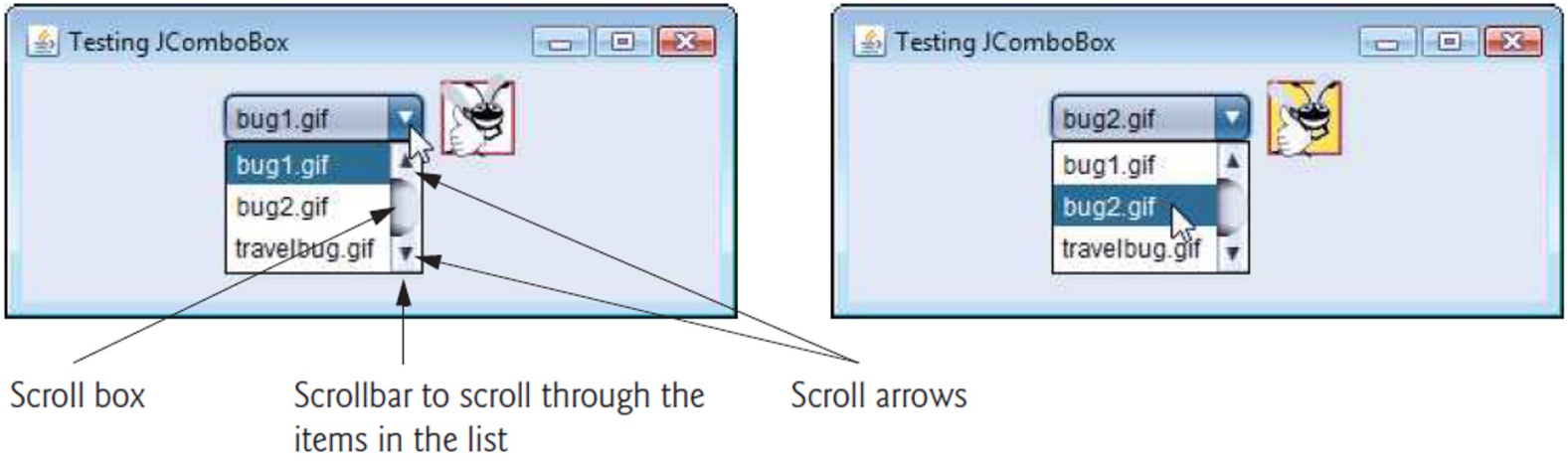


JRadioButton

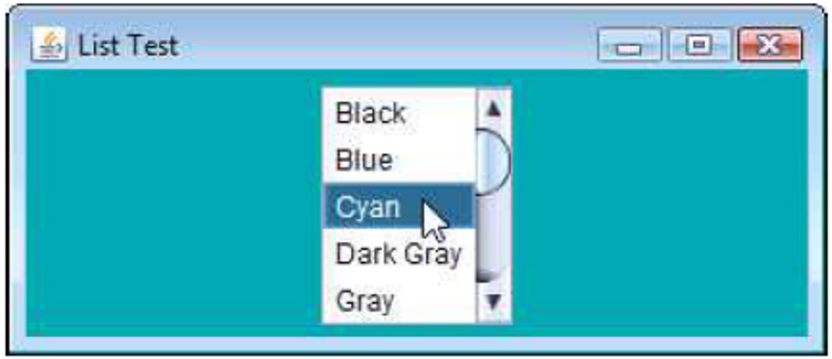


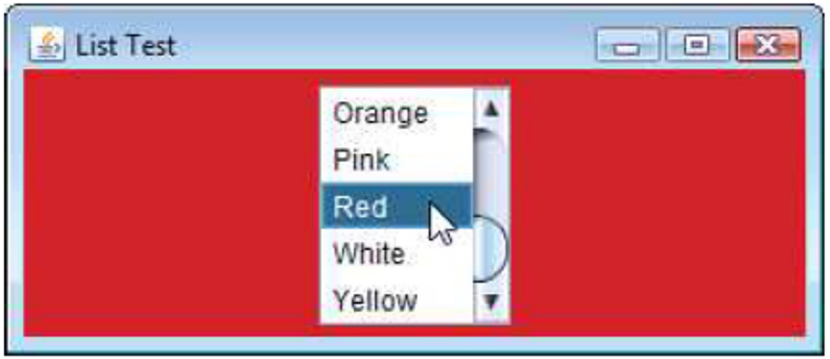


JComboBox

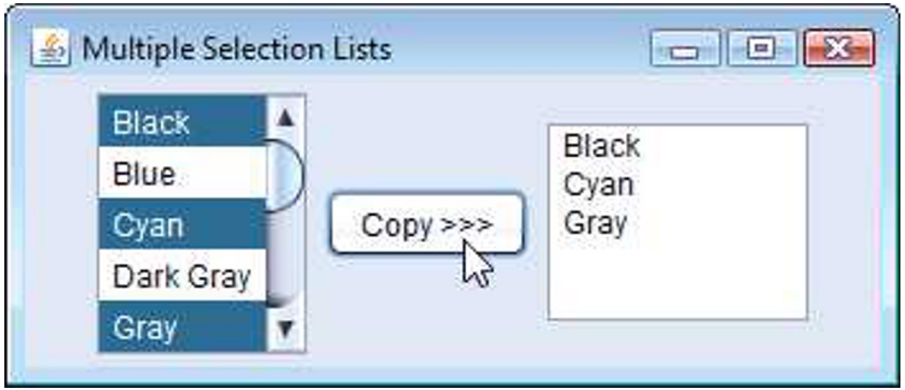

Testing JComboBox
bug1.gif
bug1.gif
bug2.gif
travelbug.gif
Testing JComboBox
bug2.gif
bug1.gif
bug2.gif
travelbug.gif
Scroll box
Scrollbar to scroll through the items in the list
Scroll arrows
Testing JComboBox
travelbug.gif
bug2.gif
travelbug.gif
buganim.gif
Testing JComboBox
buganim.gif
bug2.gif
travelbug.gif
buganim.gif

JList





Selecting colors from a `JList`.



Testing `MultipleSelectionFrame`.

Mouse Event Handling

MouseListener and MouseMotionListener interface methods

Methods of interface `MouseListener`

`public void mousePressed(MouseEvent event)`

Called when a mouse button is *pressed* while the mouse cursor is on a component.

`public void mouseClicked(MouseEvent event)`

Called when a mouse button is *pressed and released* while the mouse cursor remains stationary on a component. Always preceded by a call to `mousePressed` and `mouseReleased`.

`public void mouseReleased(MouseEvent event)`

Called when a mouse button is *released after being pressed*. Always preceded by a call to `mousePressed` and one or more calls to `mouseDragged`.

`public void mouseEntered(MouseEvent event)`

Called when the mouse cursor *enters* the bounds of a component.

`public void mouseExited(MouseEvent event)`

Called when the mouse cursor *leaves* the bounds of a component.

Methods of interface `MouseMotionListener`

`public void mouseDragged(MouseEvent event)`

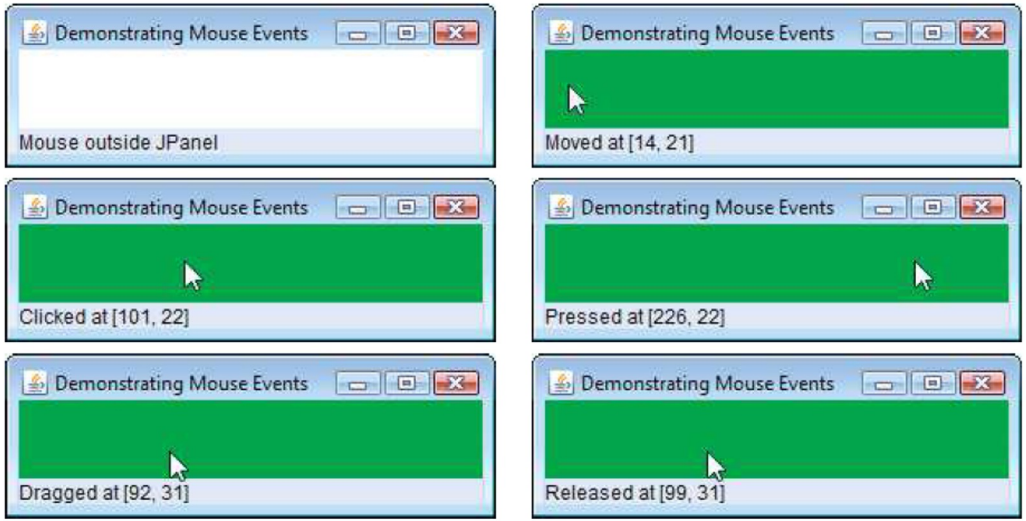
Called when the mouse button is *pressed* while the mouse cursor is on a component and the mouse is *moved* while the mouse button *remains pressed*. Always preceded by a call to `mousePressed`. All drag events are sent to the component on which the user began to drag the mouse.

`public void mouseMoved(MouseEvent event)`

Called when the mouse is *moved* (with no mouse buttons pressed) when the mouse cursor is on a component. All move events are sent to the component over which the mouse is currently positioned.

```java
// Fig. 12.28: MouseTrackerFrame.java
// Mouse event handling.
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class MouseTrackerFrame extends JFrame
{
   private final JPanel mousePanel; // panel in which mouse events occur
   private final JLabel statusBar; // displays event information

   // MouseTrackerFrame constructor sets up GUI and
   // registers mouse event handlers
   public MouseTrackerFrame()
   {
      super("Demonstrating Mouse Events");

      mousePanel = new JPanel();
      mousePanel.setBackground(Color.WHITE);
      add(mousePanel, BorderLayout.CENTER); // add panel to JFrame

      statusBar = new JLabel("Mouse outside JPanel");
      add(statusBar, BorderLayout.SOUTH); // add label to JFrame

      // create and register listener for mouse and mouse motion events
      MouseHandler handler = new MouseHandler();
      mousePanel.addMouseListener(handler);
      mousePanel.addMouseMotionListener(handler);
   }

   private class MouseHandler implements MouseListener,
      MouseMotionListener
   {
      // MouseListener event handlers
      // handle event when mouse released immediately after press
      @Override
      public void mouseClicked(MouseEvent event)'
      {
         statusBar.setText(String.format("Clicked at [%d, %d]",
            event.getX(), event.getY()));
      }

      // handle event when mouse pressed
      @Override
      public void mousePressed(MouseEvent event)
      {
         statusBar.setText(String.format("Pressed at [%d, %d]",
            event.getX(), event.getY()));
      }

      // handle event when mouse released
      @Override
      public void mouseReleased(MouseEvent event)
      {
         statusBar.setText(String.format("Released at [%d, %d]",
            event.getX(), event.getY()));
      }

      // handle event when mouse enters area
      @Override
      public void mouseEntered(MouseEvent event)
      {
         statusBar.setText(String.format("Mouse entered at [%d, %d]",
            event.getX(), event.getY()));
         mousePanel.setBackground(Color.GREEN);
      }

      // handle event when mouse exits area
      @Override
      public void mouseExited(MouseEvent event)
      {
         statusBar.setText("Mouse outside JPanel");
         mousePanel.setBackground(Color.WHITE);
      }

      // MouseMotionListener event handlers
      // handle event when user drags mouse with button pressed
      @Override
      public void mouseDragged(MouseEvent event)
      {
         statusBar.setText(String.format("Dragged at [%d, %d]",
            event.getX(), event.getY()));
      }

      // handle event when user moves mouse
      @Override
      public void mouseMoved(MouseEvent event)
      {
         statusBar.setText(String.format("Moved at [%d, %d]",
            event.getX(), event.getY()));
      }
   } // end inner class MouseHandler
} // end class MouseTrackerFrame
```

```java
// Fig. 12.29: MouseTrackerFrame.java
// Testing MouseTrackerFrame.
import javax.swing.JFrame;

public class MouseTracker
{
   public static void main(String[] args)
   {
      MouseTrackerFrame mouseTrackerFrame = new MouseTrackerFrame();
      mouseTrackerFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      mouseTrackerFrame.setSize(300, 100);
      mouseTrackerFrame.setVisible(true);
   }
} // end class MouseTracker
```



JPanel Subclass for Drawing with the Mouse

```java
// Fig. 12.34: PaintPanel.java
// Adapter class used to implement event handlers.
import java.awt.Point;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import java.util.ArrayList;
import javax.swing.JPanel;

public class PaintPanel extends JPanel
{
   // list of Point references
   private final ArrayList<Point> points = new ArrayList<>();

   // set up GUI and register mouse event handler
   public PaintPanel()
   {
      // handle frame mouse motion event
      addMouseMotionListener(
         new MouseMotionAdapter() // anonymous inner class
         {
            // store drag coordinates and repaint
            @Override
            public void mouseDragged(MouseEvent event)
            {
               points.add(event.getPoint());
               repaint(); // repaint JFrame
            }
         }
      );
   }

   // draw ovals in a 4-by-4 bounding box at specified locations on window
   @Override
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g); // clears drawing area

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

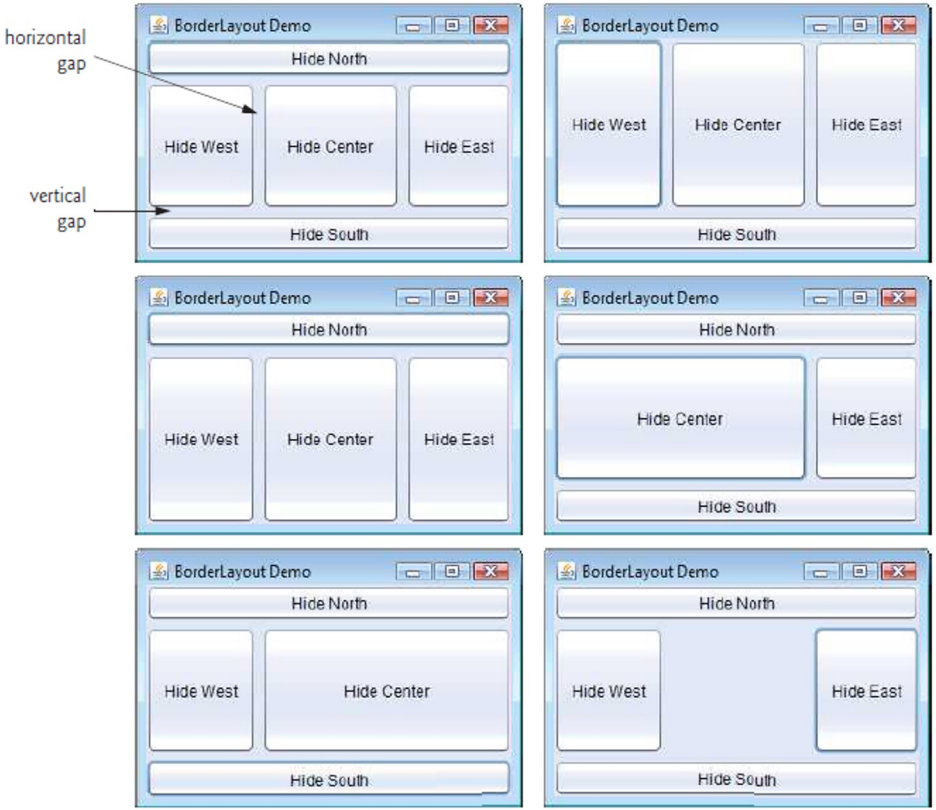
```java
// Fig. 12.35: Painter.java
// Testing PaintPanel.
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;

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
```

Key Event Handling

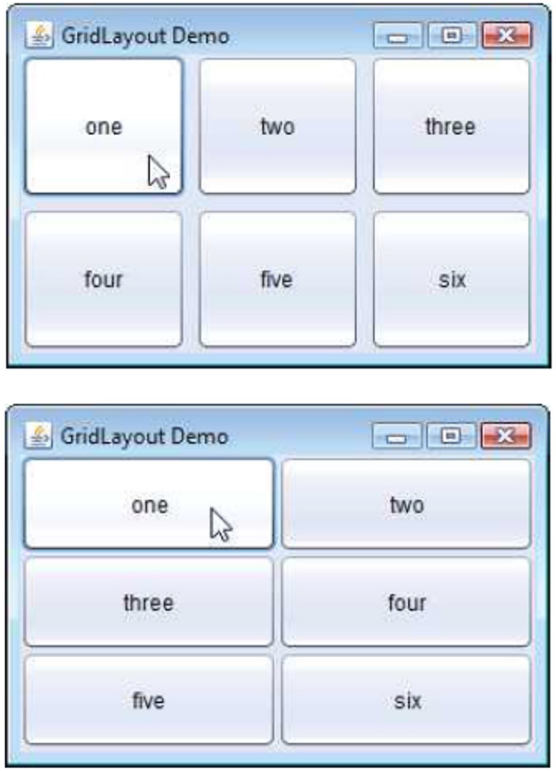
```java
// Fig. 12.36: KeyDemoFrame.java
// Key event handling.
import java.awt.Color;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import javax.swing.JFrame;
import javax.swing.JTextArea;

public class KeyDemoFrame extends JFrame implements KeyListener
{
   private final String line1 = ""; // first line of text area
   private final String line2 = ""; // second line of text area
   private final String line3 = ""; // third line of text area
   private final JTextArea textArea; // text area to display output

   // KeyDemoFrame constructor
   public KeyDemoFrame()
   {
      super("Demonstrating Keystroke Events");

      textArea = new JTextArea(10, 15); // set up JTextArea
      textArea.setText("Press any key on the keyboard...");
      textArea.setEnabled(false);
      textArea.setDisabledTextColor(Color.BLACK);
      add(textArea); // add text area to JFrame

      addKeyListener(this); // allow frame to process key events
   }

   // handle press of any key
   @Override
   public void keyPressed(KeyEvent event)
   {
      line1 = String.format("Key pressed: %s",
         KeyEvent.getKeyText(event.getKeyCode())); // show pressed key
      setLines2and3(event); // set output lines two and three
   }

   // handle release of any key
   @Override
   public void keyReleased(KeyEvent event)
   {
      line1 = String.format("Key released: %s",
         KeyEvent.getKeyText(event.getKeyCode())); // show released key
      setLines2and3(event); // set output lines two and three
   }

   // handle press of an action key
   @Override
   public void keyTyped(KeyEvent event)
   {
      line1 = String.format("Key typed: %s", event.getKeyChar());
      setLines2and3(event); // set output lines two and three
   }

   // set second and third lines of output
   private void setLines2and3(KeyEvent event)
   {
      line2 = String.format("This key is %san action key",
         (event.isActionKey() ? "" : "not "));

      String temp = KeyEvent.getKeyModifiersText(event.getModifiers());

      line3 = String.format("Modifier keys pressed: %s",
         (temp.equals("") ? "none" : temp)); // output modifiers

      textArea.setText(String.format("%s\n%s\n%s\n",
         line1, line2, line3)); // output three lines of text
   }
} // end class KeyDemoFrame
```

```java
// Fig. 12.37: KeyDemo.java
// Testing KeyDemoFrame.
import javax.swing.JFrame;

public class KeyDemo
{
   public static void main(String[] args)
   {
      KeyDemoFrame keyDemoFrame = new KeyDemoFrame();
      keyDemoFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      keyDemoFrame.setSize(350, 100);
      keyDemoFrame.setVisible(true);
   }
} // end class KeyDemo
```

BorderLayoutFrame & GridLayout Frame



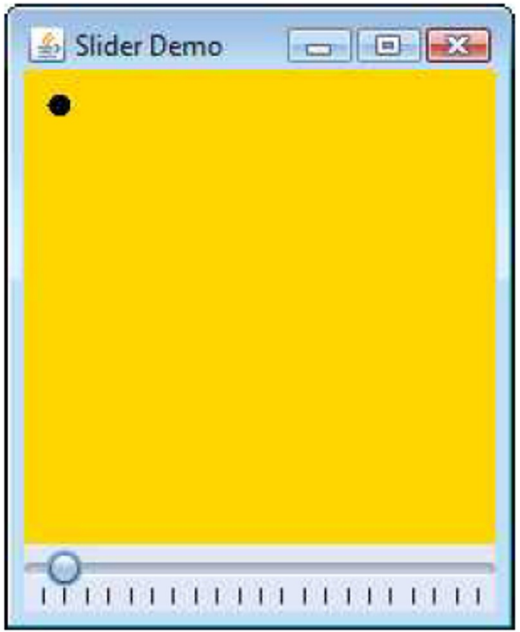
Testing BorderLayoutFrame



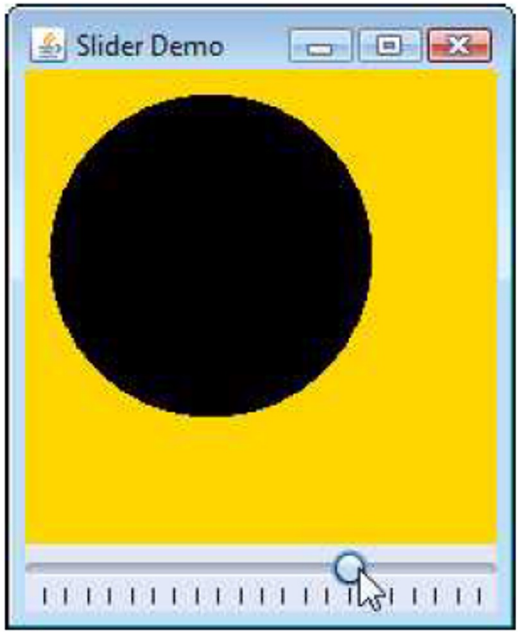
Testing GridLayoutFrame

JSlider

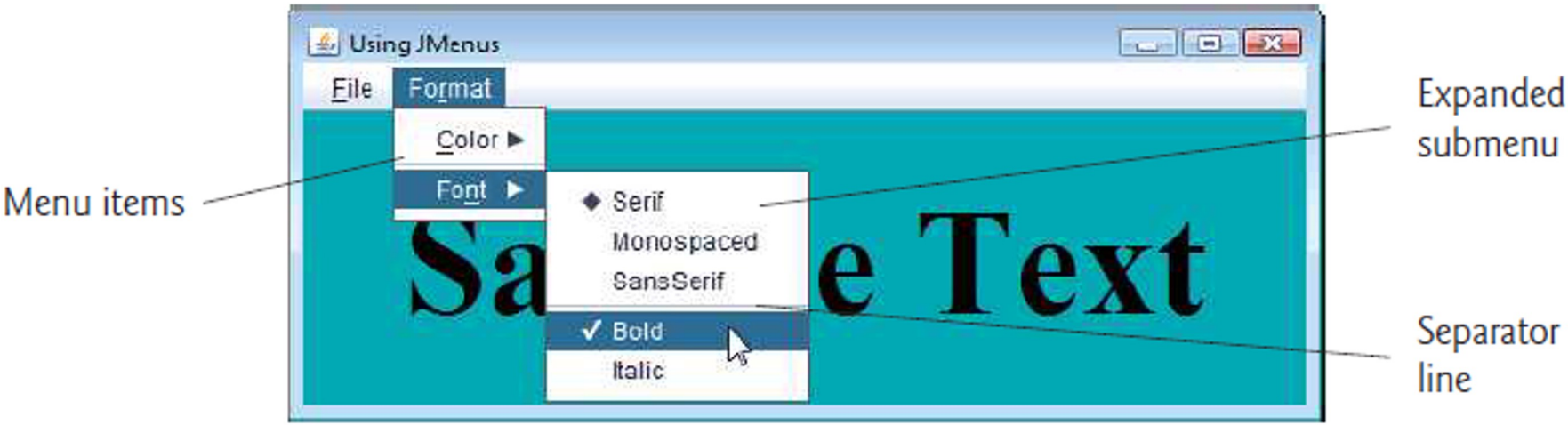
a) Initial GUI with default circle



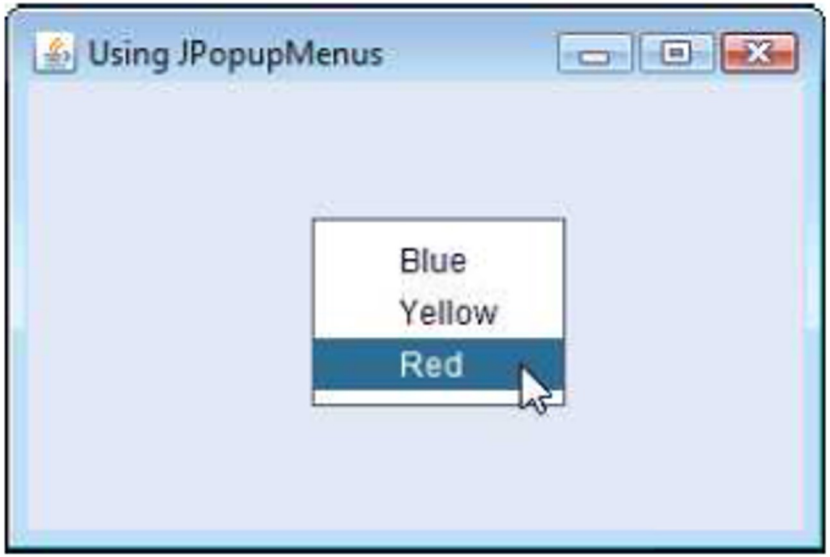
b) GUI after the user moves the `JSlider`'s thumb to the right

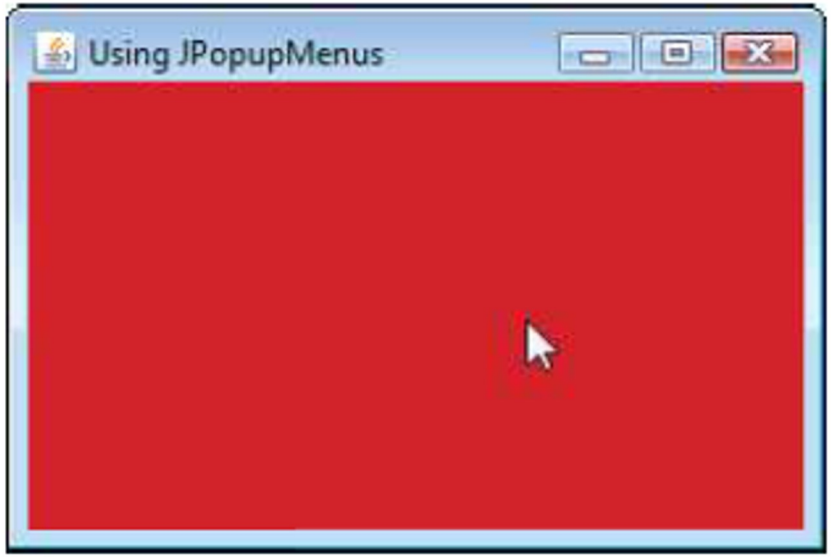


MenuFrame

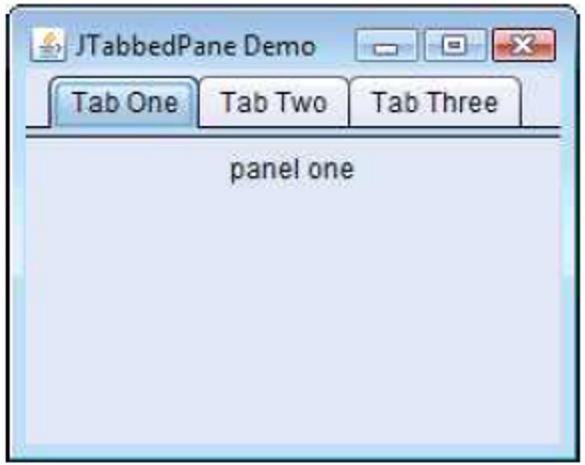

Using JMenus
File
Format
Color ▶
Font ▶
Serif
Monospaced
SansSerif
Bold
Italic
Text
Menu items
Expanded submenu
Separator line

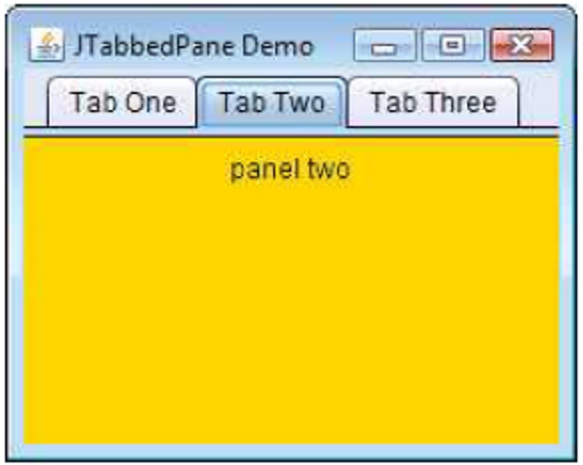
PopupFrame

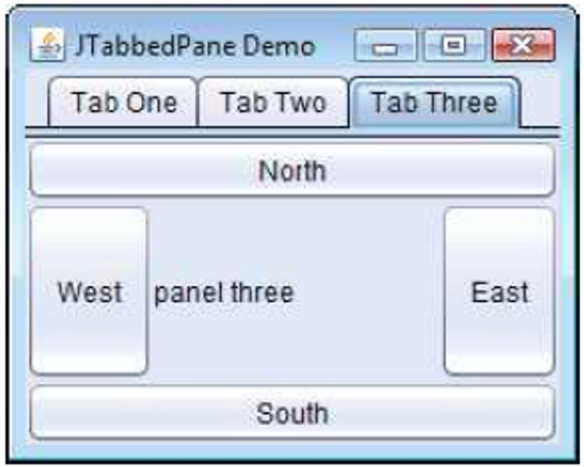




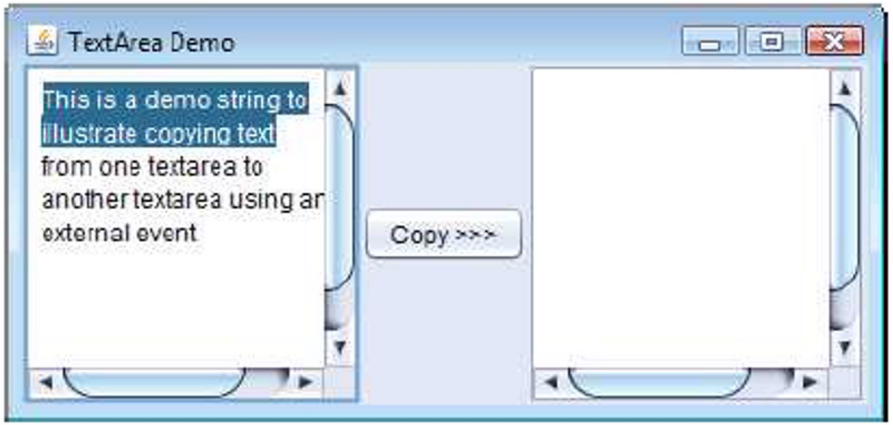
JTabbedPaneFrame

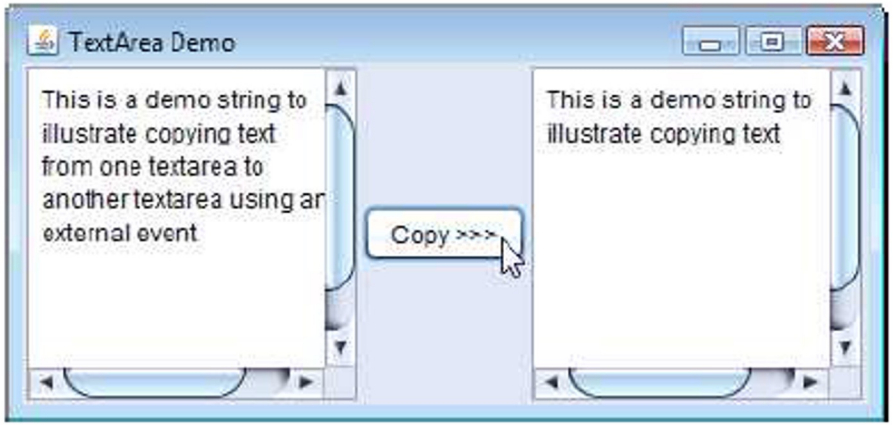






JTextArea


TextArea Demo
This is a demo string to
illustrate copying text
from one textarea to
another textarea using an
external event
Copy >>>


TextArea Demo
This is a demo string to
illustrate copying text
from one textarea to
another textarea using an
external event
Copy >>>
This is a demo string to
illustrate copying text

Recap

- **Mid-term Exam**
  - Possible Topics
  - Format
- **GUI** in Java
  - Swing Components
  - Swing vs JavaFX vs AWT
  - JOption-Pane (DialogBox)
  - Java 2D API
    - Java Coordinate System
    - Draw Lines
    - Draw Rectangles and Ovals
    - Draw Colors
    - Draw Polygons and Polylines
  - Text Fields, Buttons and Menu Nagivations.
  - Event Handler (Mouse & Keyboard)
  - Layout:
    - BorderLayout Frame & GridLayoutFrame

Game Tutorials with Java Swing

- RPG Adventure: https://www.youtube.com/watch?v=om59cwR7psI&list=PL_QPQmz5C6WUF-pOQDsbsKbaBZqXj4qSq&ab_channel=RyiSnow
- Jave 2D tile Game: https://www.youtube.com/watch?v=4D3YIYPkit4&list=PL_QPQmz5C6WVtEOazEZ_0r9kewAqvUf2Z&ab_channel=RyiSnow
- Teris with Jave Swing: https://www.youtube.com/watch?v=dgVh6S8X25k&list=PL8wKnrrMApCoP9xTwOmQRHsZ8_Kr9c_DY
- Chess with Java Swing: https://www.youtube.com/watch?v=vO7wHV0HB8w&list=PLnEZvjtYfz6uvTjfNcklgDMnUZTKid6zx&ab_channel=ScreenWorks
- Snake game: https://www.youtube.com/watch?v=bI6e6qjJ8JQ&t=1967s&ab_channel=BroCode
- Tic Tac Toe: https://www.youtube.com/watch?v=rA7tfvpkw0I&ab_channel=BroCode

Java Game Libraries

- LWJGL:

  https://www.lwjgl.org/frameworks

- Jmonkeyengine:

  https://jmonkeyengine.org/

- libGDX:

  https://libgdx.com/

Other Popular Game Engine

- Unity3D:
  - You need to learn C#.
  - https://unity.com/madewith

- UnrealEngine:
  - You need to learn C++.
  - https://www.unrealengine.com/en-US/

Thank you for your listening!

“Motivation is what gets you started.
Habit is what keeps you going!”

Jim Ryun