Week10

Programming Fundamentals II

Course Outline

8. GUI	เสนอหัวข้อ Project
9. Event	Lab7
10. Graphic	Lab8
11. Testing and debugging	Lab9
12. Generics	Lab10
13. Team project	เสนอความคืบหน้า Project
14. Concurrency	-

Final Exam Lab Exam Present Project

Week 10 **Graphic**

1. Simple Drawings
2. Drawing Rectangles and Ovals
3. Colors and Filled Shapes
4. Drawing Arcs
5. Displaying Text and Images Using Labels
6. Drawing with Polymorphism
7. Keyboard Event
8. Java Project Example

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

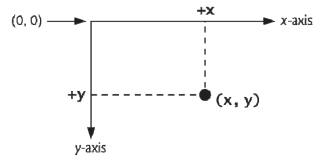


Fig. 4.17 | Java coordinate system. Units are measured in pixels.

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.)

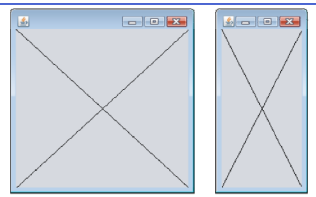


Fig. 4.19 | Creating `JFrame` to display `DrawPanel`. (Part 2 of 2.)

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.)

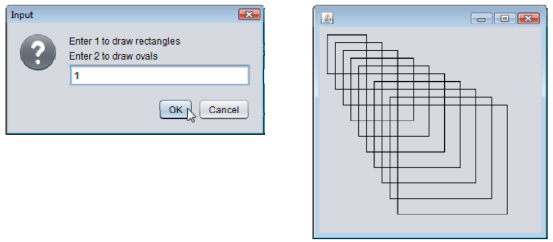


Fig. 5.27 | Obtaining user input and creating a `JFrame` to display `Shapes`. (Part 3 of 4.)

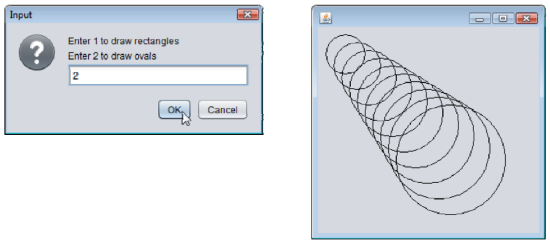


Fig. 5.27 | Obtaining user input and creating a `JFrame` to display `Shapes`. (Part 4 of 4.)

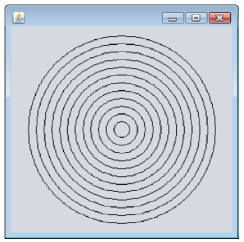
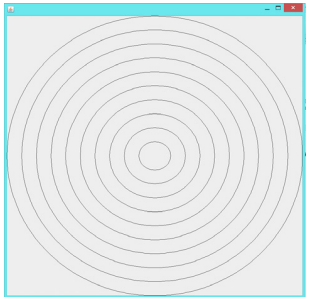


Fig. 5.28 | Drawing concentric circles.

DrawCircle



```
public void paintComponent( Graphics g)
{
    super.paintComponent(g);

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

    int gapWidth    = width/20;
    int gapHeight   = height/20;

    int xOrigins = (width/10) - (gapWidth*2);
    int yOrigins = (height/10) - (gapHeight*2);

    int gapRatioX = (width/10) - (xOrigins*2);
    int gapRatioY = (height/10) - (yOrigins*2);

    for(int i = 0 ; i < 10 ; i++)
    {
        switch(this.choice)
        {
            case 1 :
                g.drawRect(xOrigins + i*gapWidth, yOrigins + i*gapHeight,
                        (width - (xOrigins)) - i*gapRatioX, (height - (yOrigins)) - i*gapRatioY);
                break;
            case 2 :
                g.drawOval(xOrigins + i*gapWidth, yOrigins + i*gapHeight,
                        (width - (xOrigins)) - i*gapRatioX, (height - (yOrigins)) - i*gapRatioY);
                break;
        }
    }
}
```


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.)

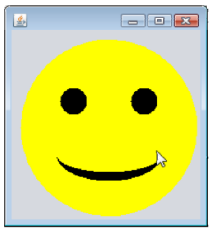


Fig. 6.12 | Creating `JFrame` to display a smiley face. (Part 2 of 2.)

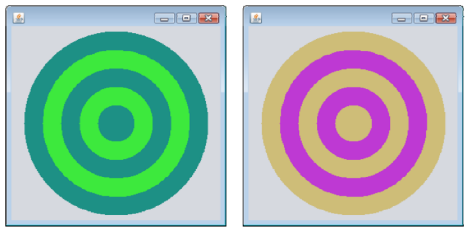
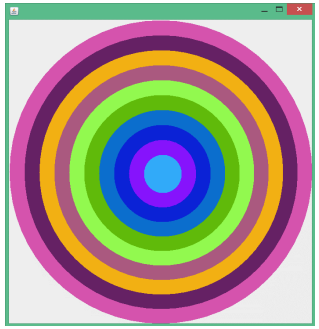


Fig. 6.13 | A bull's-eye with two alternating, random colors.

FillCircle



```
public void paintComponent( Graphics g)
{
    super.paintComponent(g);

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

    int gapWidth    = width/20;
    int gapHeight   = height/20;

    int xOrigins = (width/10) - (gapWidth*2);
    int yOrigins = (height/10) - (gapHeight*2);

    int gapRatioX = (width/10) - (xOrigins*2);
    int gapRatioY = (height/10) - (yOrigins*2);

    for(int i = 0 ; i < 10 ; i++)
    {
        Random rand = new Random();
        float red   = rand.nextFloat();
        float green = rand.nextFloat();
        float blue  = rand.nextFloat();
        Color c = new Color(red, green, blue);
        switch(this.choice)
        {
            case 1 :
                g.setColor(c);
                g.fillRect(xOrigins + i*gapWidth, yOrigins + i*gapHeight,
                        (width - (xOrigins)) - i*gapRatioX, (height - (yOrigins)) - i*gapRatioY);
                break;
            case 2 :
                g.setColor(c);
                g.fillOval(xOrigins + i*gapWidth, yOrigins + i*gapHeight,
                        (width - (xOrigins)) - i*gapRatioX, (height - (yOrigins)) - i*gapRatioY);
                break;
        }
    }
}
```

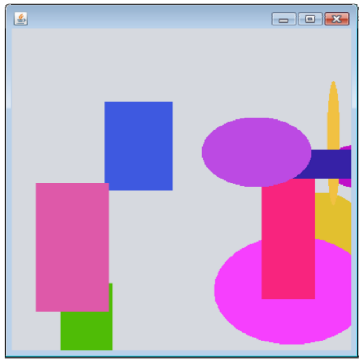
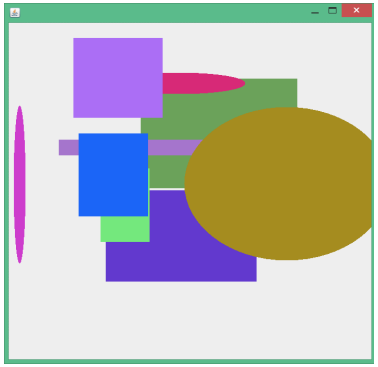



Fig. 6.14 | Randomly generated shapes.

RandomFillShape



```
public void paintComponent( Graphics g)
{
    super.paintComponent(g);

    Random rand = new Random();

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


    for(int i = 0 ; i < 10 ; i++)
    {
        int shape = rand.nextInt(2)+1;

        float red   = rand.nextFloat();
        float green = rand.nextFloat();
        float blue  = rand.nextFloat();

        Color c = new Color(red, green, blue);

        switch(shape)
        {
            case 1 :
                g.setColor(c);
                g.fillRect(rand.nextInt(width/2),rand.nextInt(height/2),
                        rand.nextInt((int)(width*0.6)),rand.nextInt((int)(height*0.6)));
                //rand.nextInt(width/2),rand.nextInt(height/2));
                break;
            case 2 :
                g.setColor(c);
                g.fillOval(rand.nextInt(width/2),rand.nextInt(height/2),
                        rand.nextInt((int)(width*0.6)),rand.nextInt((int)(height*0.6)));
                break;
        }
    }
}
```



```
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.)

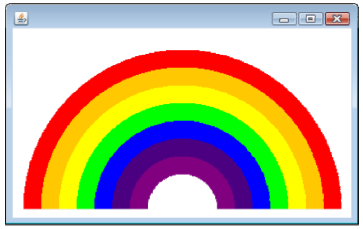


Fig. 7.26 | Creating `JFrame` to display a rainbow. (Part 2 of 2.)

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.

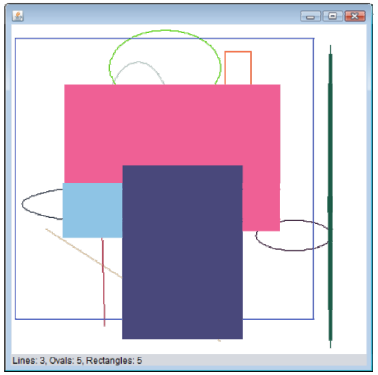
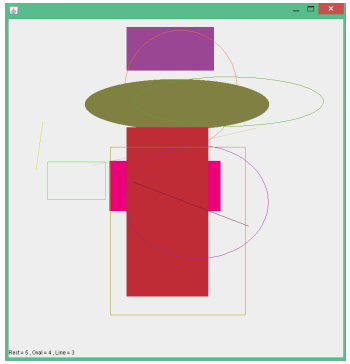


Fig. 9.14 | `JLabel` displaying shape statistics.

Random Draw+Fill (Oval+Rect+Line) 1/2

```
public void paintComponent( Graphics g)
{
    super.paintComponent(g);
    Random rand = new Random();
    int width = getWidth();
    int height = getHeight();
    int sumRect = 0;
    int sumOval = 0;
    int sumLine = 0;
    for(int i = 0 ; i < (rand.nextInt(21)+5) ; i++)
    {
        int shape = rand.nextInt(5)+1;
        float red   = rand.nextFloat();
        float green = rand.nextFloat();
        float blue  = rand.nextFloat();
        Color c = new Color(red, green, blue);
        switch(shape)
        {
            case 1 :
                g.setColor(c);
                g.fillRect(rand.nextInt(width/2),rand.nextInt(height/2),
                        rand.nextInt((int)(width*0.6)),rand.nextInt((int)(height*0.6)));
                        //rand.nextInt(width/2),rand.nextInt(height/2));
                sumRect += 1;
                break;
            case 2 :
                g.setColor(c);
                g.drawRect(rand.nextInt(width/2),rand.nextInt(height/2),
                        rand.nextInt((int)(width*0.6)),rand.nextInt((int)(height*0.6)));
                sumRect += 1;
                break;
```



Random Draw+Fill (Oval+Rect+Line) 2/2

```
            case 3 :
                g.setColor(c);
                g.fillOval(rand.nextInt(width/2),rand.nextInt(height/2),
                        rand.nextInt((int)(width*0.6)),rand.nextInt((int)(height*0.6)));
                sumOval += 1;
                break;
            case 4 :
                g.setColor(c);
                g.drawOval(rand.nextInt(width/2),rand.nextInt(height/2),
                        rand.nextInt((int)(width*0.6)),rand.nextInt((int)(height*0.6)));
                sumOval += 1;
                break;
            case 5 :
                g.setColor(c);
                g.drawLine(rand.nextInt(width/2),rand.nextInt(height/2),
                        rand.nextInt((int)(width*0.9)),rand.nextInt((int)(height*0.9)));
                sumLine += 1;
                break;
        }
    }
    g.setColor(Color.BLACK);
    String str = "Rect = " + sumRect + " , Oval = " + sumOval + " , Line = " + sumLine;
    g.drawString(str, 0, height - 7);
}
```

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.

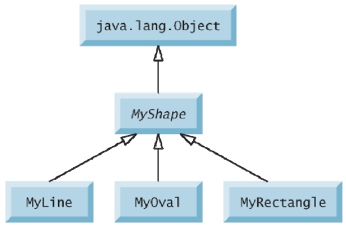


Fig. 10.17 | **MyShape** hierarchy.

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`.

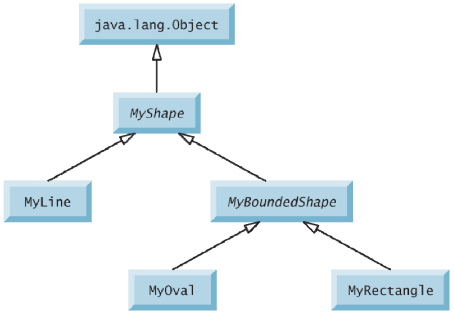
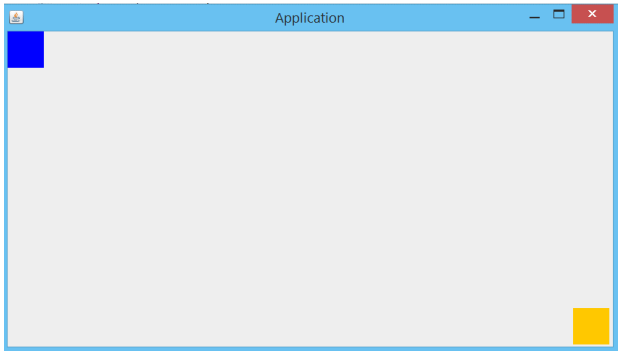


Fig. 10.18 | **MyShape** hierarchy with **MyBoundedShape**.

MovingObject

ActionListener

KeyListener



MoveObject.java (1/4)

```
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class MoveObject extends JPanel implements ActionListener, KeyListener
8  {
9      Timer tm = new Timer(5,this);        // Timer Swing
10
11     int x1 = 780, y1 = 380, velX1 = 0, velY1 = 0;
12     int x2 = 0, y2 = 0, velX2 = 0, velY2 = 0;
13
14     public MoveObject()
15     {
16         tm.start();
17         addKeyListener(this);
18         setFocusable(true);
19         setFocusTraversalKeysEnabled(false);
20     }
21
22     public void paintComponent(Graphics g)
23     {
24         super.paintComponent(g);
25         g.setColor(Color.ORANGE);
26         g.fillRect(x1, y1, 50, 50);
27
28         g.setColor(Color.BLUE);
29         g.fillRect(x2, y2, 50, 50);
30
31     }
```


MoveObject.java (2/4)

```
34     @Override
35     public void actionPerformed(ActionEvent e)
36     {
37         x1 = x1 + velX1;
38         y1 = y1 + velY1;
39
40         x2 = x2 + velX2;
41         y2 = y2 + velY2;
42         repaint();
43
44     }
45
46     @Override
47     public void keyTyped(KeyEvent e)
48     {
49         // TODO Auto-generated method stub
50
51     }
52
53     @Override
54     public void keyPressed(KeyEvent e)
55     {
56         int key = e.getKeyCode();
57
58         if(key == KeyEvent.VK_LEFT)
59         {
60             velX1 = -1;
61             velY1 = 0;
62         }
63         if(key == KeyEvent.VK_UP)
64         {
65             velX1 = 0;
66             velY1 = -1;
67         }
```


MoveObject.java (3/4)

```
68         if(key == KeyEvent.VK_RIGHT)
69         {
70             velX1 = 1;
71             velY1 = 0;
72         }
73         if(key == KeyEvent.VK_DOWN)
74         {
75             velX1 = 0;
76             velY1 = 1;
77         }
78
79         if(key == KeyEvent.VK_A)
80         {
81             velX2 = -1;
82             velY2 = 0;
83         }
84         if(key == KeyEvent.VK_W)
85         {
86             velX2 = 0;
87             velY2 = -1;
88         }
89         if(key == KeyEvent.VK_D)
90         {
91             velX2 = 1;
92             velY2 = 0;
93         }
94         if(key == KeyEvent.VK_S)
95         {
96             velX2 = 0;
97             velY2 = 1;
98         }
99
100    }
```

MoveObject.java (4/4)

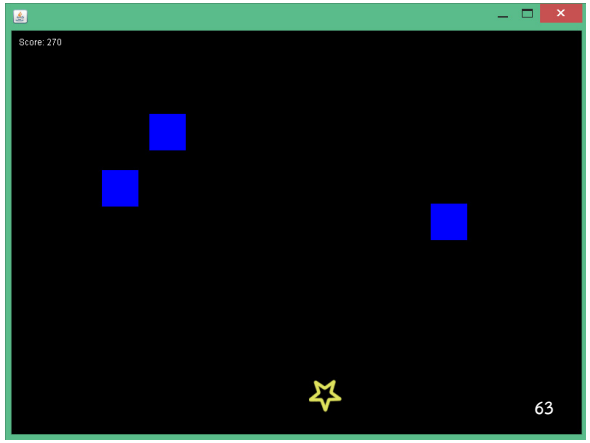
```
    @Override
    public void keyReleased(KeyEvent e)
    {
        int key = e.getKeyCode();

        if((key == KeyEvent.VK_LEFT) || (key == KeyEvent.VK_UP) || (key == KeyEvent.VK_RIGHT) || (key == KeyEvent.VK_DOWN))
        {
            velX1 = 0;
            velY1 = 0;
        }
        if((key == KeyEvent.VK_A) || (key == KeyEvent.VK_W) || (key == KeyEvent.VK_D) || (key == KeyEvent.VK_S))
        {
            velX2 = 0;
            velY2 = 0;
        }

    }

    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Application");
        MoveObject panel = new MoveObject();

        frame.add(panel);
        frame.setSize(853, 480);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

    }
}
```


Java Project Ex

Programming Fundamentals II

Game Project Example (Ref Aj.Poona)

- P10_11G
  - Alien.java
  - BasicGame.java
  - Bullet.java
  - GameObject.java
  - GameScene.java
  - SpaceShip.java



BasicGame.java

```
/**
 * Created by poonna on 4/24/15 AD.
 */
import java.awt.*;
import javax.swing.JFrame;

public class BasicGame {
    public static void main(String[] args) {
        JFrame game = new JFrame();
        game.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        game.setSize(800, 600);
        game.add(new GameScene(), BorderLayout.CENTER);
        game.setVisible(true);
    }
}
```

GameScene.java

public void update()

paintComponent(Graphics g)

public boolean getKeyState(int key)

public void addScore(int points)

Implement Action + Key Listener

```java
/**
 * Created by poonna on 4/19/15 AD.
 */
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;
import java.util.Random;
import javax.swing.JPanel;
import javax.swing.Timer;

public class GameScene extends JPanel implements ActionListener, KeyListener {
    public static final int KEY_UP = 0;
    public static final int KEY_DOWN = 1;
    public static final int KEY_LEFT = 2;
    public static final int KEY_RIGHT = 3;
    public static final int KEY_FIRE = 4;

    private boolean[] keyStates = new boolean[5];

    private SpaceShip player;
    private ArrayList<GameObject> enemies = new ArrayList<GameObject>();
    private ArrayList<GameObject> bullets = new ArrayList<GameObject>();

    private Random random;
    private int score;

    public GameScene() {
        setBackground(Color.BLACK);
        setDoubleBuffered(true);
```


GameObject.java

```java
/**
 * Created by poonna on 4/24/15 AD.
 */
import java.awt.Graphics;
import java.awt.Rectangle;

public abstract class GameObject {
    // update() returns true in most cases, and false when it needs to destroy itself
    public abstract boolean update();

    public abstract void draw(Graphics g);

    public abstract Rectangle getBounds();

    public boolean collidedWith(GameObject other) {
        return getBounds().intersects(other.getBounds());
    }
}
```

- Alien.java
- Bullet.java
- SpaceShip.java

Alien.java

```java
public class Alien extends GameObject {
    private GameScene scene;
    private int x, y;
    private int speedX = 10;
    private int speedY = 10;

    public Alien(GameScene scene, int x, int y) {
        this.scene = scene;
        this.x = x;
        this.y = y;
    }
    @Override
    public boolean update() {
        if (x < 100) {
            speedX = 10;
        } else if (x > 700) {
            speedX = -10;
        }
        if (y < 100) {
            speedY = 10;
        } else if (y > 300) {
            speedY = -10;
        }
        x += speedX;
        y += speedY;
        return true;
    }
    @Override
    public void draw(Graphics g) {
        g.setColor(Color.BLUE);
        g.fillRect(x, y, 50, 50);
    }
    @Override
    public Rectangle getBounds() {
        return new Rectangle(x, y, 50, 50);
    }
}
```


Bullet.java

```java
/**
 * Created by poonna on 4/24/15 AD.
 */
public class Bullet extends GameObject {
    private GameScene scene;
    private int x, y;
    private int speed = 20;

    public Bullet(GameScene scene, int x, int y) {
        this.scene = scene;
        this.x = x;
        this.y = y;
    }

    @Override
    public boolean update() {
        y -= speed;
        if (y < 0) {
            return false;
        } else {
            return true;
        }
    }

    @Override
    public void draw(Graphics g) {
        g.setColor(Color.WHITE);
        g.fillRect(x, y, 5, 10);
    }

    @Override
    public Rectangle getBounds() {
        return new Rectangle(x, y, 5, 10);
    }
}
```

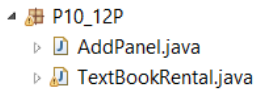
SpaceShip.java (1/2)

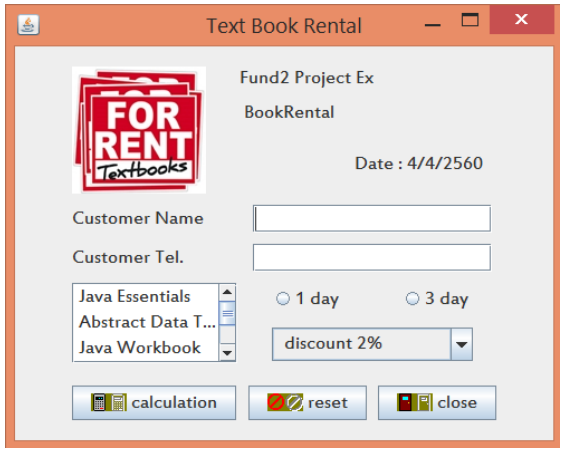
```java
public class SpaceShip extends GameObject {
    private GameScene scene;
    private Image image;
    private int x, y;
    private int speed = 10;

    public SpaceShip(GameScene scene, int x, int y) {
        this.scene = scene;

        ImageIcon loader = new ImageIcon("images/star.png");
        image = loader.getImage();

        this.x = x - image.getWidth(null) / 2;
        this.y = y - image.getHeight(null) / 2;
    }

    @Override
    public boolean update() {
        if (scene.getKeyState(GameScene.KEY_UP)) {
            y -= speed;
        }
        if (scene.getKeyState(GameScene.KEY_DOWN)) {
            y += speed;
        }
        if (scene.getKeyState(GameScene.KEY_LEFT)) {
            x -= speed;
        }
        if (scene.getKeyState(GameScene.KEY_RIGHT)) {
            x += speed;
        }

        return true;
    }
```


SpaceShip.java (2/2)

```java
    @Override
    public void draw(Graphics g) {
        g.drawImage(image, x, y, scene);
    }

    @Override
    public Rectangle getBounds() {
        return new Rectangle(x, y, image.getWidth(null), image.getHeight(null));
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}
```


Project Example





https://www.eventpop.me/e/1613-wtm-2017-by-gdgsriracha

09:30-09:45	อ.ปุณณะ ยศปัญญา คณบดีฝ่ายเทคโนโลยีสารสนเทศ กล่าวเปิดงาน
09:45-10:20	**Tell our story** **Speaker: Pichaya Penjan (P' Par)** • Scrum Master / Agile Coach Practitioner @Siam Chamnankit Co.,Ltd.
10:20-10:35	Break I
10:35-11:10	**Tell our story** **Speaker: Karun Warapongsittikul (P' Tong)** • CEO of Mojito Technology Co.,Ltd. • Google UI/UX Expert • Co-founder of UX Academy
11:10-11:45	**Tell our story** **Speaker: Nattanicha Phatharamalai (P' Kae)** • Agile Coach / R&D Manager • Girls Who Dev Founder

13:00-13:15	Brief Session
13:15-17:30	**Code Lab**
17:30-18:00	Find Winner
18:00-18:15	Close Session

กิจกรรมรอบบ่าย Code Lab รับ 56 ท่านเท่านั้น (8 ทีม / ทีมละ 7 ท่าน) ชิงรางวัลกว่า 25,000 บาท

END