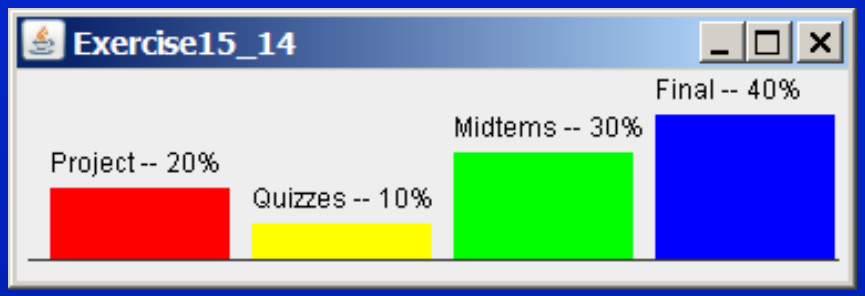
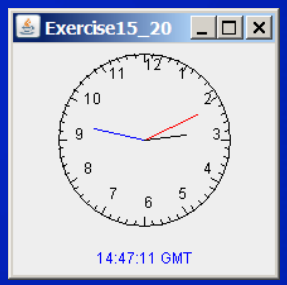
Chapter 15 Graphics

Motivations

If you want to draw shapes such as a bar chart, a clock, or a stop sign, how do you do it?

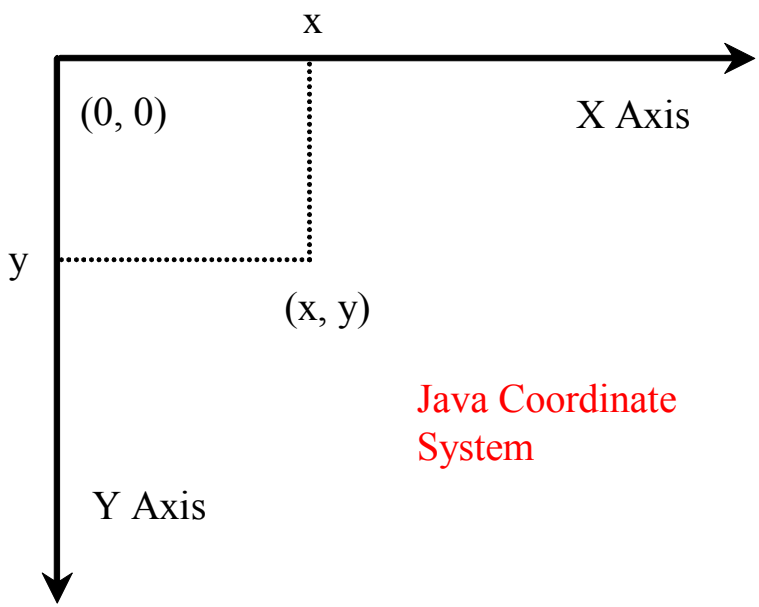




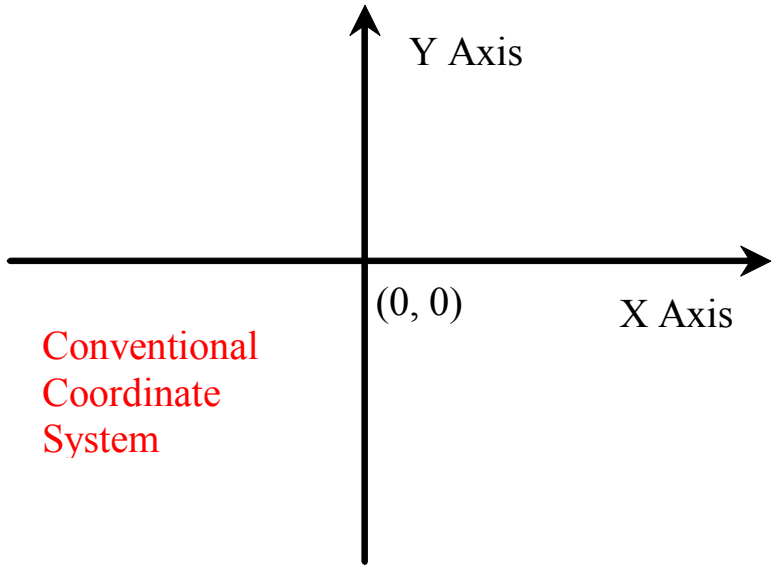
Objectives

- To describe Java coordinate systems in a GUI component (§15.2).
- To draw things using the methods in the Graphics class (§15.3).
- To override the paintComponent method to draw things on a GUI component (§15.3).
- To use a panel as a canvas to draw things (§15.3).
- To draw strings, lines, rectangles, ovals, arcs, and polygons (§§15.4, 15.6-15.7).
- To obtain font properties using FontMetrics and know how to center a message (§15.8).
- To display an image in a GUI component (§15.11).
- To develop reusable GUI components FigurePanel, MessagePanel, StillClock, and ImageViewer (§§15.5, 15.9, 15.10, 15.12).

Java Coordinate System

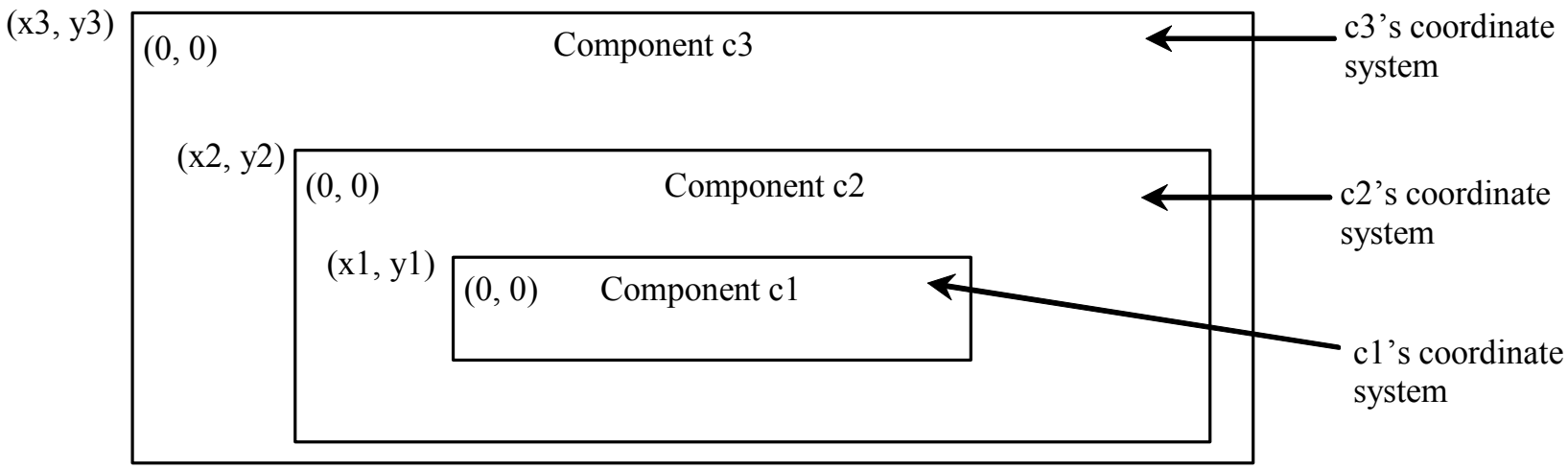


Java Coordinate System



Conventional Coordinate System

Each GUI Component Has its Own Coordinate System



The Graphics Class

You can draw strings, lines, rectangles, ovals, arcs, polygons, and polylines, using the methods in the Graphics class.

java.awt.Graphics	
+setColor(color: Color): void	Sets a new color for subsequent drawings.
+setFont(font: Font): void	Sets a new font for subsequent drwings.
+drawString(s: String, x: int, y: int): void	Draws a string starting at point (x, y).
+drawLine(x1: int, y1: int, x2: int, y2: int): void	Draws a line from (x1, y1) to (x2, y2).
+drawRect(x: int, y: int, w: int, h: int): void	Draws a rectangle with specified upper-left corner point at (x, y) and width w and height h.
+fillRect(x: int, y: int, w: int, h: int): void	Draws a filled rectangle with specified upper-left corner point at (x, y) and width w and height h.
+drawRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int): void	Draws a round-cornered rectangle with specified arc width aw and arc height ah.
+fillRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int): void	Draws a filled round-cornered rectangle with specified arc width aw and arc height ah.
+draw3DRect(x: int, y: int, w: int, h: int, raised: boolean): void	Draws a 3-D rectangle raised above the surface or sunk into the surface.
+fill3DRect(x: int, y: int, w: int, h: int, raised: boolean): void	Draws a filled 3-D rectangle raised above the surface or sunk into the surface.
+drawOval(x: int, y: int, w: int, h: int): void	Draws an oval bounded by the rectangle specified by the parameters x, y, w, and h.
+fillOval(x: int, y: int, w: int, h: int): void	Draws a filled oval bounded by the rectangle specified by the parameters x, y, w, and h.
+drawArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void	Draws an arc conceived as part of an oval bounded by the rectangle specified by the parameters x, y, w, and h.
+fillArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void	Draws a filled arc conceived as part of an oval bounded by the rectangle specified by the parameters x, y, w, and h.
+drawPolygon(xPoints: int[], yPoints: int[], nPoints: int): void	Draws a closed polygon defined by arrays of x and y coordinates. Each pair of (x[i], y[i]) coordinates is a point.
+fillPolygon(xPoints: int[], yPoints: int[], nPoints: int): void	Draws a filled polygon defined by arrays of x and y coordinates. Each pair of (x[i], y[i]) coordinates is a point.
+drawPolygon(g: Polygon): void	Draws a closed polygon defined by a Polygon object.
+fillPolygon(g: Polygon): void	Draws a filled polygon defined by a Polygon object.
+drawPolyline(xPoints: int[], yPoints: int[], nPoints: int): void	Draws a polyline defined by arrays of x and y coordinates. Each pair of (x[i], y[i]) coordinates is a point.

paintComponent Example

In order to draw things on a component, you need to define a class that extends JPanel and overrides its paintComponent method to specify what to draw. The first program in this chapter can be rewritten using paintComponent.

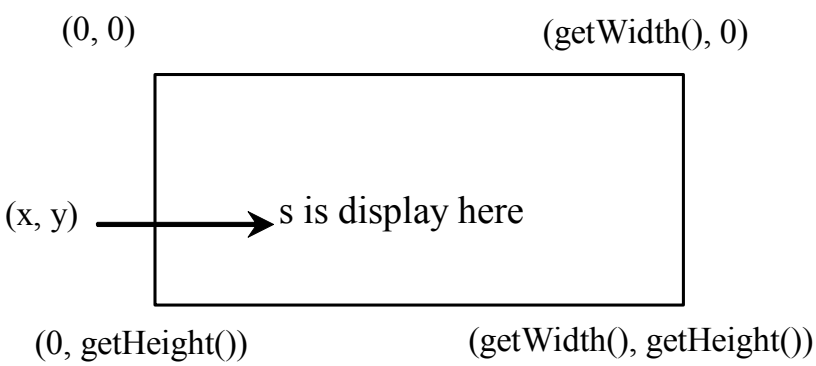
TestPaintComponent

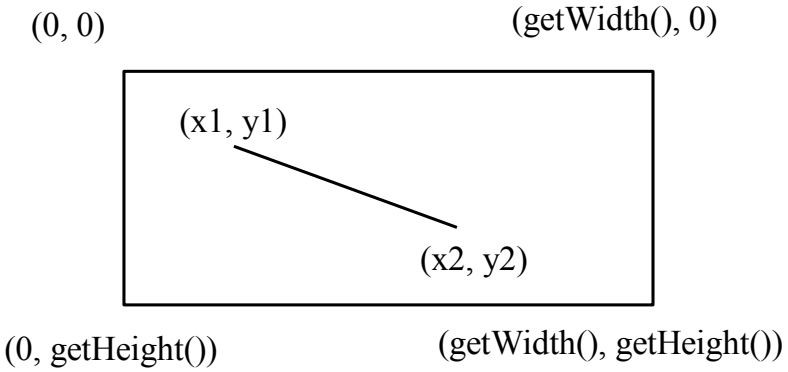
Run

Drawing Geometric Figures

- Drawing Strings
- Drawing Lines
- Drawing Rectangles
- Drawing Ovals
- Drawing Arcs
- Drawing Polygons

Drawing Strings





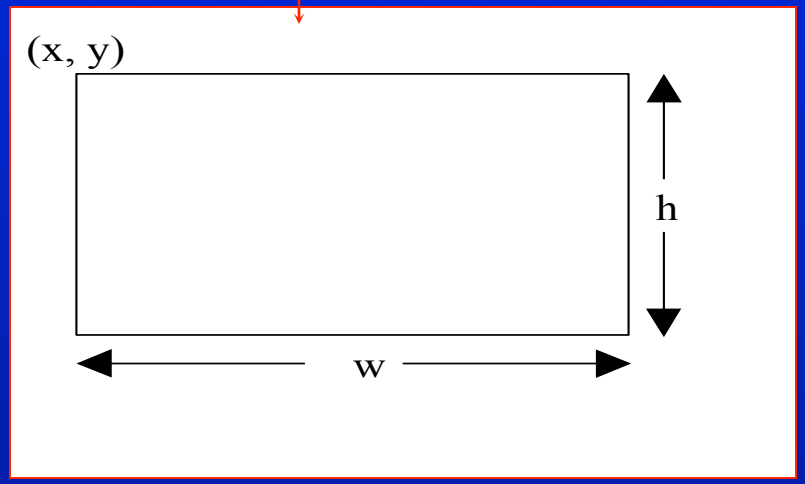
drawString(String s, int x, int y);

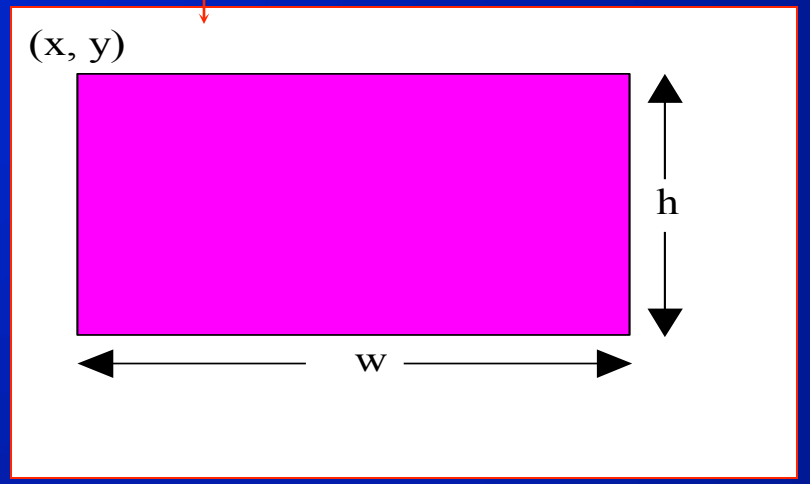
drawLine(int x1, int y1, int x2, int y2);

Drawing Rectangles

drawRect(int x, int y, int w, int h);

fillRect(int x, int y, int w, int h);

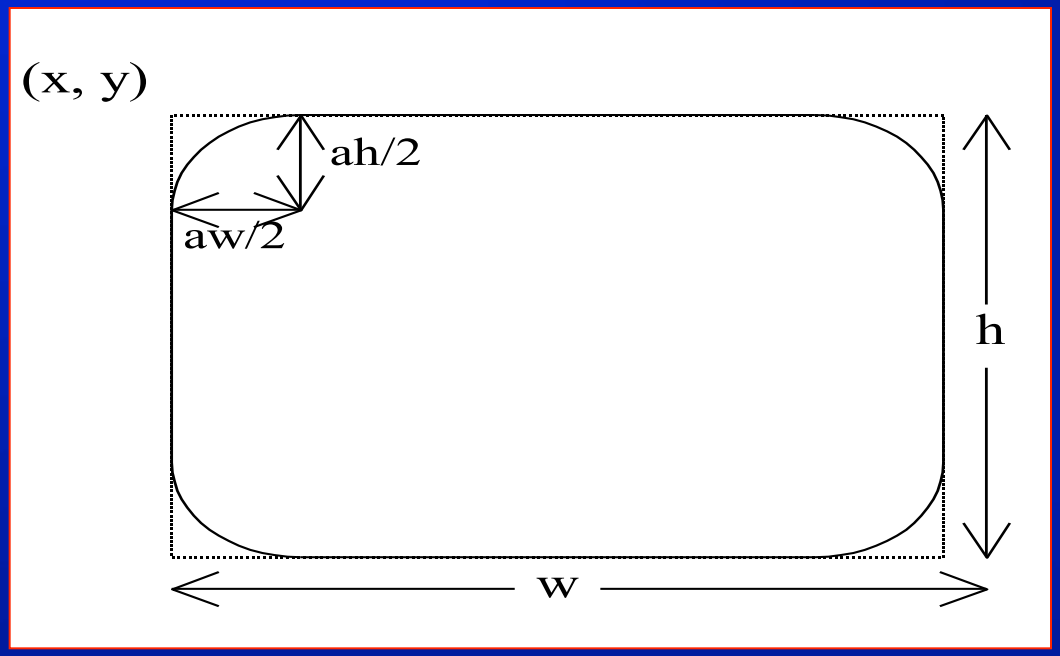




Drawing Rounded Rectangles

drawRoundRect(int x, int y, int w, int h, int aw, int ah);

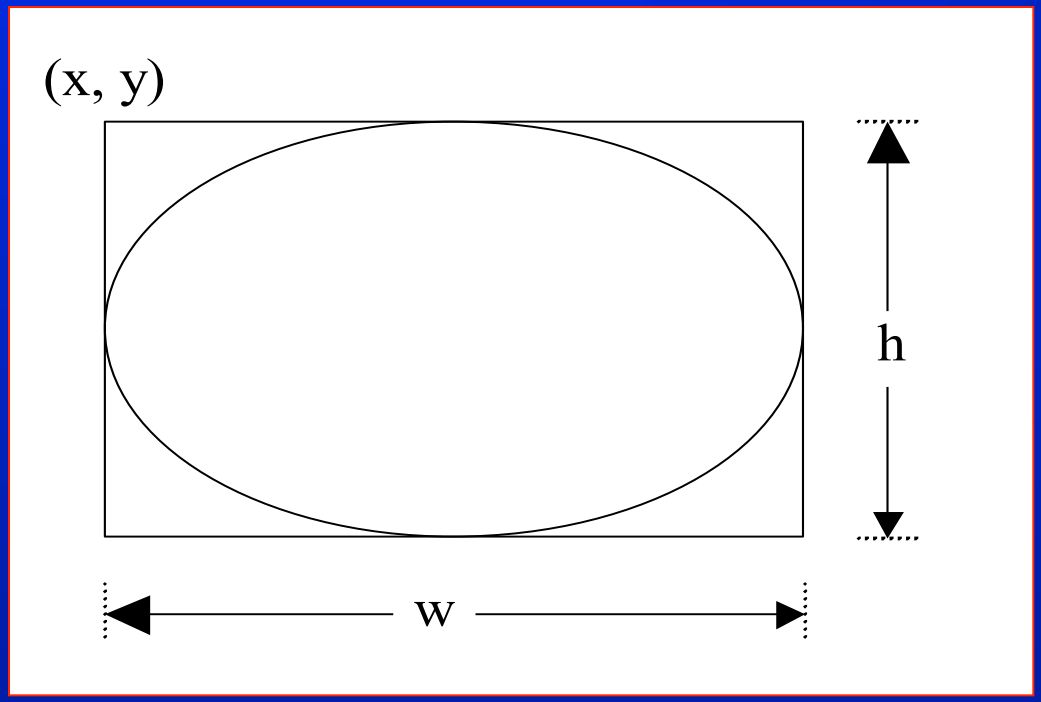
fillRoundRect(int x, int y, int w, int h, int aw, int ah);



Drawing Ovals

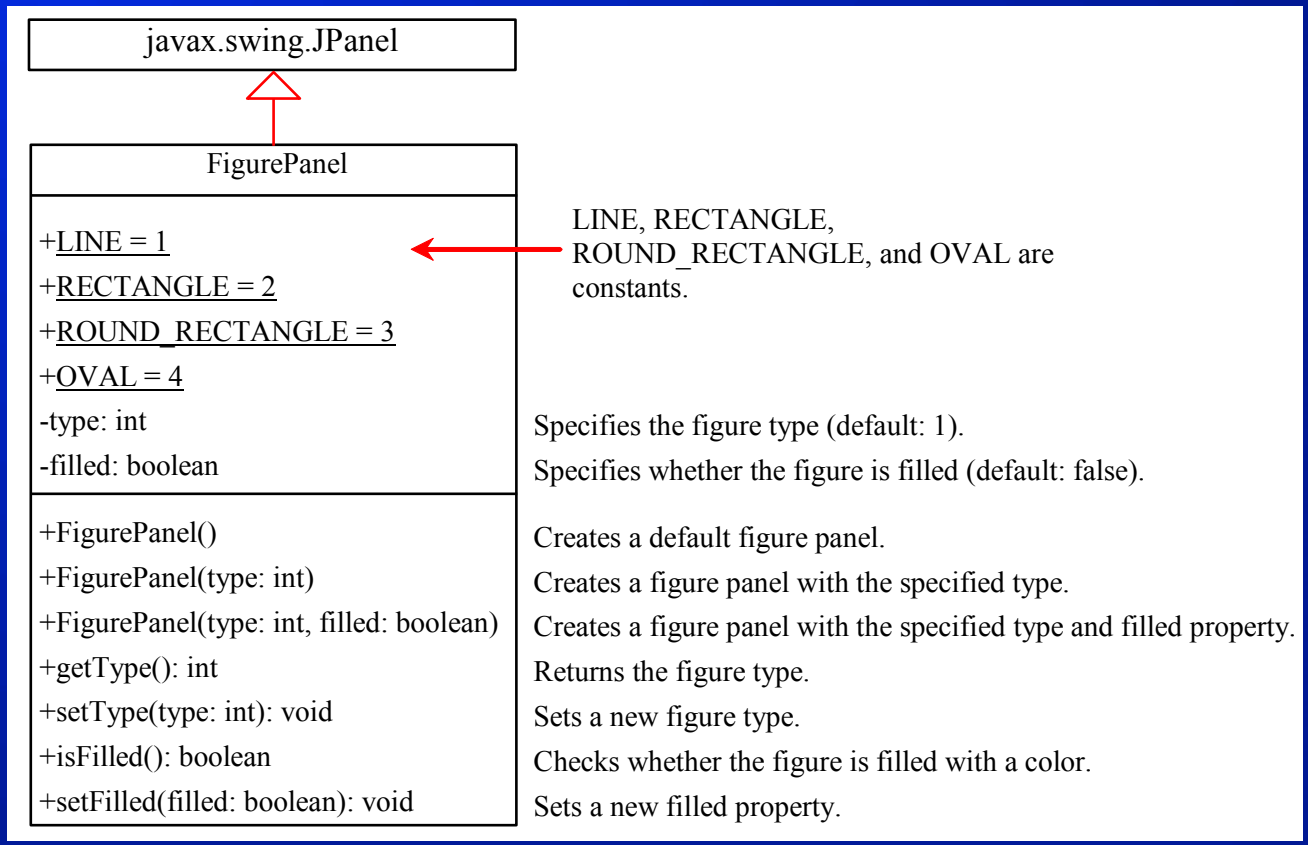
drawOval(int x, int y, int w, int h);

fillOval(int x, int y, int w, int h);



Case Study: The FigurePanel Class

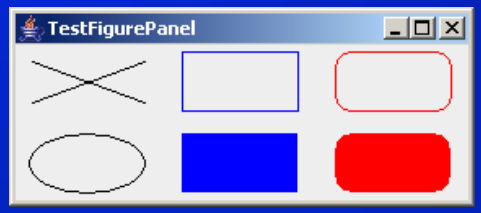
This example develops a useful class for displaying various figures. The class enables the user to set the figure type and specify whether the figure is filled, and displays the figure on a panel.



FigurePanel

Test FigurePanel

This example develops a useful class for displaying various figures. The class enables the user to set the figure type and specify whether the figure is filled, and displays the figure on a panel.

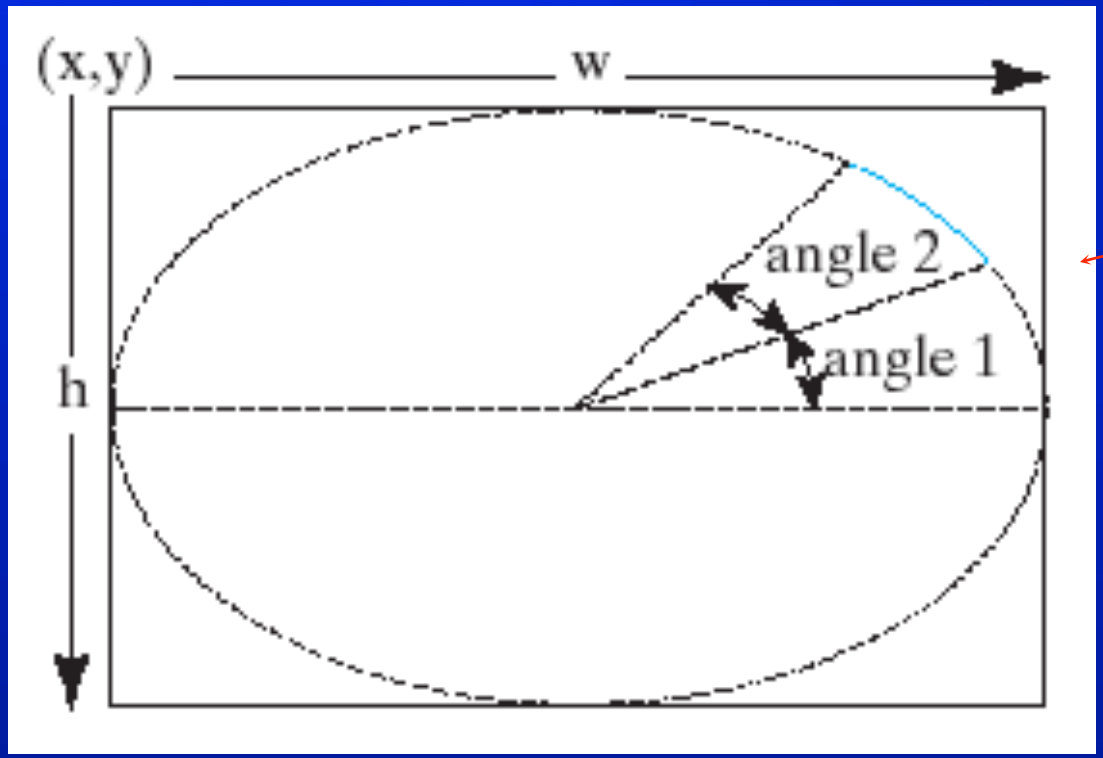


TestFigurePanel

Run

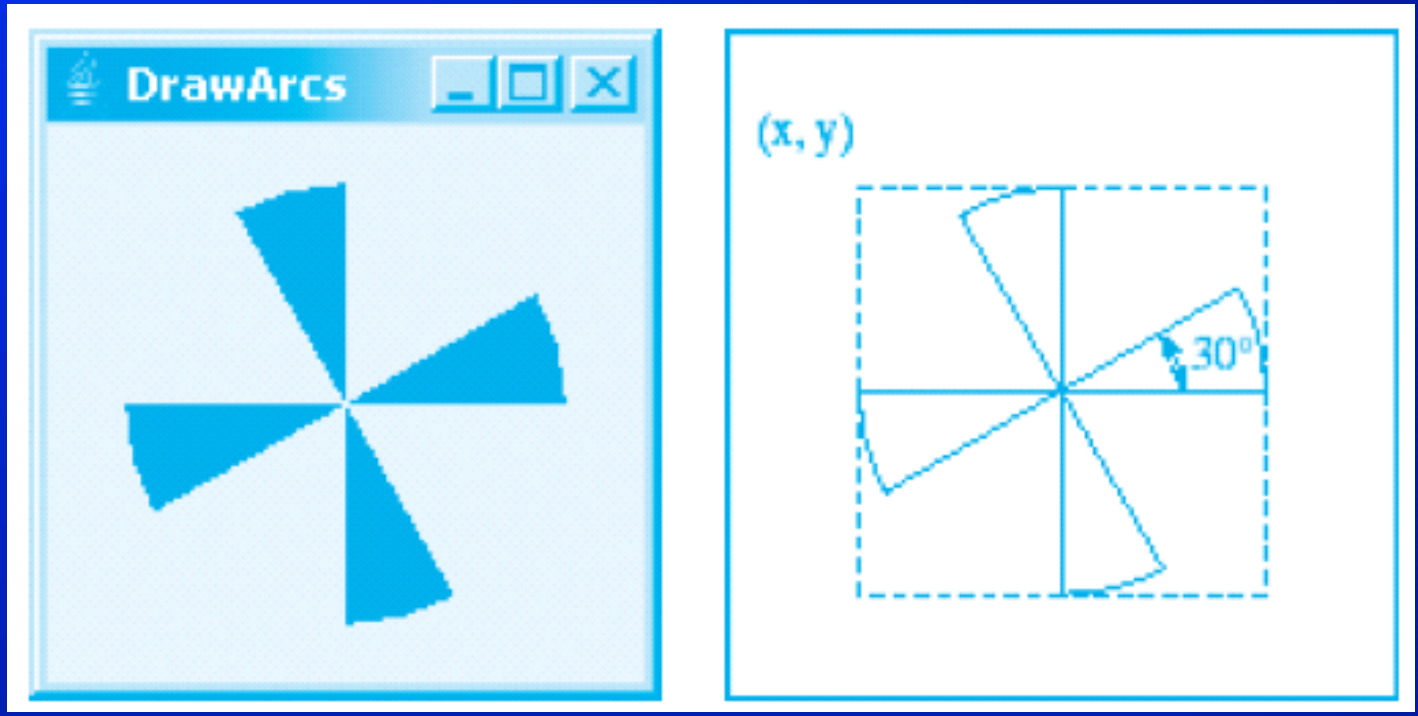
Drawing Arcs

drawArc(int x, int y, int w, int h, int angle1, int angle2);
fillArc(int x, int y, int w, int h, int angle1, int angle2);



Angles are in degree

Drawing Arcs Example



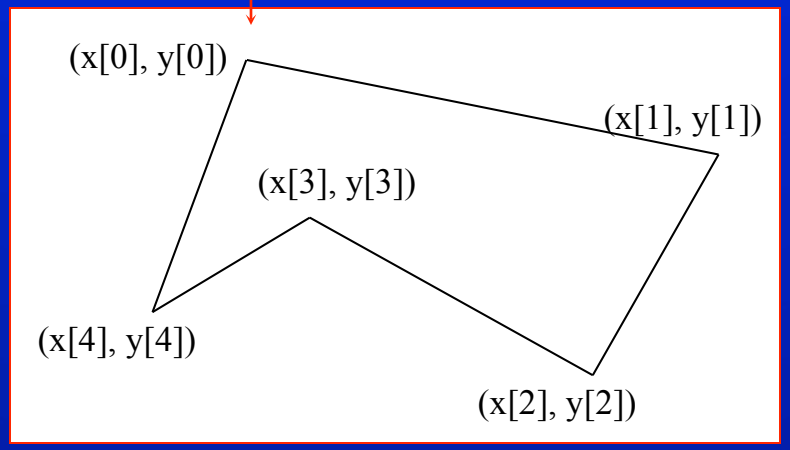
DrawArcs

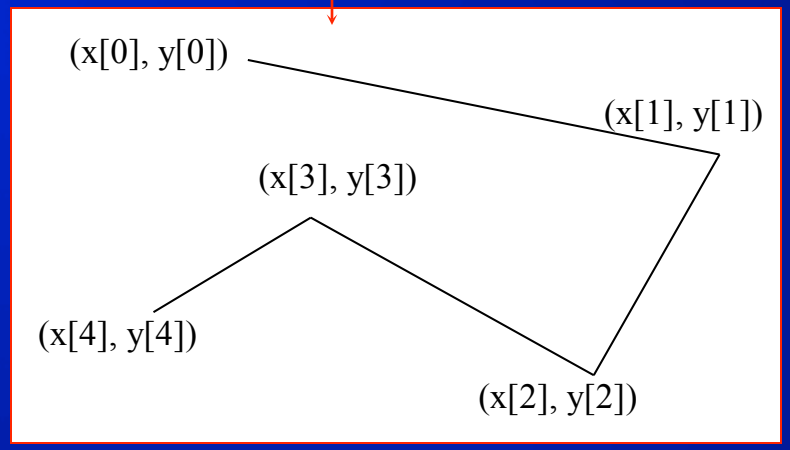
Run

Drawing Polygons and Polylines

```
int[] x = {40, 70, 60, 45, 20};
int[] y = {20, 40, 80, 45, 60};
g.drawPolygon(x, y, x.length);
```

```
g.drawPolyline(x, y, x.length);
```

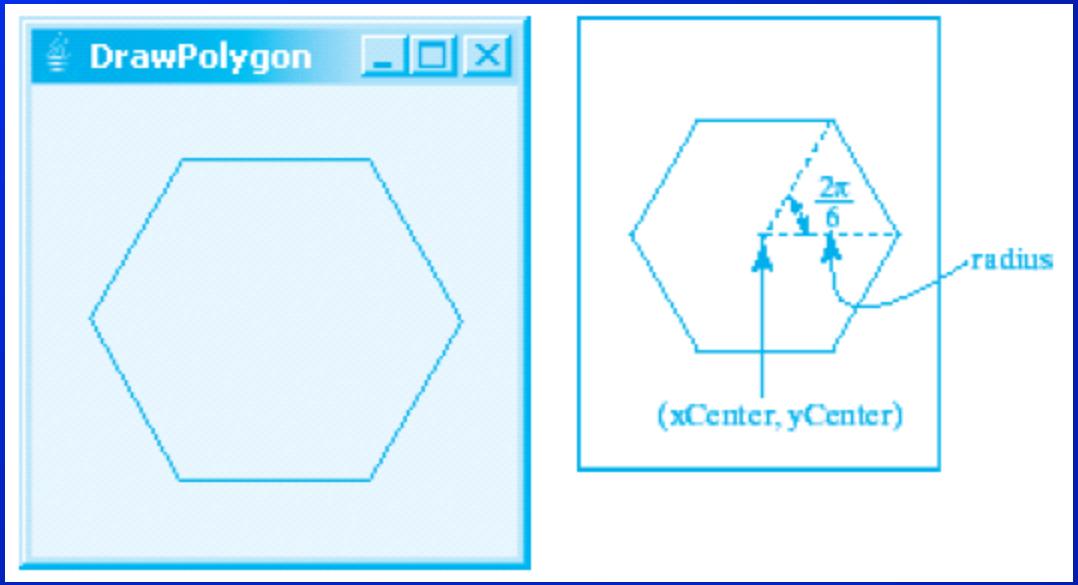




Drawing Polygons Using the Polygon Class

```
Polygon polygon = new Polygon();
polygon.addPoint(40, 59);
polygon.addPoint(40, 100);
polygon.addPoint(10, 100);
g.drawPolygon(polygon);
```

Drawing Polygons Example



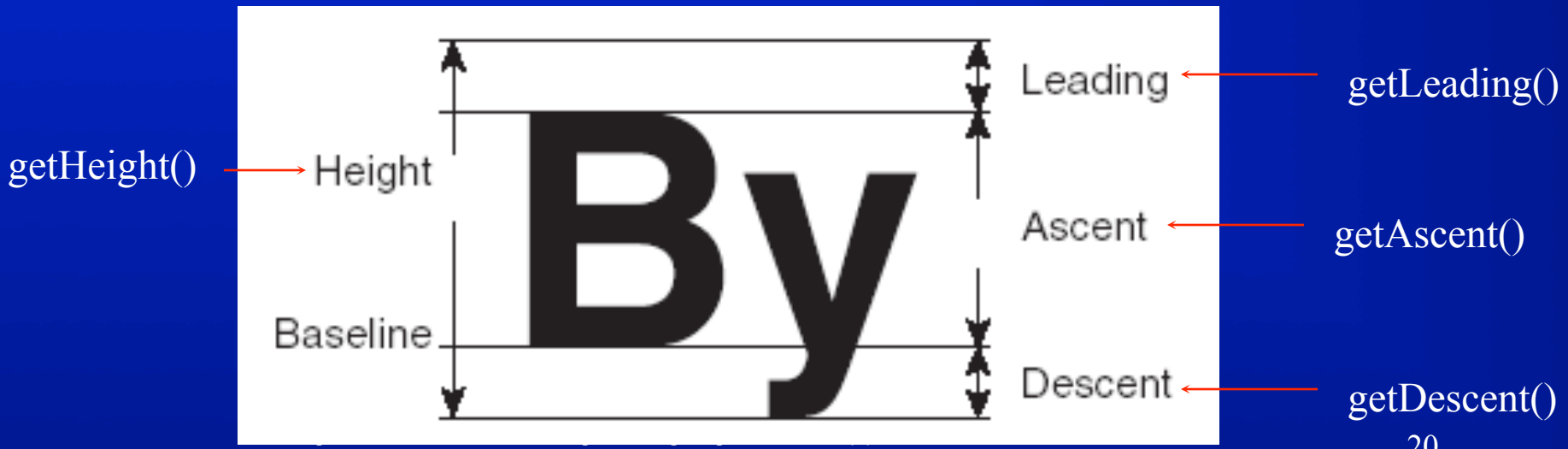
DrawPolygon

Run

Centering Display Using the FontMetrics Class

You can display a string at any location in a panel. Can you display it centered? To do so, you need to use the FontMetrics class to measure the exact width and height of the string for a particular font. A FontMetrics can measure the following attributes:

- public int getAscent()
- public int getDescent()
- public int getLeading()
- public int getHeight()
- public int stringWidth(String str)



The FontMetrics Class

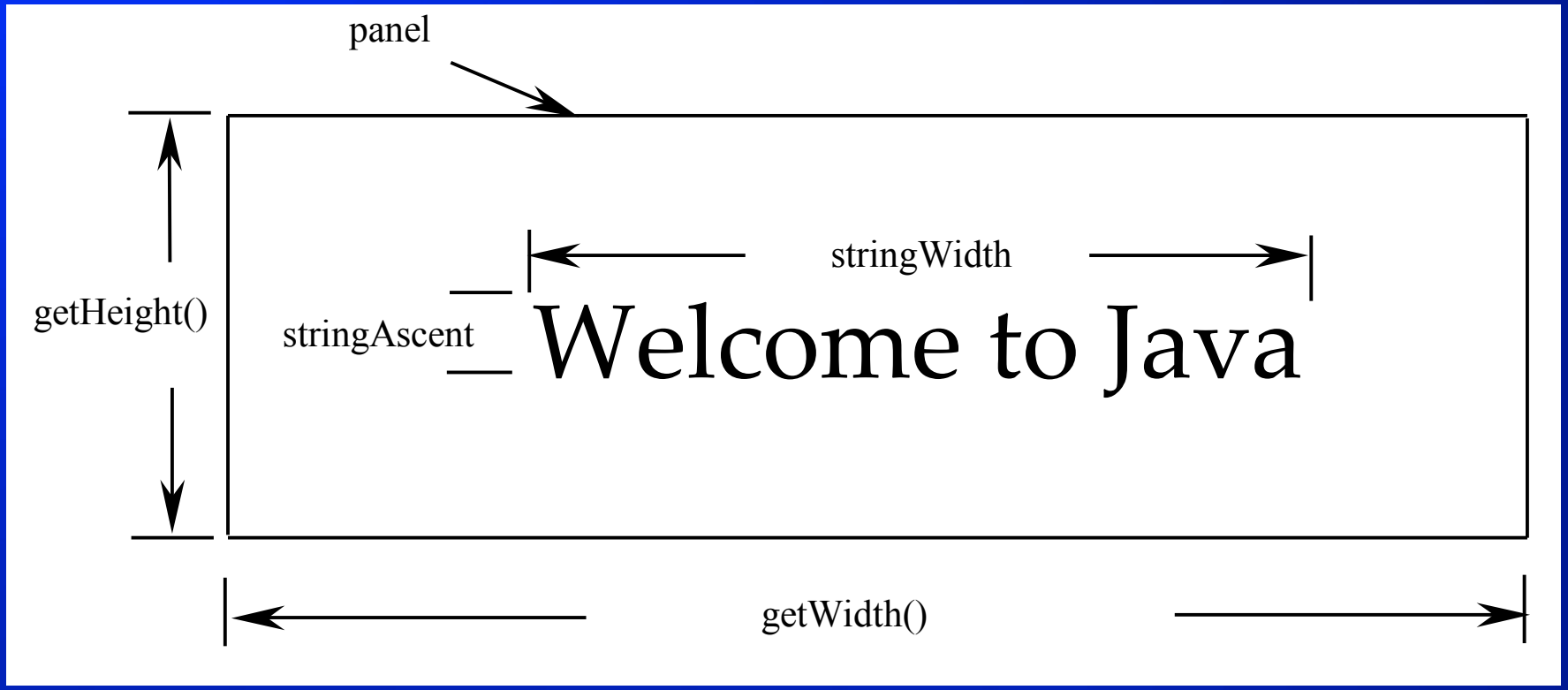
FontMetrics is an abstract class. To get a FontMetrics object for a specific font, use the following getFontMetrics methods defined in the Graphics class:

- public FontMetrics getFontMetrics(Font f)

Returns the font metrics of the specified font.

- public FontMetrics getFontMetrics()

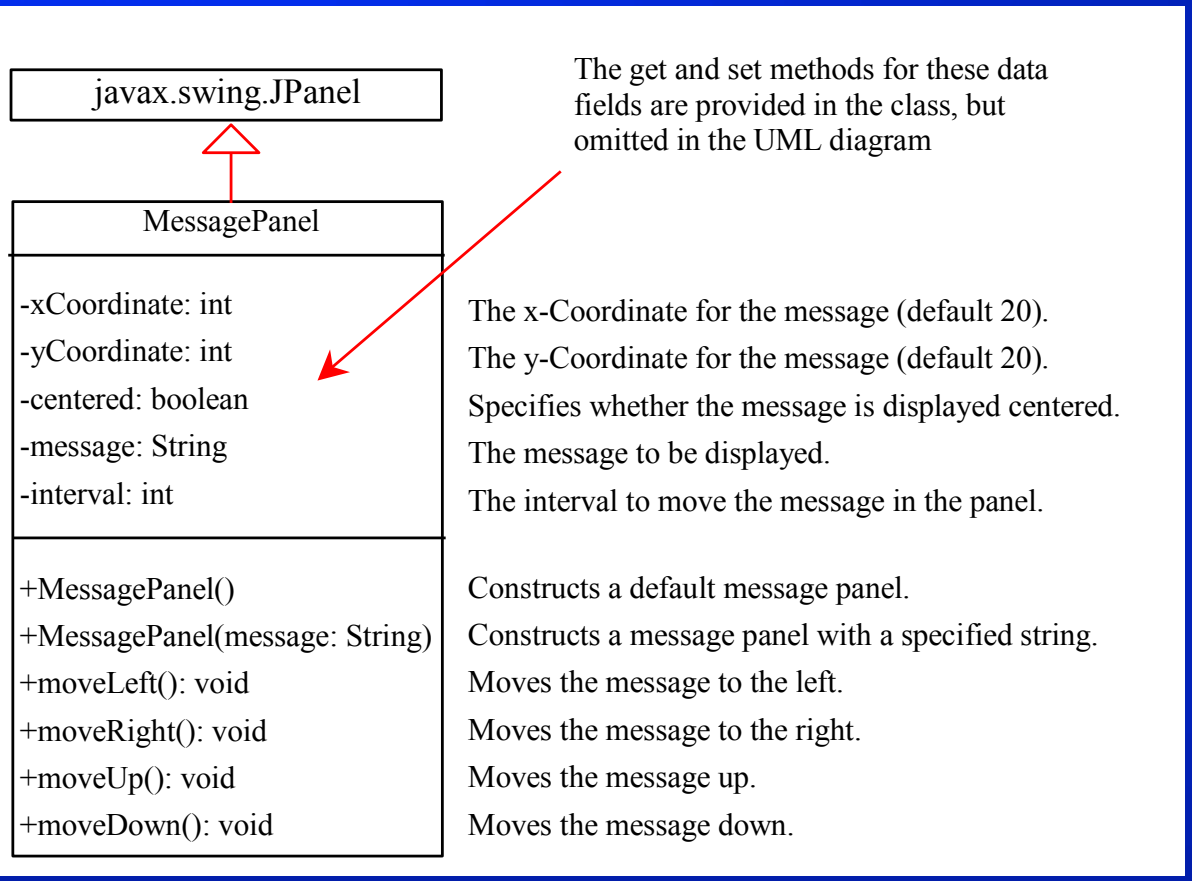
Returns the font metrics of the current font.



TestCenterMessage

Run

Case Study: MessagePanel



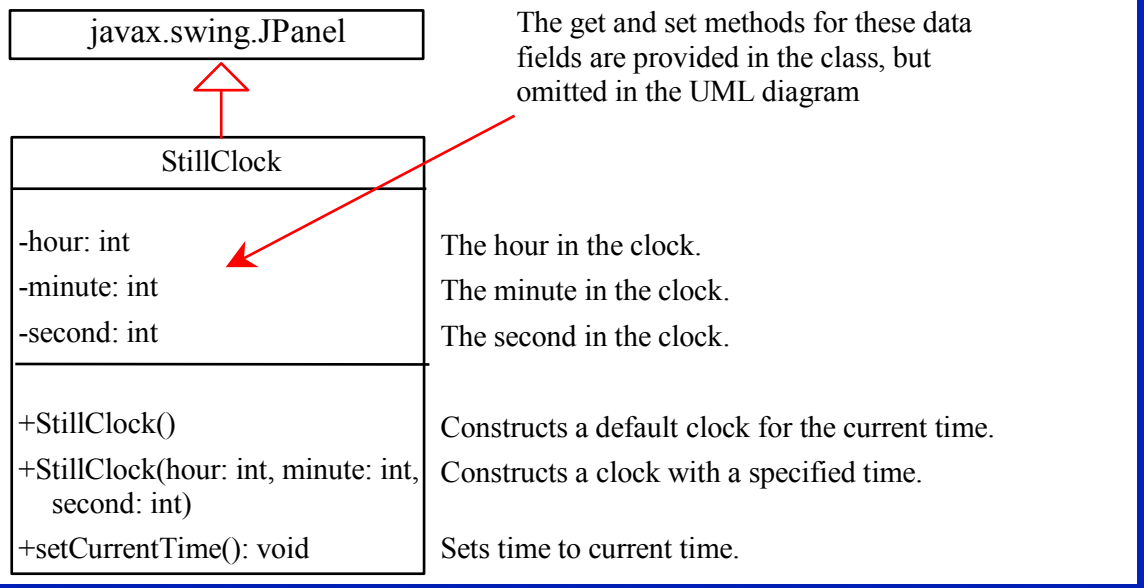
This case study develops a useful class that displays a message in a panel. The class enables the user to set the location of the message, center the message, and move the message with the specified interval.

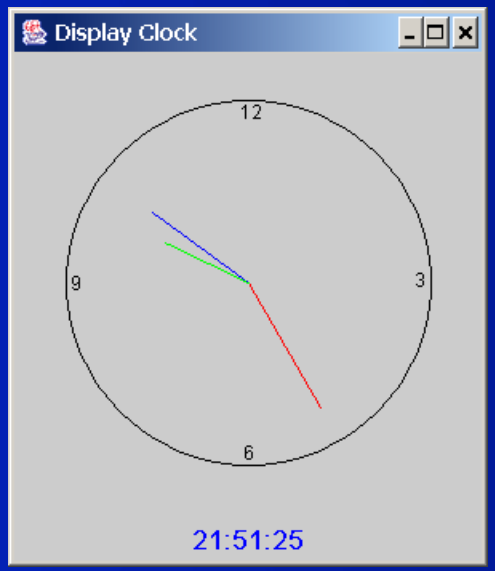
MessagePanel

TestMessagePanel

Run

Case Study: StillClock





StillClock

DisplayClock

Run

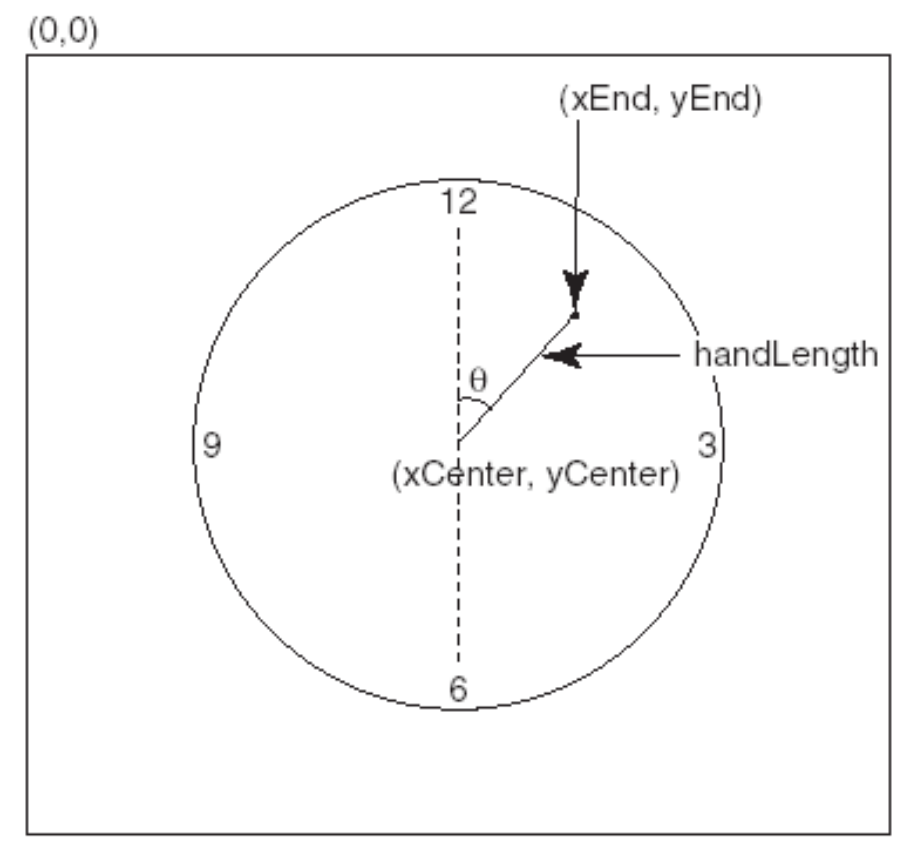
Drawing Clock

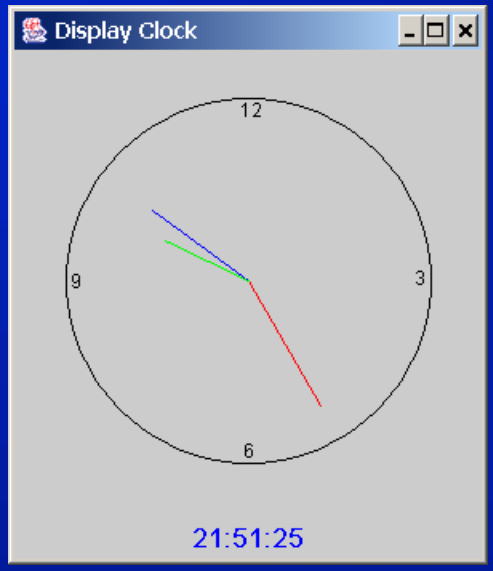
$xEnd = xCenter + handLength \times \sin(\theta)$

$yEnd = yCenter - handLength \times \cos(\theta)$

Since there are sixty seconds in one minute, the angle for the second hand is

$second \times (2\pi/60)$



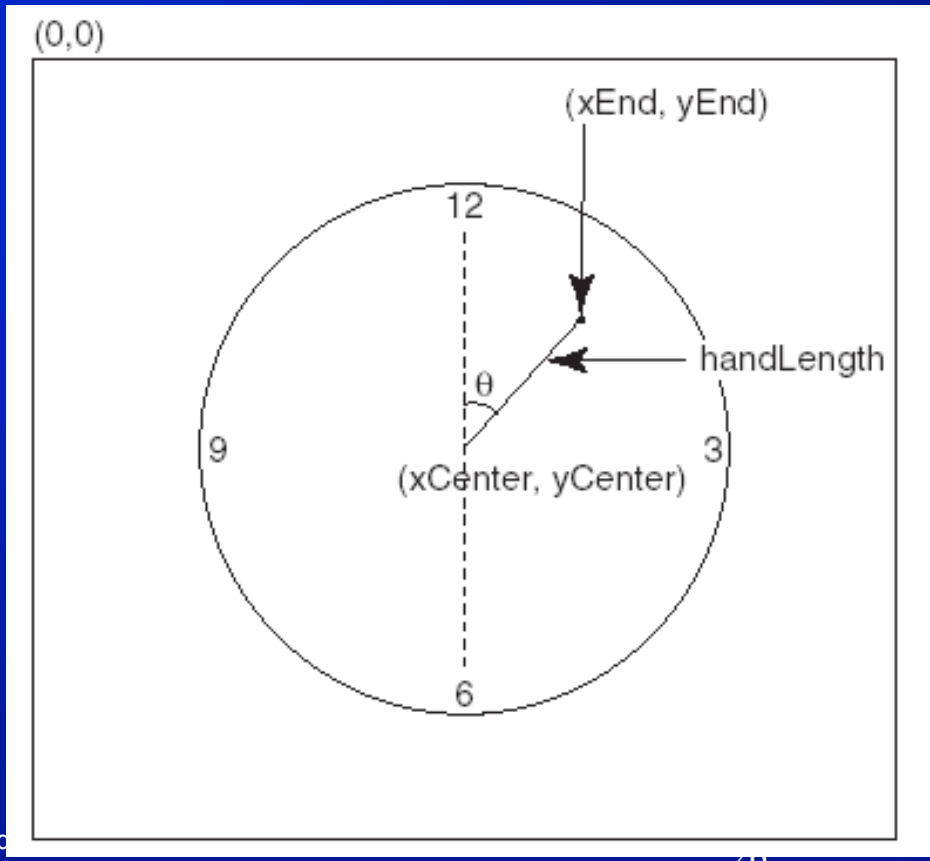


Drawing Clock, cont.

$$xEnd = xCenter + handLength \times \sin(\theta)$$

$$yEnd = yCenter - handLength \times \cos(\theta)$$

The position of the minute hand is determined by the minute and second. The exact minute value combined with seconds is minute + second/60. For example, if the time is 3 minutes and 30 seconds. The total minutes are 3.5. Since there are sixty minutes in one hour, the angle for the minute hand is (minute + second/60) × (2π/60)

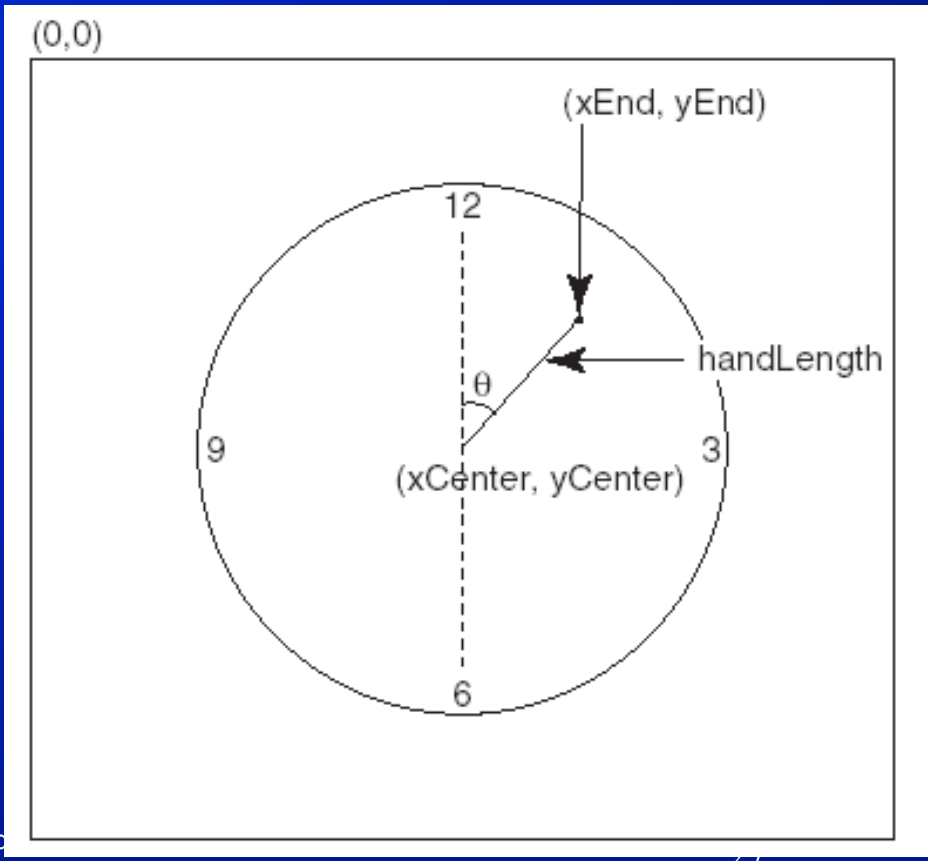


Drawing Clock, cont.

$xEnd = xCenter + handLength \times \sin(\theta)$

$yEnd = yCenter - handLength \times \cos(\theta)$

Since one circle is divided into twelve hours, the angle for the hour hand is
(hour + minute/60 + second/(60 × 60))) × (2π/12)



Displaying Image Icons

You learned how to create image icons and display image icons in labels and buttons. For example, the following statements create an image icon and display it in a label:

```
ImageIcon icon = new ImageIcon("image/us.gif");
JLabel jlblImage = new JLabel(imageIcon);
```

An image icon displays a fixed-size image. To display an image in a flexible size, you need to use the java.awt.Image class. An image can be created from an image icon using the getImage() method as follows:

```
Image image = imageIcon.getImage();
```

Displaying Images

Using a label as an area for displaying images is simple and convenient, but you don't have much control over how the image is displayed. A more flexible way to display images is to use the drawImage method of the Graphics class on a panel. Four versions of the drawImage method are shown here.

java.awt.Graphics	
+drawImage(image: Image, x: int, y: int, bgcolor: Color, observer: ImageObserver): void	Draws the image in a specified location. The image's top-left corner is at (x, y) in the graphics context's coordinate space. Transparent pixels in the image are drawn in the specified color bgcolor. The observer is the object on which the image is displayed. The image is cut off if it is larger than the area it is being drawn on.
+drawImage(image: Image, x: int, y: int, observer: ImageObserver): void	Same as the preceding method except that it does not specify a background color.
+drawImage(image: Image, x: int, y: int, width: int, height: int, observer: ImageObserver): void	Draws a scaled version of the image that can fill all of the available space in the specified rectangle.
+drawImage(image: Image, x: int, y: int, width: int, height: int, bgcolor: Color, observer: ImageObserver): void	Same as the preceding method except that it provides a solid background color behind the image being drawn.

Displaying Images Example

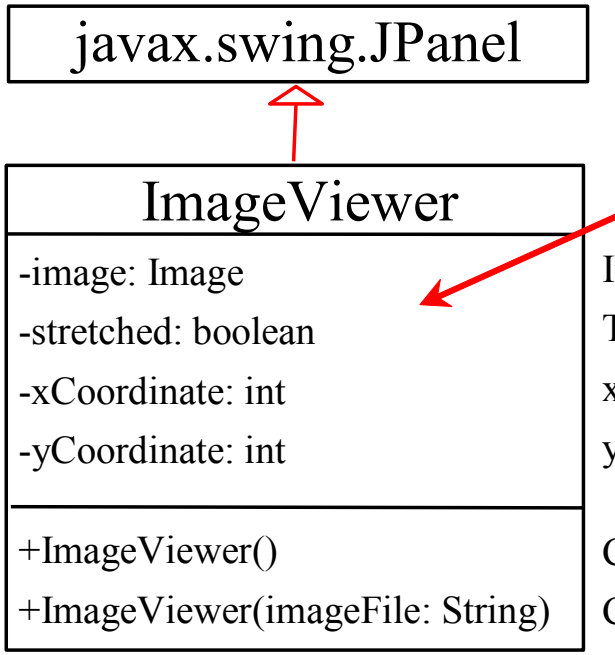
This example gives the code that displays an image from image/us.gif. The file image/us.gif is under the class directory. The Image from the file is created in the program. The drawImage method displays the image to fill in the whole panel, as shown in the figure.

DisplayImage

Run

Case Study: ImageViewer Class

Displaying an image is a common task in Java programming. This case study develops a reusable component named ImageViewer that displays an image in a panel. The ImageViewer class contains the properties image, imageFilename, stretched, xCoordinate, and yCoordinate.



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

Image in the image viewer.

True if the image is stretched in the viewer.

x-coordinate of the upper-left corner of the image in the viewer.

y-coordinate of the upper-left corner of the image in the viewer.

Constructs an image viewer with no image.

Constructs an image viewer with the specified image file.

ImageView Example

This example gives an example that creates six images using the ImageViewer class.

SixFlags

Run