Programming Graphics

Advanced Programming

ICOM 4015

Lecture 14

Reading: Java Concepts Chapter 5

Chapter Goals

- To be able to write simple applications
- To display graphical shapes such as lines and ellipses
- To use colors
- To display drawings consisting of many shapes
- To read input from a dialog box
- To develop test cases that validate the correctness of your programs

Frame Windows

- **The `JFrame` class**

```
JFrame frame = new JFrame();
frame.setSize(300, 400);
frame.setTitle("An Empty Frame");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

- **`import javax.swing.*;`**

A Frame Window

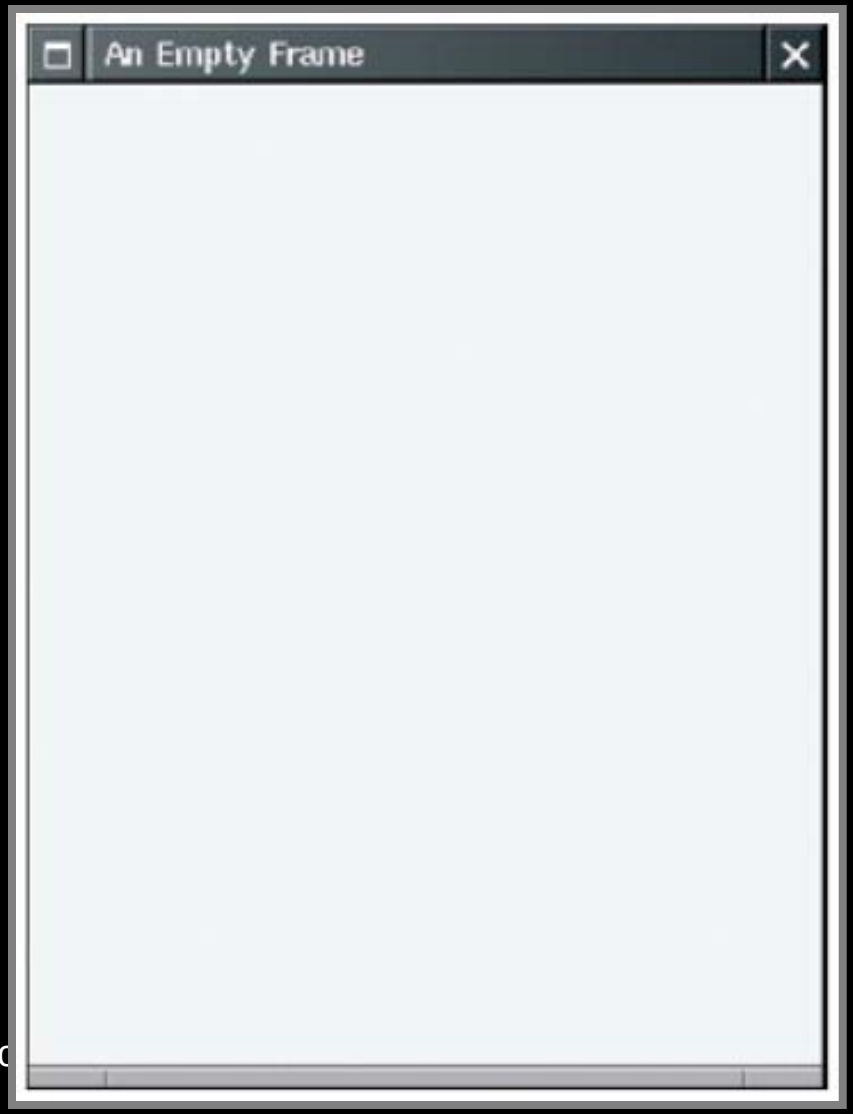


Figure 1:
A Frame Window

File EmptyFrameViewer.java

```
01: import javax.swing.*;
02:
03: public class EmptyFrameViewer
04: {
05:    public static void main(String[] args)
06:    {
07:       JFrame frame = new JFrame();
08:
09:       final int FRAME_WIDTH = 300;
10:       final int FRAME_HEIGHT = 400;
11:
12:       frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
13:       frame.setTitle("An Empty Frame");
14:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:
16:       frame.setVisible(true);
17:    }
18: }
```

Self Check

1. How do you display a square frame with a title bar that reads `"Hello, World!"`?
2. How can a program display two frames at once?

Answers

1. Modify the `EmptyFrameViewer` program as follows:

```
frame.setSize(300, 300);
frame.setTitle("Hello, World!");
```

2. Construct two `JFrame` objects, set each of their sizes, and call `setVisible(true)` on each of them

Drawing Shapes

- `paintComponent`: called whenever the component needs to be repainted:

```
public class RectangleComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      // Recover Graphics2D
      Graphics2D g2 = (Graphics2D) g;
      . . .
   }
}
```

Drawing Shapes

- `Graphics` class lets you manipulate the graphics state (such as current color)
- `Graphics2D` class has methods to draw shape objects
- Use a cast to recover the `Graphics2D` object from the `Graphics` parameter

```
Rectangle box = new Rectangle(5, 10, 20, 30);
g2.draw(box);
```

- `java.awt` package

Drawing Rectangles

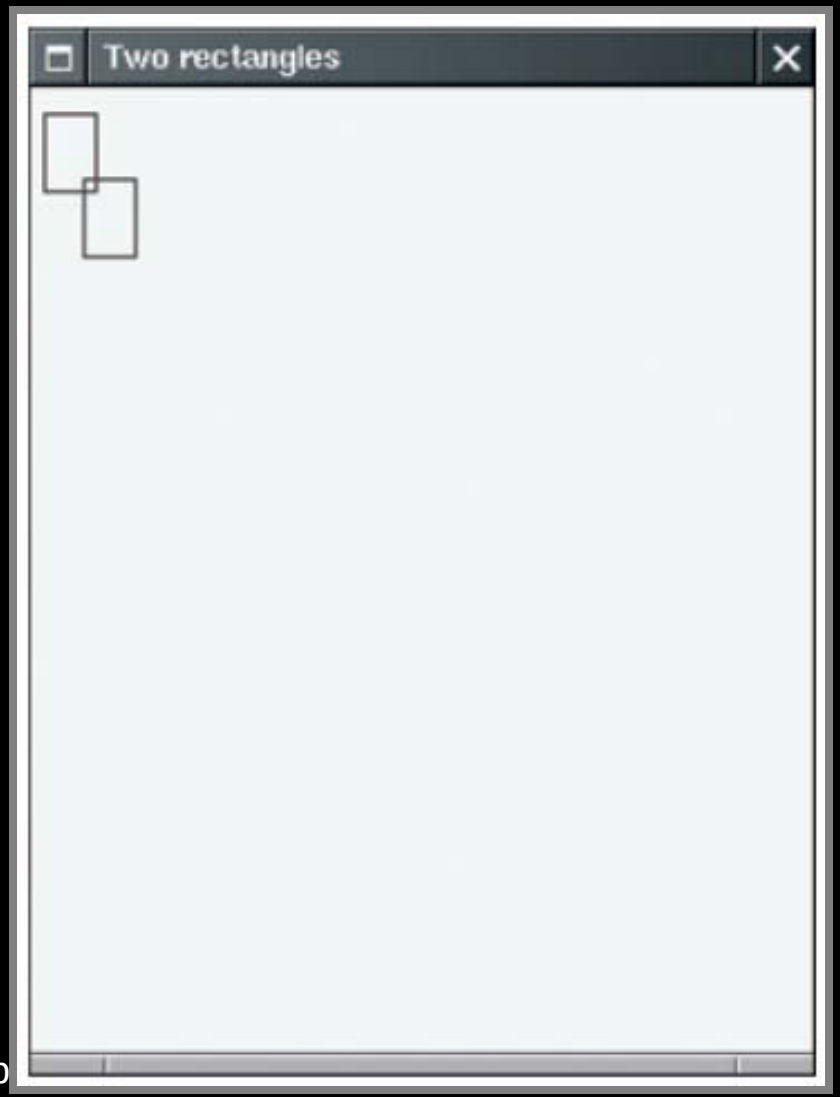


Figure 2:
Drawing Rectangles

Rectangle Drawing Program Classes

- `RectangleComponent`: its `paintComponent` method produces the drawing
- `RectangleViewer`: its `main` method constructs a frame and a `RectangleComponent`, adds the component to the frame, and makes the frame visible

Continued…

Rectangle Drawing Program Classes

1. Construct a frame
2. Construct an object of your component class:

```
RectangleComponent component = new RectangleComponent();
```

3. Add the component to the frame

```
frame.add(component);
```

However, if you use an older version of Java (before Version 5), you must make a slightly more complicated call:

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

4. Make the frame visible

File RectangleComponent.java

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.Rectangle;
04: import javax.swing.JPanel;
05: import javax.swing.JComponent;
06:
07: /**
08:    A component that draws two rectangles.
09: */
10: public class RectangleComponent extends JComponent
11: {
12:    public void paintComponent(Graphics g)
13:    {
14:       // Recover Graphics2D
15:       Graphics2D g2 = (Graphics2D) g;
16:
```


Continued…

File RectangleComponent.java

```
17:         // Construct a rectangle and draw it
18:         Rectangle box = new Rectangle(5, 10, 20, 30);
19:         g2.draw(box);
20:
21:         // Move rectangle 15 units to the right and 25 units
    // down
22:         box.translate(15, 25);
23:
24:         // Draw moved rectangle
25:         g2.draw(box);
26:      }
27: }
```

File RectangleViewer.java

```
01: import javax.swing.JFrame;
02:
03: public class RectangleViewer
04: {
05:    public static void main(String[] args)
06:    {
07:       JFrame frame = new JFrame();
08:
09:       final int FRAME_WIDTH = 300;
10:       final int FRAME_HEIGHT = 400;11:
12:       frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
13:       frame.setTitle("Two rectangles");
14:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:
```


Continued…

File RectangleViewer.java

```
16:         RectangleComponent component = new RectangleComponent();
17:         frame.add(component);
18:
19:         frame.setVisible(true);
20:     }
21: }
```

Self Check

1. How do you modify the program to draw two squares?
2. How do you modify the program to draw one rectangle and one square?
3. What happens if you call `g.draw(box)` instead of `g2.draw(box)`?

Answers

1.
```
Rectangle box = new Rectangle(5, 10, 20, 20);
```

2. **Replace the call to**

```
box.translate(15, 25)
```

with

```
box = new Rectangle(20, 35, 20, 20);
```

3. **The compiler complains that `g` doesn't have a `draw` method**

Applets

- **Applets are programs that run inside a web browser**
- **To implement an applet, use this code outline:**

```
public class MyApplet extends JApplet
{
   public void paint(Graphics g)
   {
      // Recover Graphics2D
      Graphics2D g2 = (Graphics2D) g;
      // Drawing instructions go here
      . . .
   }
}
```

Applets

- **This is almost the same outline as for a component, with two minor differences:**
  1. You extend `JApplet`, not `JComponent`
  2. You place the drawing code inside the `paint` method, not inside `paintComponent`
- **To run an applet, you need an HTML file with the `applet` tag**

Applets

- An HTML file can have multiple applets; add a separate `applet` tag for each applet
- You view applets with the `applet` viewer or a Java enabled browser

```
appletviewer RectangleApplet.html
```

File RectangleApplet.java

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.Rectangle;
04: import javax.swing.JApplet;
05:
06: /**
07:    An applet that draws two rectangles.
08: */
09: public class RectangleApplet extends JApplet
10: {
11:    public void paint(Graphics g)
12:    {
13:       // Prepare for extended graphics
14:       Graphics2D g2 = (Graphics2D) g;
```


Continued…

File RectangleApplet.java

```
15:
16:         // Construct a rectangle and draw it
17:         Rectangle box = new Rectangle(5, 10, 20, 30);
18:         g2.draw(box);
19:
20:         // Move rectangle 15 units to the right and 25 units
    // down
21:         box.translate(15, 25);
22:
23:         // Draw moved rectangle
24:         g2.draw(box);
25:     }
26: }
27:
```

File RectangleApplet.html

```
<applet code="RectangleApplet.class" width="300" height="400"
> </applet>
```

File RectangleAppletExplained.html

```
<html>
   <head>
      <title>Two rectangles</title>
   </head>
   <body>
      <p>Here is my <i>first applet</i>:</p>
      <applet code="RectangleApplet.class" width="300" height="400">
      </applet>
   </body>
</html>
```

Applets

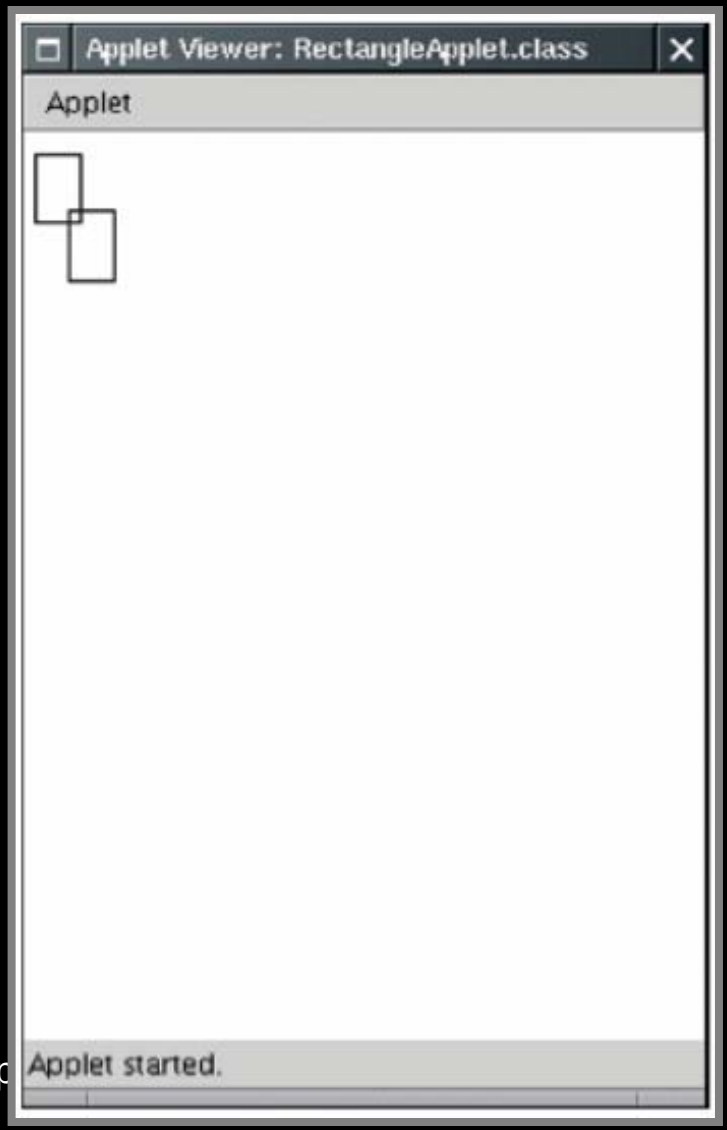


Figure 3:
An Applet in the Applet Viewer

Applets

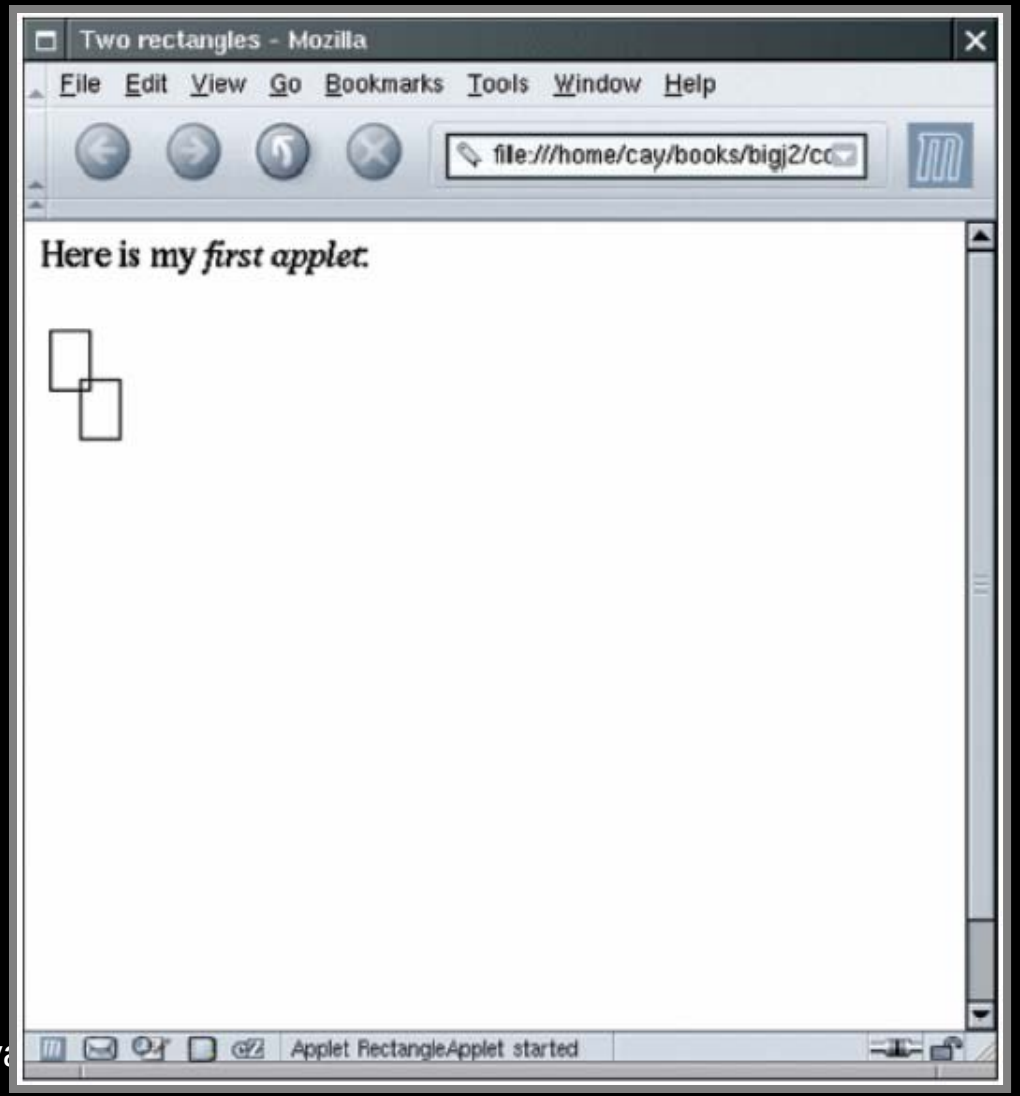


Figure 4:
An Applet in a Web Browser

Graphical Shapes

- `Rectangle`, `Ellipse2D.Double`, and `Line2D.Double` describe graphical shapes
- We won't use the `.Float` classes
- These classes are inner classes–doesn't matter to us except for the `import` statement:

```
import java.awt.geom.Ellipse2D; // no .Double
```

- Must construct *and draw* the shape

```
Ellipse2D.Double ellipse = new Ellipse2D.Double(x, y, width, height);
g2.draw(ellipse);
```

An Ellipse

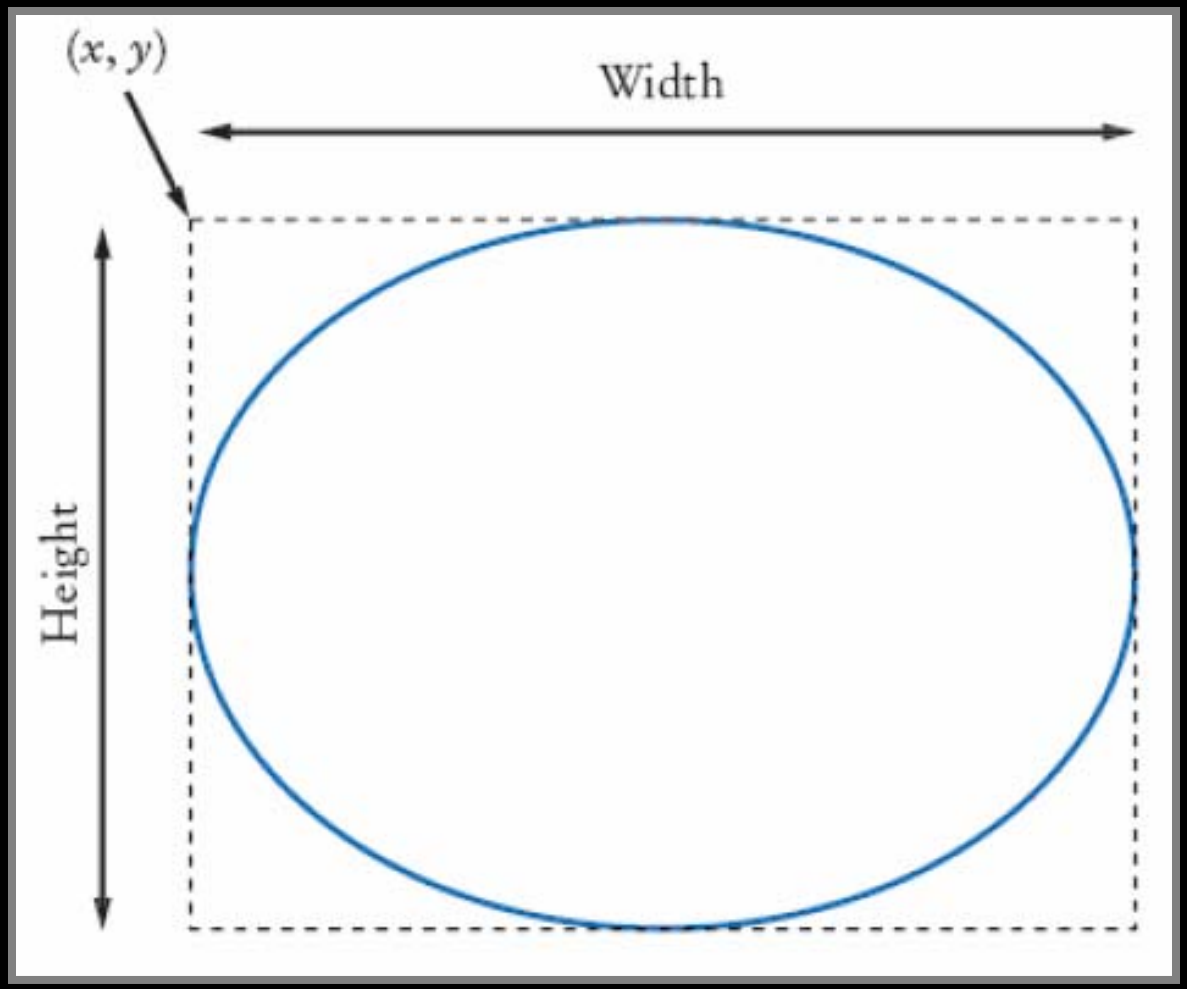


Figure 6:
An Ellipse and Its Bounding Box

Drawing Lines

- **To draw a line:**

```
Line2D.Double segment = new Line2D.Double(x1, y1, x2, y2);
```

or,

```
Point2D.Double from = new Point2D.Double(x1, y1);
Point2D.Double to = new Point2D.Double(x2, y2);
Line2D.Double segment = new Line2D.Double(from, to);
```

Drawing Strings

```
g2.drawString("Message", 50, 100);
```

Figure 7:
Basepoint and Baseline

Self Test

1. Give instructions to draw a circle with center (100, 100) and radius 25
2. Give instructions to draw a letter "V" by drawing two line segments
3. Give instructions to draw a string consisting of the letter "V"

Answers

1.
```
g2.draw(new Ellipse2D.Double(75, 75, 50, 50);
```

2.
```
Line2D.Double segment1 = new Line2D.Double(0, 0, 10, 30);
g2.draw(segment1);
Line2D.Double segment2 = new Line2D.Double(10, 30, 20, 0);
g2.draw(segment2);
```

3.
```
g2.drawString("V", 0, 30);
```

Colors

- Standard colors `Color.BLUE`, `Color.RED`, `Color.PINK` etc.
- Specify red, green, blue between `0.0F` and `1.0F` `Color magenta = new Color(1.0F, 0.0F, 1.0F); // F = float`
- Set color in graphics context

```
g2.setColor(magenta);
```

- Color is used when drawing and filling shapes

```
g2.fill(rectangle); // filled with current color
```

Self Check

1. What are the RGB color values of `Color.BLUE`?
2. How do you draw a yellow square on a red background?

Answers

1. `0.0F`, `0.0F`, and `0.1F`
2. First fill a big red square, then fill a small yellow square inside:

```
g2.setColor(Color.RED);
g2.fill(new Rectangle(0, 0, 200, 200));
g2.setColor(Color.YELLOW);
g2.fill(new Rectangle(50, 50, 100, 100));
```

Drawing Complex Shapes

- Good practice: Make a class for each graphical shape

```
class Car
{
   . . .
   public void draw(Graphics2D g2)
   {
      // Drawing instructions
      . . .
   }
}
```

- Plan complex shapes by making sketches on graph paper

Drawing Cars

- Draw two cars: one in top-left corner of window, and another in the bottom right
- Compute bottom right position, inside `paintComponent` method:

```
int x = getWidth() - 60;
int y = getHeight() - 30;
Car car2 = new Car(x, y);
```

- `getWidth` and `getHeight` are applied to object that executes `paintComponent`
- If window is resized `paintComponent` is called and car position recomputed

Continued…

Drawing Cars

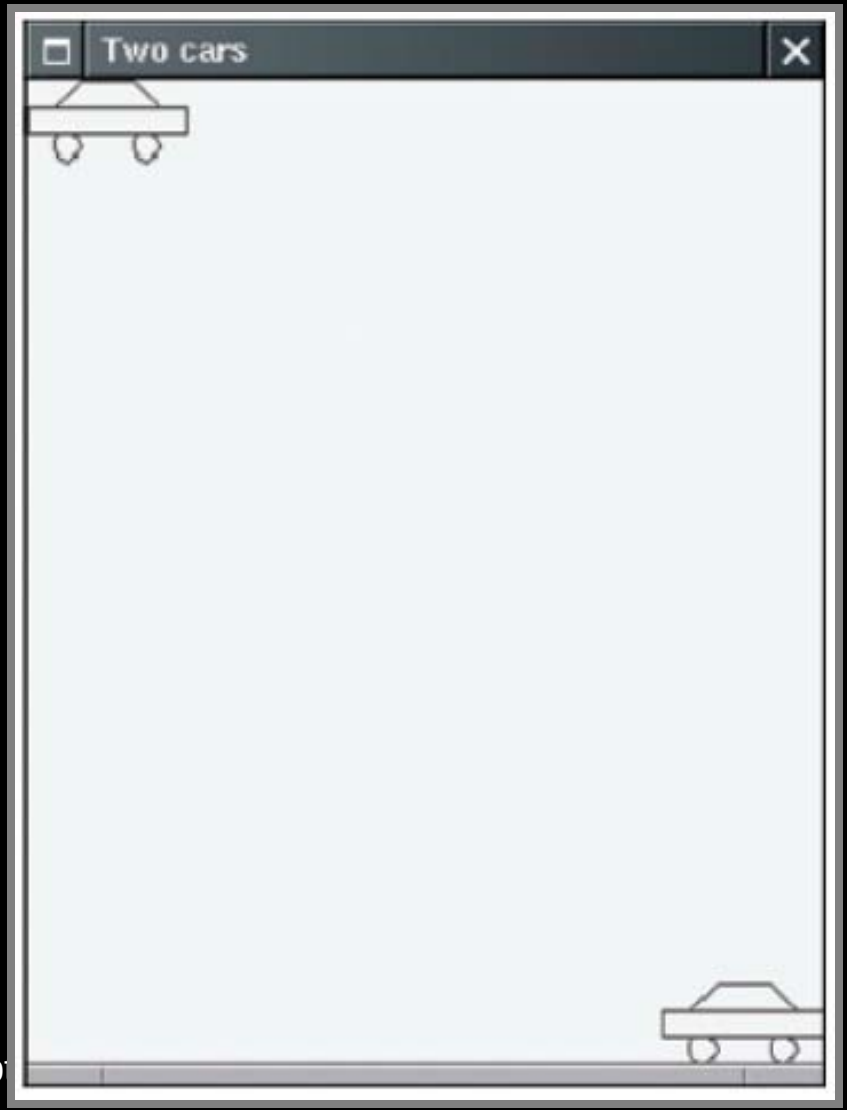


Figure 8:
The Car Component Draws Two Shapes

Plan Complex Shapes on Graph Paper

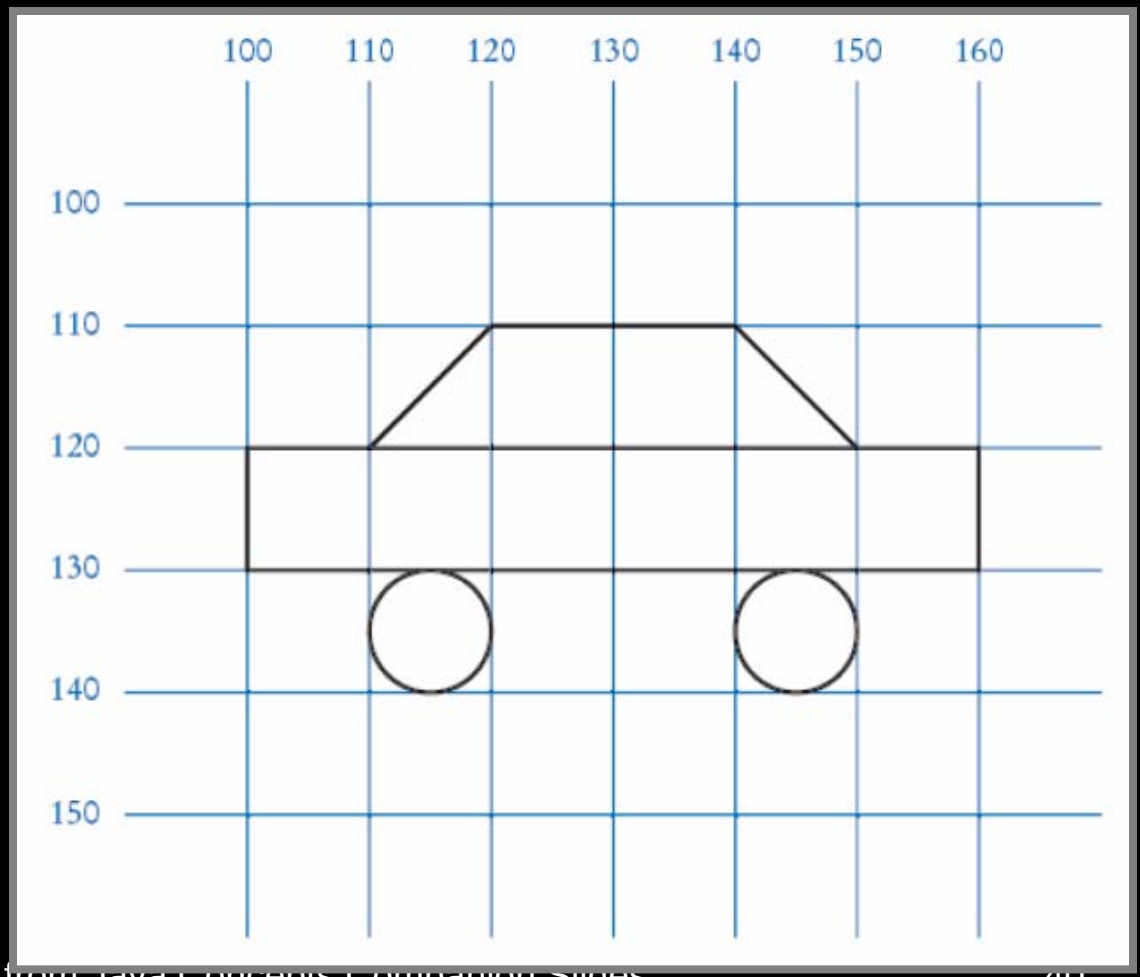


Figure 9:
Using Graph Paper to Find Shape Coordinates

File CarComponent.java

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import javax.swing.JComponent;
04:
05: /**
06:    This component draws two car shapes.
07: */
08: public class CarComponent extends JComponent
09: {
10:    public void paintComponent(Graphics g)
11:    {
12:       Graphics2D g2 = (Graphics2D) g;
13:
14:       Car car1 = new Car(0, 0);
15:
```


Continued…

File CarComponent.java

```
16:         int x = getWidth() - Car.WIDTH;
17:         int y = getHeight() - Car.HEIGHT;
18:
19:         Car car2 = new Car(x, y);
20:
21:         car1.draw(g2);
22:         car2.draw(g2);
23:     }
24: }
```

File Car.java

```
01: import java.awt.Graphics2D;
02: import java.awt.Rectangle;
03: import java.awt.geom.Ellipse2D;
04: import java.awt.geom.Line2D;
05: import java.awt.geom.Point2D;
06:
07: /**
08:    A car shape that can be positioned anywhere on the screen.
09: */
10: public class Car
11: {
12:    /**
13:       Constructs a car with a given top left corner
14:       @param x the x coordinate of the top left corner
15:       @param y the y coordinate of the top left corner
16:    */
```


Continued…

File Car.java

```
17:    public Car(int x, int y)
18:    {
19:       xLeft = x;
20:       yTop = y;
21:    }
22:
23:    /**
24:       Draws the car.
25:       @param g2 the graphics context
26:    */
27:    public void draw(Graphics2D g2)
28:    {
29:       Rectangle body
30:             = new Rectangle(xLeft, yTop + 10, 60, 10);
31:       Ellipse2D.Double frontTire
32:             = new Ellipse2D.Double(xLeft + 10, yTop
                     + 20, 10, 10);
33:       Ellipse2D.Double rearTire
```

Continued…

File Car.java

```
34:             = new Ellipse2D.Double(xLeft + 40, yTop
                      + 20, 10, 10);
35:
36:       // The bottom of the front windshield
37:       Point2D.Double r1
38:             = new Point2D.Double(xLeft + 10, yTop + 10);
39:       // The front of the roof
40:       Point2D.Double r2
41:             = new Point2D.Double(xLeft + 20, yTop);
42:       // The rear of the roof
43:       Point2D.Double r3
44:             = new Point2D.Double(xLeft + 40, yTop);
45:       // The bottom of the rear windshield
46:       Point2D.Double r4
47:             = new Point2D.Double(xLeft + 50, yTop + 10);
48:
49:       Line2D.Double frontWindshield
50:             = new Line2D.Double(r1, r2);
```

Continued…

File Car.java

```
51:         Line2D.Double roofTop
52:               = new Line2D.Double(r2, r3);
53:         Line2D.Double rearWindshield
54:               = new Line2D.Double(r3, r4);
55:
56:         g2.draw(body);
57:         g2.draw(frontTire);
58:         g2.draw(rearTire);
59:         g2.draw(frontWindshield);
60:         g2.draw(roofTop);
61:         g2.draw(rearWindshield);
62:     }
63:
64:     public static int WIDTH = 60;
65:     public static int HEIGHT = 30;
66:     private int xLeft;
67:     private int yTop;
68: }
```

File CarViewer.java

```
01: import javax.swing.JFrame;
02:
03: public class CarViewer
04: {
05:    public static void main(String[] args)
06:    {
07:       JFrame frame = new JFrame();
08:
09:       final int FRAME_WIDTH = 300;
10:       final int FRAME_HEIGHT = 400;
11:
12:       frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
13:       frame.setTitle("Two cars");
14:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```


Continued…

File CarViewer.java

```
15:
16:         CarComponent component = new CarComponent();
17:         frame.add(component);
18:
19:         frame.setVisible(true);
20:     }
21: }
```

Self Check

1. Which class needs to be modified to have the two cars positioned next to each other?
2. Which class needs to be modified to have the car tires painted in black, and what modification do you need to make?
3. How do you make the cars twice as big?

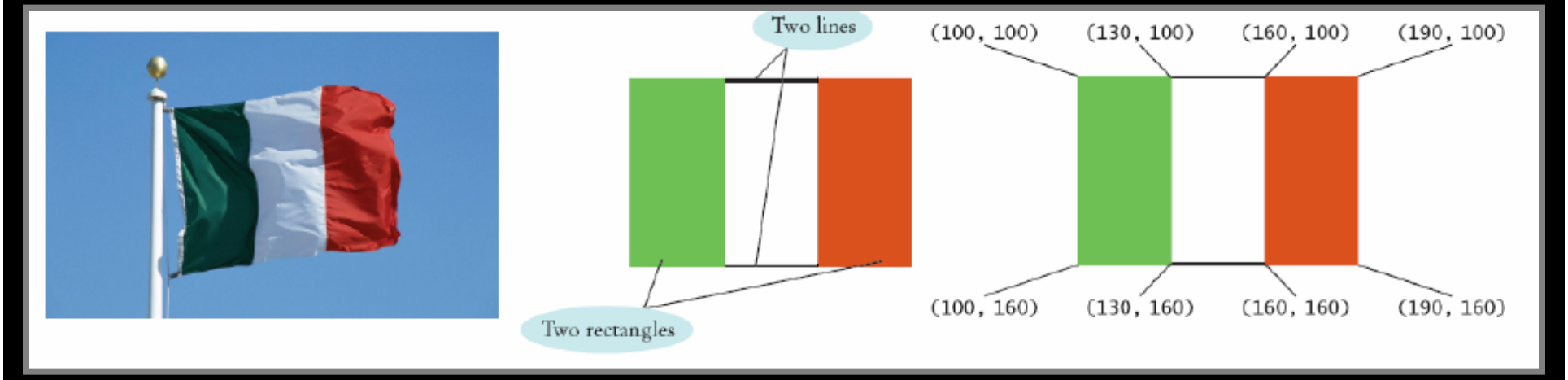
Answers

1. `CarComponent`
2. In the `draw` method of the Car class, call

```
g2.fill(frontTire);
g2.fill(rearTire);
```

3. Double all measurements in the draw method of the `Car` class

Drawing Graphical Shapes




```
Rectangle leftRectangle = new Rectangle(100, 100, 30, 60);
Rectangle rightRectangle = new Rectangle(160, 100, 30, 60);
Line2D.Double topLine
   = new Line2D.Double(130, 100, 160, 100);
Line2D.Double bottomLine
   = new Line2D.Double(130, 160, 160, 160);
```

Computer Graphics

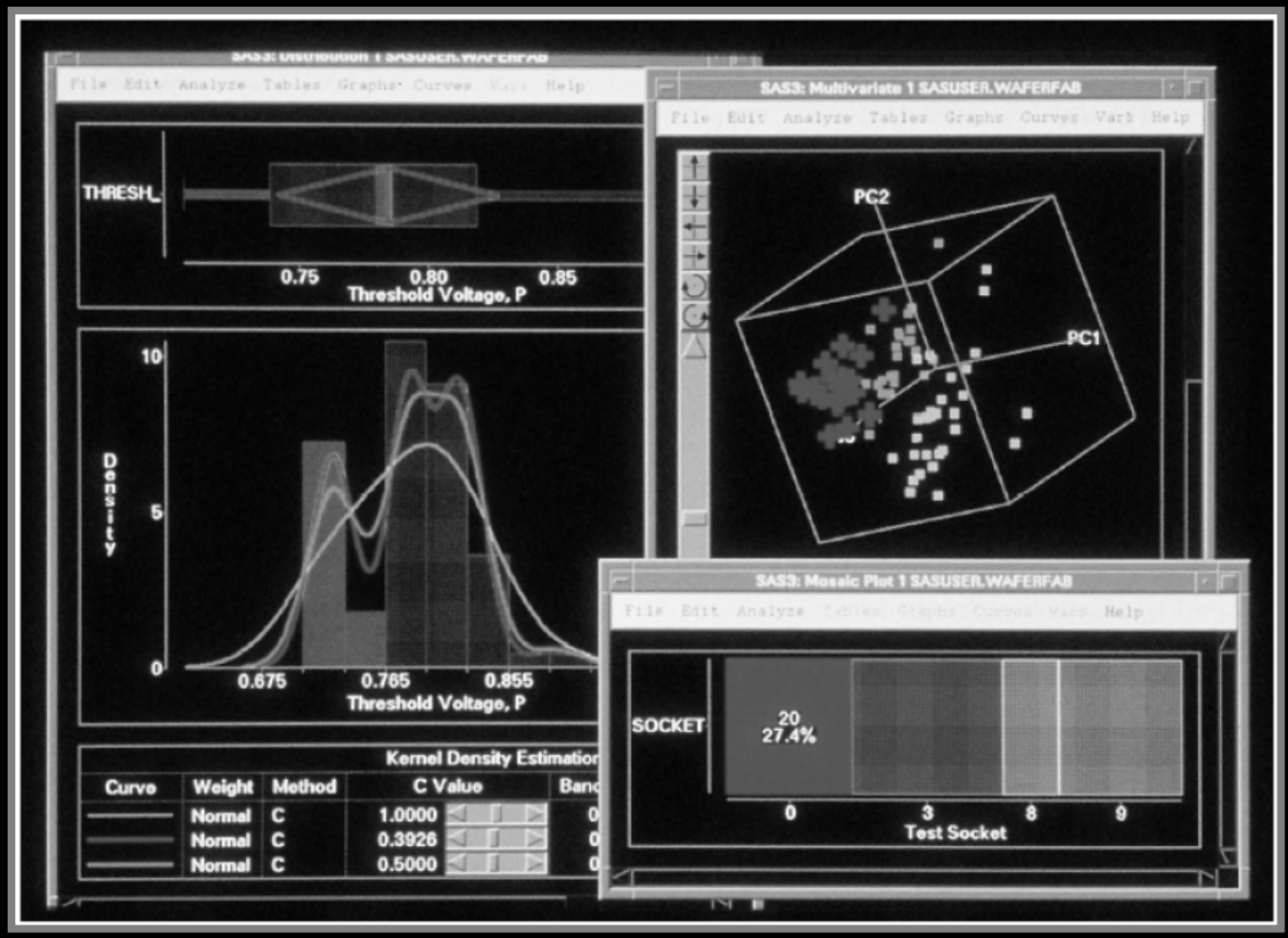


Figure 10: Diagrams

Computer Graphics

Figure 11: Scene

Computer Graphics

Figure 12:
Manipulated Image

Reading `Text` Input

- A graphical application can obtain input by displaying a `JOptionPane`
- The `showInputDialog` method displays a prompt and waits for user input
- The `showInputDialog` method returns the string that the user typed

```
String input = JOptionPane.showInputDialog("Enter x");
double x = Double.parseDouble(input);
```


Continued…

Reading `Text` Input

Figure 13:
An Input Dialog Box

File ColorViewer.java

```
01: import java.awt.Color;
02: import javax.swing.JFrame;
03: import javax.swing.JOptionPane;
04:
05: public class ColorViewer
06: {
07:    public static void main(String[] args)
08:    {
09:       JFrame frame = new JFrame();
10:
11:       final int FRAME_WIDTH = 300;
12:       final int FRAME_HEIGHT = 400;
13:
14:       frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
15:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16:
17:       String input;
18:
```

Continued…

File ColorViewer.java

```
19:         // Ask the user for red, green, blue values
20:
21:         input = JOptionPane.showInputDialog("red:");
22:         double red = Double.parseDouble(input);
23:
24:         input = JOptionPane.showInputDialog("green:");
25:         double green = Double.parseDouble(input);
26:
27:         input = JOptionPane.showInputDialog("blue:");
28:         double blue = Double.parseDouble(input);
29:
30:         Color fillColor = new Color(
31:               (float) red, (float) green, (float) blue);
32:         ColoredSquareComponent component
33:               = new ColoredSquareComponent(fillColor);
34:         frame.add(component);
35:
36:         frame.setVisible(true);
37:    }
38: }
```

File ColoredSquareComponent.java

```
01: import java.awt.Color;
02: import java.awt.Graphics;
03: import java.awt.Graphics2D;
04: import java.awt.Rectangle;
05: import javax.swing.JComponent;
06:
07: /**
08:    A component that shows a colored square.
09: */
10: public class ColoredSquareComponent extends JComponent
11: {
12:    /**
13:       Constructs a component that shows a colored square.
14:       @param aColor the fill color for the square
15:    */
16:    public ColoredSquareComponent(Color aColor)
```

Continued…

File ColoredSquareComponent.java

```
17:    {
18:       fillColor = aColor;
19:    }
20:
21:    public void paintComponent(Graphics g)
22:    {
23:       Graphics2D g2 = (Graphics2D) g;
24:
25:       // Select color into graphics context
26:
27:       g2.setColor(fillColor);
28:
29:       // Construct and fill a square whose center is
30:       // the center of the window
31:
```

Continued…

File ColoredSquareComponent.java

```
32:         final int SQUARE_LENGTH = 100;
33:
34:         Rectangle square = new Rectangle(
35:               (getWidth() - SQUARE_LENGTH) / 2,
36:               (getHeight() - SQUARE_LENGTH) / 2,
37:               SQUARE_LENGTH,
38:               SQUARE_LENGTH);
39:
40:         g2.fill(square);
41:     }
42:
43:     private Color fillColor;
44: }
```

Output

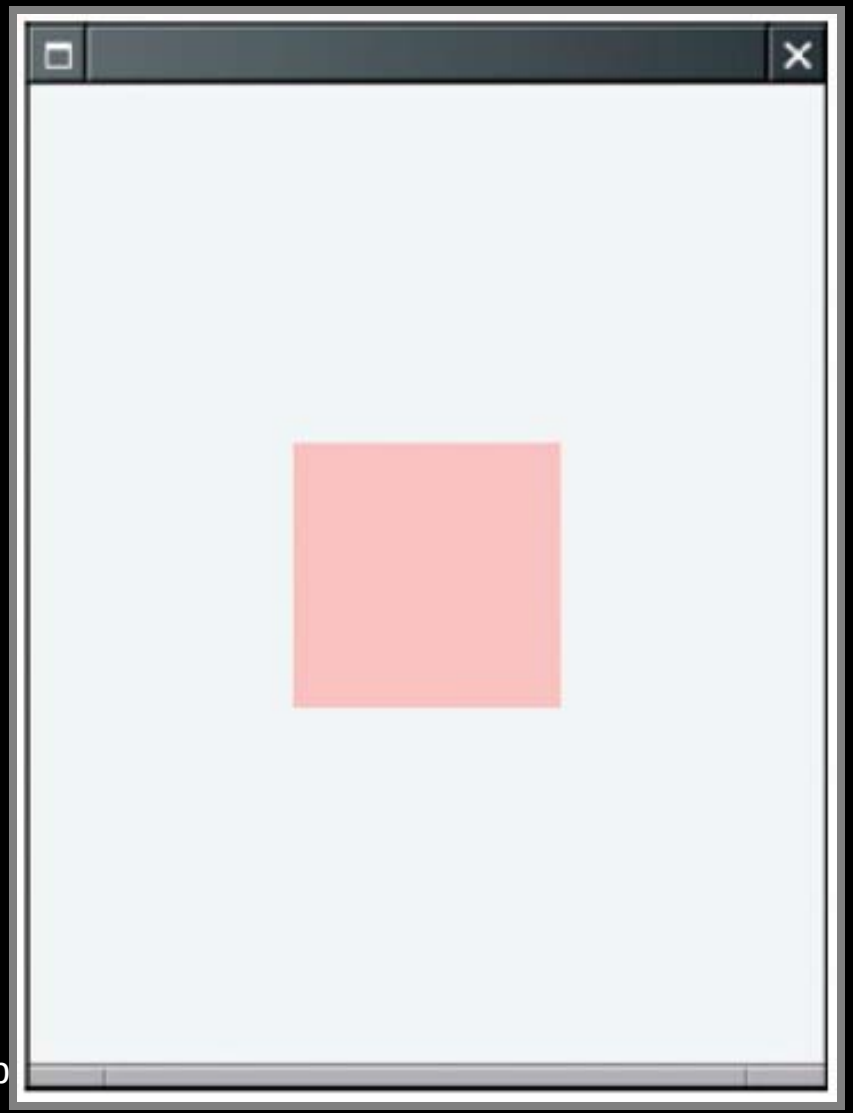


Figure 14:
A Square Filled with a User-Specified Color

Self Check

1. Why does this program produce three separate dialog boxes instead of inviting the user to type all three values in a single dialog box?
2. Why does this program place the `showInputDialog` call into the `main` method of the `ColorViewer` class and not into the `paintComponent` method of the `ColorComponent` class?

Answers

1. If the user entered a string, such as `"1.0 0.7 0.7"`, you would need to break it up into three separate strings. That can be done, but it is more tedious to program than three calls to `showInputDialog`.
2. You don't want the dialog boxes to appear every time the component is repainted.

Comparing Visual and Numerical Information

- Compute intersection between circle and vertical line
- Circle has radius $r = 100$ and center $(a, b) = (100, 100)$
- Line has constant x value

Continued…

Comparing Visual and Numerical Information

- **Calculate intersection points using mathematics: Equation of a circle with radius *r* and center point *(a, b)* is**

$$(x-a)^2+(y-b)^2=r^2$$

If you know *x*, then you can solve for *y:*

$$(y-b)^2=r^2-(x-a)^2$$

$$y-b=\pm\sqrt{r^2-(x-a)^2}$$

$$y=b\pm\sqrt{r^2-(x-a)^2}$$

Comparing Visual and Numerical Information

- That is easy to compute in Java:

```
double root = Math.sqrt(r * r - (x - a) * (x - a));
double y1 = b + root;
double y2 = b - root;
```

- Plot circle, line, computed intersection points
- Visual and numerical results should be the same

Intersection of a Line and a Circle

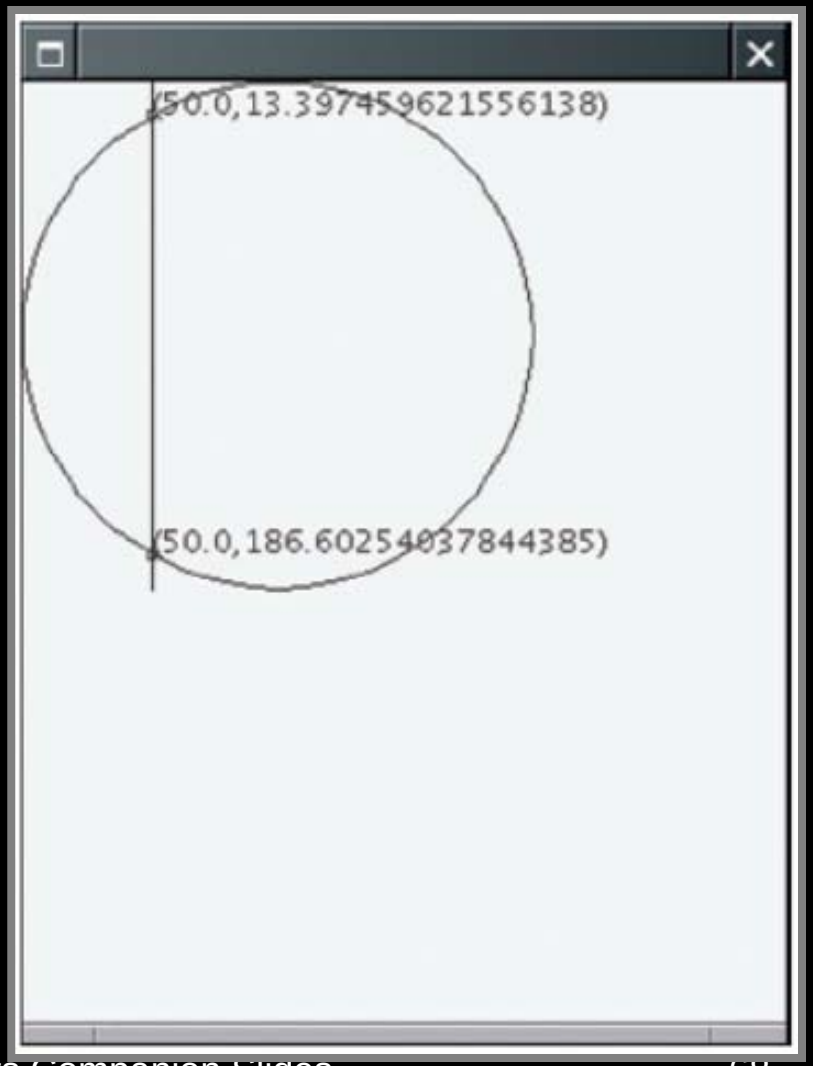


Figure 15
Intersection of a Line and a Circle

File IntersectionComponent.java

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.geom.Ellipse2D;
04: import java.awt.geom.Line2D;
05: import javax.swing.JComponent;
06:
07: /**
08:    A component that computes and draws the intersection points
09:    of a circle and a line.
10: */
11: public class IntersectionComponent extends JComponent
12: {
13:    /**
14:       Constructs the component from a given x-value for the line
15:       @param anX  the x-value for the line (between 0 and 200)
16:    */
```


Continued…

File IntersectionComponent.java

```
17:    public IntersectionComponent(double anX)
18:    {
19:       x = anX;
20:    }
21:
22:    public void paintComponent(Graphics g)
23:    {
24:       Graphics2D g2 = (Graphics2D) g;
25:
26:       // Draw the circle
27:
28:       final double RADIUS = 100;
29:
30:       Ellipse2D.Double circle
31:             = new Ellipse2D.Double(0, 0, 2 * RADIUS, 2 * RADIUS);
32:       g2.draw(circle);
33:
34:       // Draw the vertical line
35:
```

Continued...

File IntersectionComponent.java

```
36:         Line2D.Double line
37:               = new Line2D.Double(x, 0, x, 2 * RADIUS);
38:         g2.draw(line);
39:
40:         // Compute the intersection points
41:
42:         double a = RADIUS;
43:         double b = RADIUS;
44:
45:         double root = Math.sqrt(RADIUS * RADIUS - (x -  * (x - a));
46:         double y1 = b + root;
47:         double y2 = b - root;
48:
49:         // Draw the intersection points
50:
51:         LabeledPoint p1 = new LabeledPoint(x, y1);
52:         LabeledPoint p2 = new LabeledPoint(x, y2);
```

Continued…

File IntersectionComponent.java

```
53:
54:         p1.draw(g2);
55:         p2.draw(g2);
56:     }
57:
58:     private double x;
59: }
```

File IntersectionViewer.java

```
01: import javax.swing.JFrame;
02: import javax.swing.JOptionPane;
03:
04: public class IntersectionViewer
05: {
06:    public static void main(String[] args)
07:    {
08:       JFrame frame = new JFrame();
09:
10:       final int FRAME_WIDTH = 300;
11:       final int FRAME_HEIGHT = 400;
12:
13:       frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
14:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:
```


Continued…

File IntersectionViewer.java

```
16:         String input = JOptionPane.showInputDialog("Enter x");
17:         double x = Double.parseDouble(input);
18:         IntersectionComponent component
19:               = new IntersectionComponent(x);
20:         frame.add(component);
21:
22:         frame.setVisible(true);
23:     }
24: }
```

File LabeledPoint.java

```
01: import java.awt.Graphics2D;
02: import java.awt.geom.Ellipse2D;
03:
04: /**
05:    A point with a label showing the point's coordinates.
06: */
07: public class LabeledPoint
08: {
09:    /**
10:       Construct a labeled point.
11:       @param anX the x coordinate
12:       @param aY the y coordinate
13:    */
14:    public LabeledPoint(double anX, double aY)
15:    {
16:       x = anX;17:          y = aY;
18:    }
```


Continued...

File LabeledPoint.java

```
19:
20:    /**
21:       Draws the point as a small circle with a coordinate label.
22:       @param g2 the graphics context23:    */
24:    public void draw(Graphics2D g2)
25:    {
26:       // Draw a small circle centered around (x, y)
27:
28:       Ellipse2D.Double circle = new Ellipse2D.Double(
29:             x - SMALL_CIRCLE_RADIUS,
30:             y - SMALL_CIRCLE_RADIUS,
31:             2 * SMALL_CIRCLE_RADIUS,
32:             2 * SMALL_CIRCLE_RADIUS);
33:
34:       g2.draw(circle);
35:
36:       // Draw the label
```


Continued...

File LabeledPoint.java

```
37:
38:       String label = "(" + x + "," + y + ")";
39:
40:       g2.drawString(label, (float) x, (float) y);
41:    }
42:
43:    private static final double SMALL_CIRCLE_RADIUS = 2;
44:
45:    private double x;
46:    private double y;
47: }
```

Self Check

1. Suppose you make a mistake in the math, say, by using a + sign instead of a - sign in the formula for root. How can you tell that the program does not run correctly?
2. Which intersection points does the program draw when you provide an input of 0?

Answers

1. The intersection points will be drawn at a location that is different from the true intersection of the line and the circle
2. The point (0, 100) is drawn twice