EE3206
Java Programming and Applications

Lecture 9
Graphical User Interface (GUI)

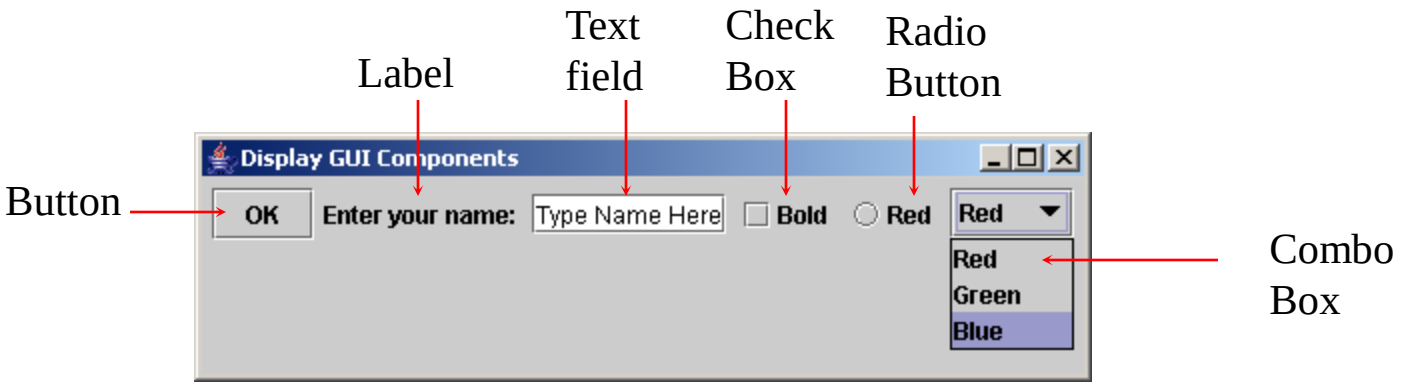
Mr. Van Ting, Dept. of EE, CityU HK

Intended Learning Outcomes

- To distinguish simple GUI components.
- To describe the Java GUI API hierarchy.
- To create user interfaces using frames, panels, and simple UI components.
- To understand the role of layout managers.
- To use the FlowLayout, GridLayout, and BorderLayout managers to layout components in a container.
- To specify colors and fonts using the Color and Font classes.
- To use JPanel as subcontainers.
- To understand Java coordinate systems.
- To draw things using the methods in the Graphics class.
- To obtain a graphics context using the getGraphics() method.
- To override the paintComponent method to draw things on a graphical context.
- To use a panel as a canvas to draw things.
- To draw strings, lines, rectangles, ovals, arcs, and polygons.

Creating GUI Objects

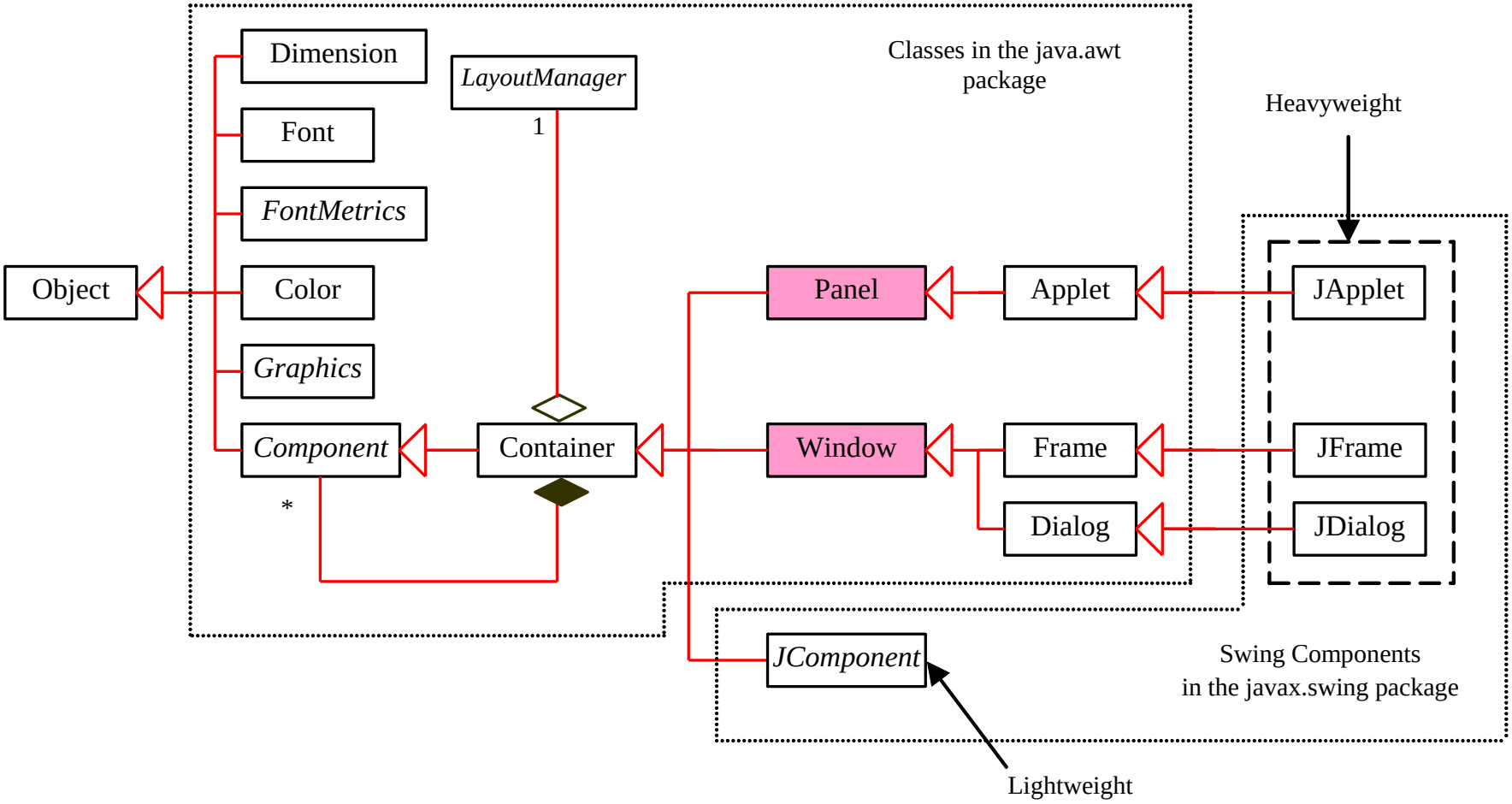
- // Create a button with text OK
  - JButton jbtOK = new JButton("OK");
- // Create a label with text "Enter your name: "
  - JLabel jlblName = new JLabel("Enter your name: ");
- // Create a text field with text "Type Name Here"
  - JTextField jtfName = new JTextField("Type Name Here");
- // Create a check box with text bold
  - JCheckBox jchkBold = new JCheckBox("Bold");
- // Create a radio button with text red
  - JRadioButton jrbRed = new JRadioButton("Red");
- // Create a combo box with choices red, green, and blue
  - JComboBox jcboColor = new JComboBox(new String[]{"Red", "Green", "Blue"});



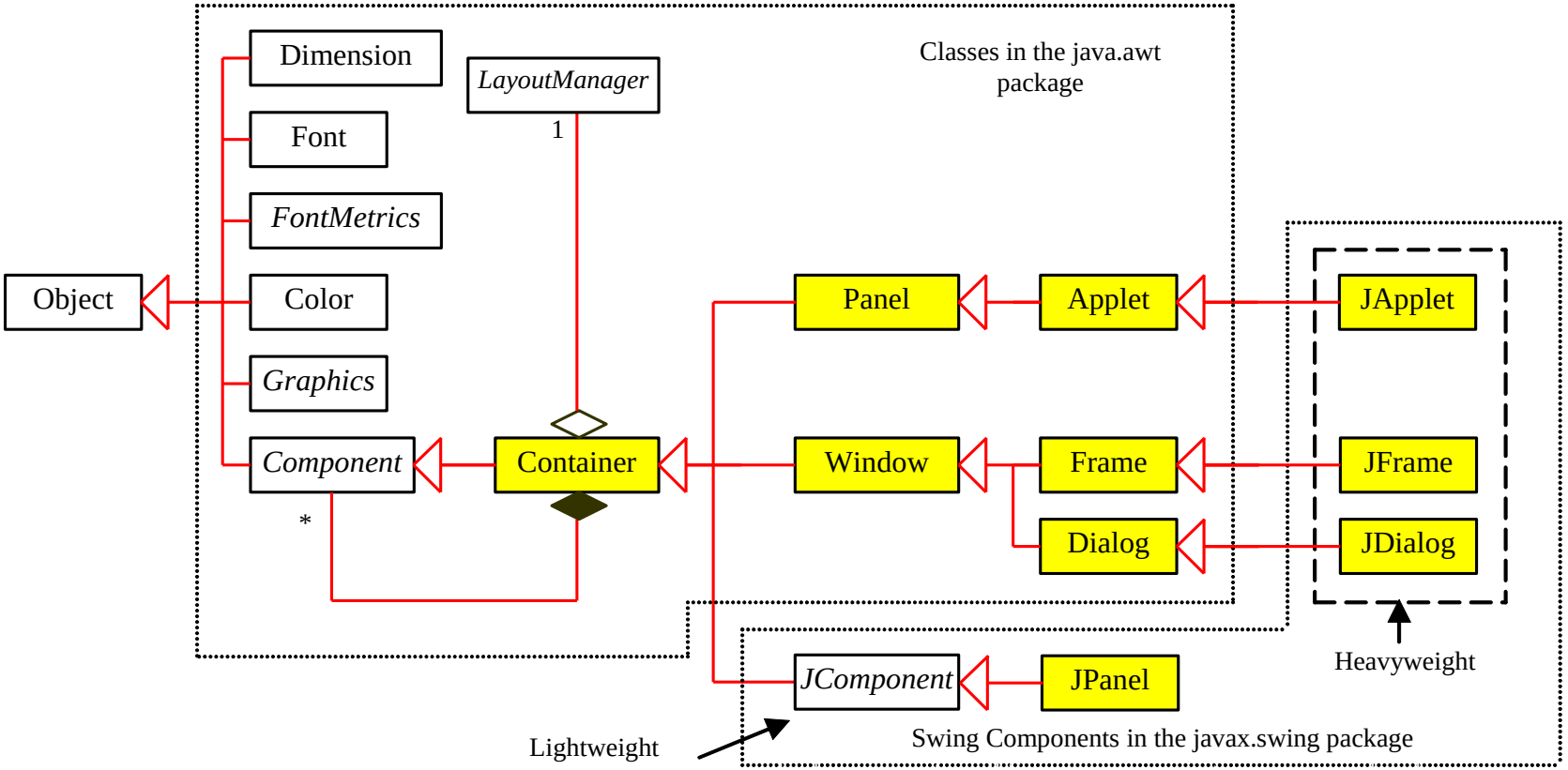
AWT and Swing

- When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT). For every platform on which Java runs, the AWT components are automatically mapped to the platform-specific components through their respective agents, known as peers.
  - AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects.
  - Besides, AWT is prone to platform-specific bugs because its peer-based approach relies heavily on the underlying platform.
- With the release of Java 2, the AWT user-interface components were replaced by a more robust, versatile, and flexible library known as Swing components.
  - Swing components are painted directly on canvases using pure Java code, except for components that are the subclasses of java.awt.Window or java.awt.Panel, which must be drawn using native GUI on a specific platform.
  - Swing components are less dependent on the target platform and use less of the native GUI resource. For this reason, Swing components that don't rely on native GUI are referred to as lightweight components, and AWT components are referred to as heavyweight components.

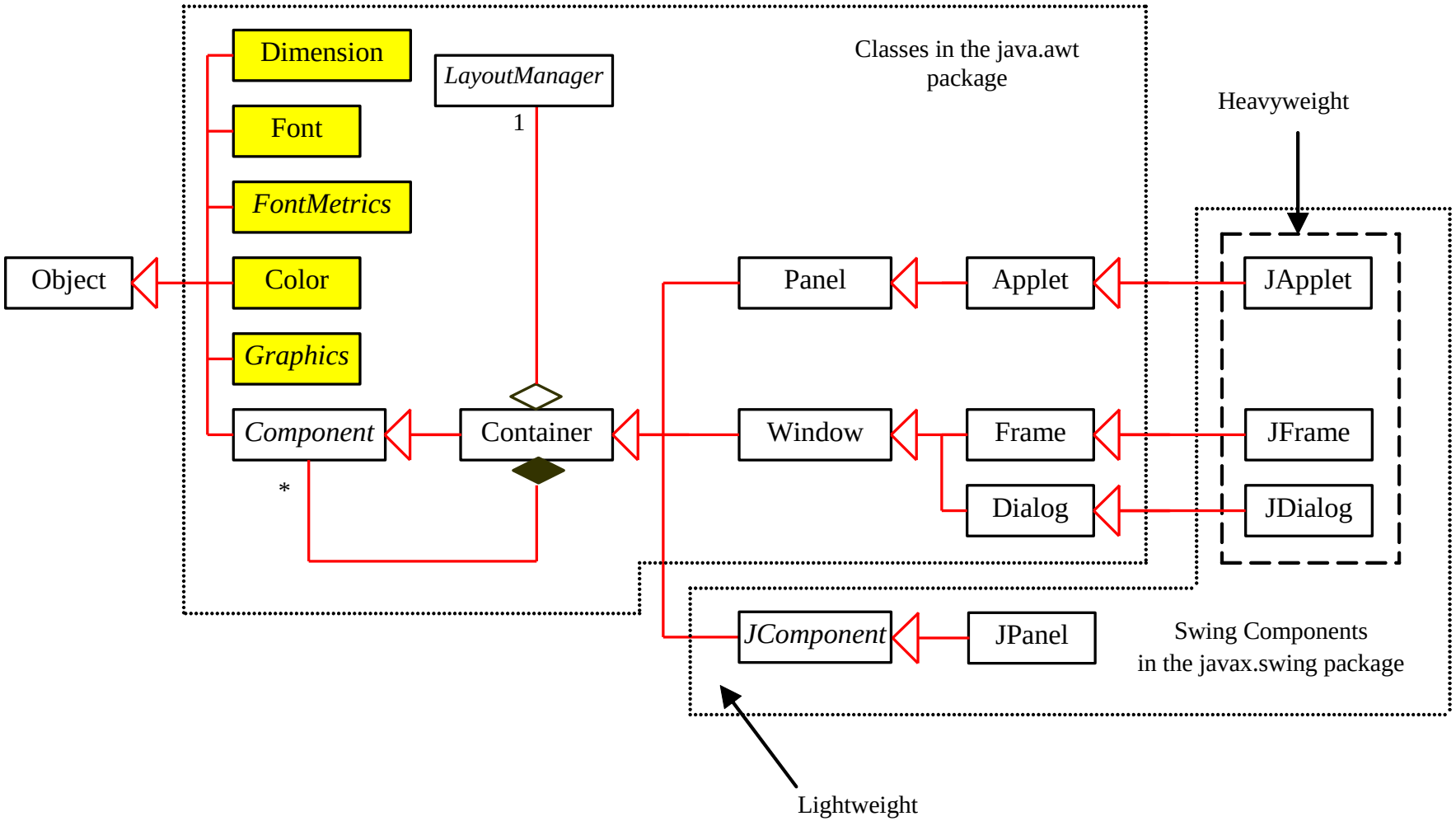
Heavyweight (AWT) vs. Lightweight (Swing)



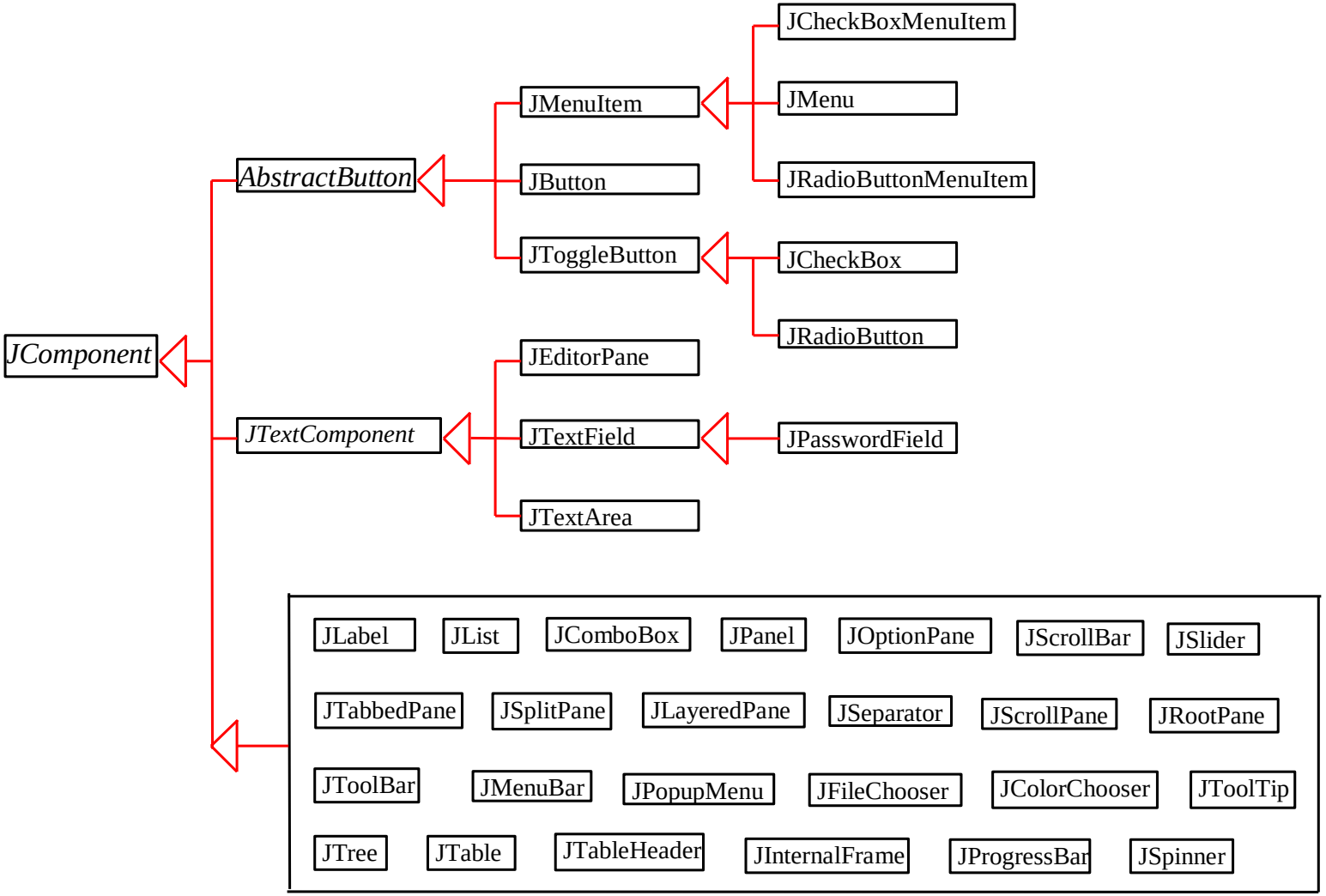
GUI Container Classes



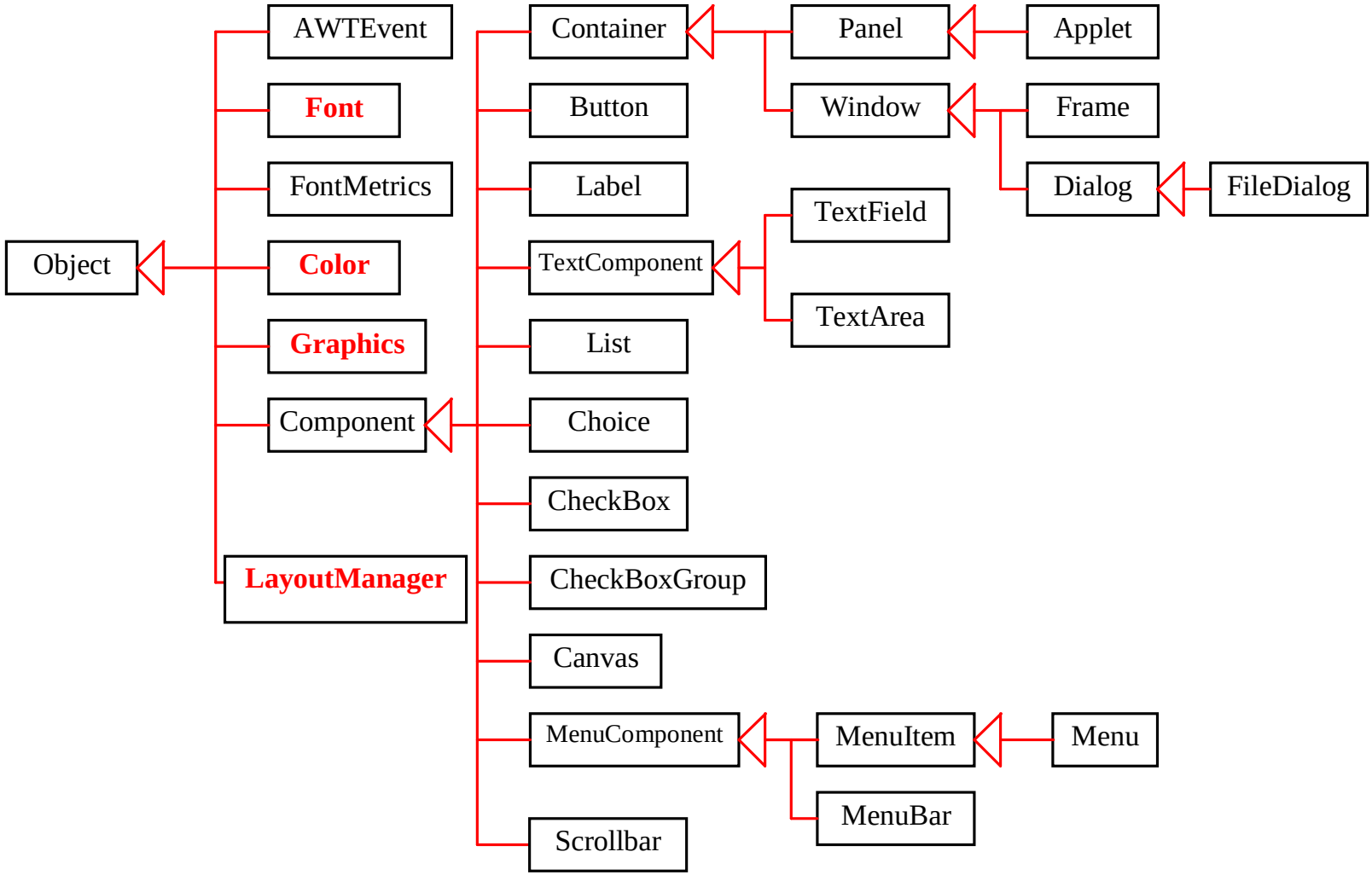
GUI Helper Classes



Swing GUI Components



AWT (for your reference)



Frames and JFrame Class

- Frame is a top-level container, a window that is not contained inside another window.
- Frame is the base to contain other user interface components in Java GUI applications.
- For Swing GUI programs, use JFrame class to create widows.

javax.swing.JFrame	
+JFrame()	Creates a default frame with no title.
+JFrame(title: String)	Creates a frame with the specified title.
+setSize(width: int, height: int): void	Specifies the size of the frame.
+setLocation(x: int, y: int): void	Specifies the upper-left corner location of the frame.
+setVisible(visible: boolean): void	Sets true to display the frame.
+setDefaultCloseOperation(mode: int): void	Specifies the operation when the frame is closed.
+setLocationRelativeTo (c: Component): void	Sets the location of the frame relative to the specified component. If the component is null, the frame is centered on the screen.

Creating Frames

- Here is an example that creates a frame of size 400x300 and makes it visible.

```
import javax.swing.*;
public class MyFrame {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Test Frame");
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

- To add a button into the frame
  - frame.add(new JButton("OK"));

MyFrame

MyFrameWithComponents

Layout Managers

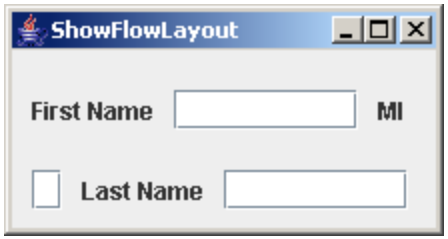
- Java's layout managers provide a level of abstraction to automatically map your user interface on all window systems.
- The UI components are placed in containers. Each container has a layout manager to arrange the UI components within the container.
- Layout managers are set in containers using the setLayout(LayoutManager) method in a container.
- Three simple Layout Managers
  - FlowLayout
  - GridLayout
  - BorderLayout

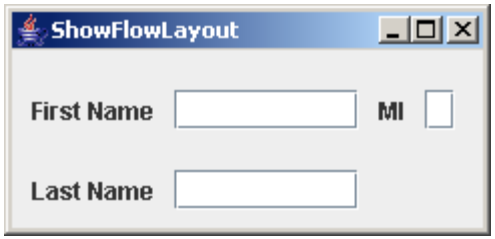
FlowLayout Manager

- Flow means – from left to right or right to left.
- Write a program that adds three labels and text fields into the content pane of a frame with a FlowLayout manager.

java.awt.FlowLayout	
-alignment: int	The alignment of this layout manager (default: CENTER).
-hgap: int	The horizontal gap of this layout manager (default: 5 pixels).
-vgap: int	The vertical gap of this layout manager (default: 5 pixels).
+FlowLayout()	Creates a default FlowLayout manager.
+FlowLayout(alignment: int)	Creates a FlowLayout manager with a specified alignment.
+FlowLayout(alignment: int, hgap: int, vgap: int)	Creates a FlowLayout manager with a specified alignment, horizontal gap, and vertical gap.

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





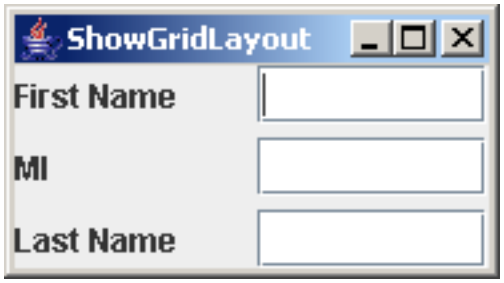
ShowFlowLayout

GridLayout Manager

- Rewrite the program in the preceding example using a GridLayout manager instead of a FlowLayout manager to display the labels and text fields.

java.awt.GridLayout	
-rows: int	The number of rows in this layout manager (default: 1).
-columns: int	The number of columns in this layout manager (default: 1).
-hgap: int	The horizontal gap of this layout manager (default: 0).
-vgap: int	The vertical gap of this layout manager (default: 0).
+GridLayout()	Creates a default GridLayout manager.
+GridLayout(rows: int, columns: int)	Creates a GridLayout with a specified number of rows and columns.
+GridLayout(rows: int, columns: int, hgap: int, vgap: int)	Creates a GridLayout manager with a specified number of rows and columns, horizontal gap, and vertical gap.

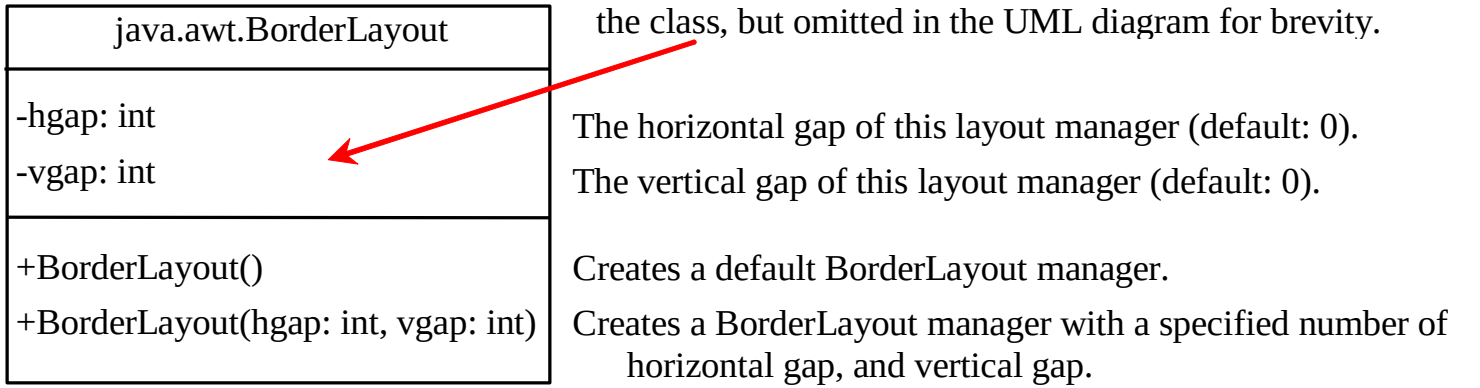
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.



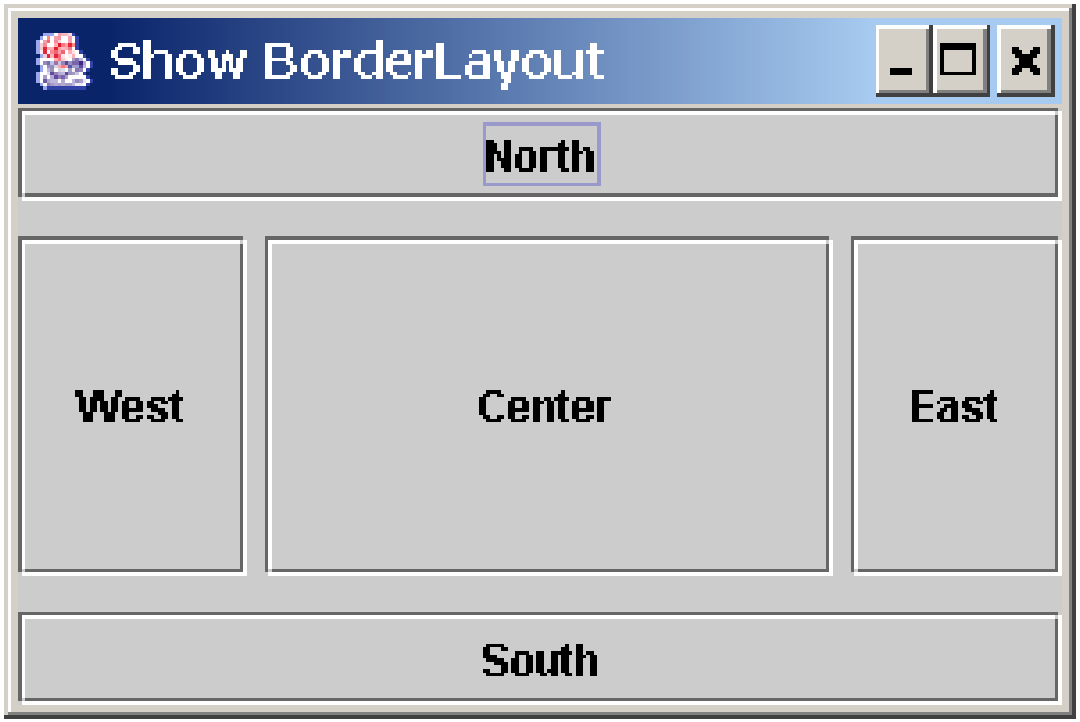
ShowGridLayout

BorderLayout Manager

- The BorderLayout manager divides the container into five areas: East, South, West, North, and Center. Components are added to a BorderLayout by using the add method.
  - add(Component, constraint);
  - where constraint is BorderLayout.EAST, BorderLayout.SOUTH,
  - BorderLayout.WEST, BorderLayout.NORTH, or BorderLayout.CENTER.



BorderLayout Example



ShowBorderLayout

The Color Class

- You can set colors for GUI components by using the java.awt.Color class. Colors are made of red, green, and blue components, each of which is represented by a byte value that describes its intensity, ranging from 0 (darkest shade) to 255 (lightest shade). This is known as the RGB model.
  - Color c = new Color(r, g, b);
  - r, g, and b specify a color by its red, green, and blue components.

- Example:
  - Color c = new Color(228, 100, 255);

Standard Colors

- There are thirteen standard color constants:
  - BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, and YELLOW.

- You can use these two methods to set the component's background and foreground colors:
  - setBackground(Color c) // background color
  - setForeground(Color c) // text color

- Example: // jbt is an instance of JButton
  - jbt.setBackground(Color.YELLOW);
  - jbt.setBackground(Color.LIGHT_GRAY);

The Font Class

- Font Names

  Standard font names that are supported in all platforms are: SansSerif, Serif, Monospaced, Dialog, or DialogInput.

- Font Styles

  Font.PLAIN, Font.BOLD, Font.ITALIC

- Font myFont = new Font(name, style, size);
- Example:
  - Font myFont = new Font("SansSerif", Font.BOLD, 16);
  - Font myFont = new Font("Serif", Font.BOLD|Font.ITALIC, 12);
  - JButton jbtOK = new JButton("OK");
  - jbtOK.setFont(myFont);

a bitwise union

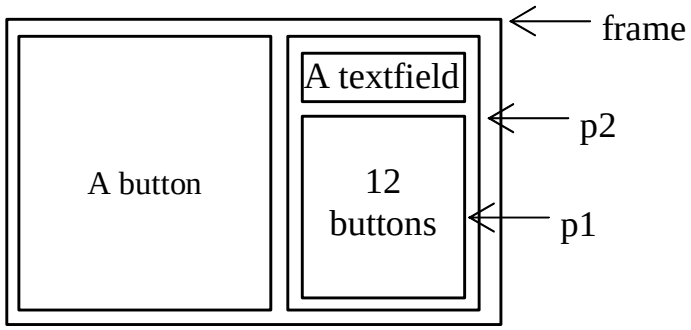
Using Panels as Sub-Containers

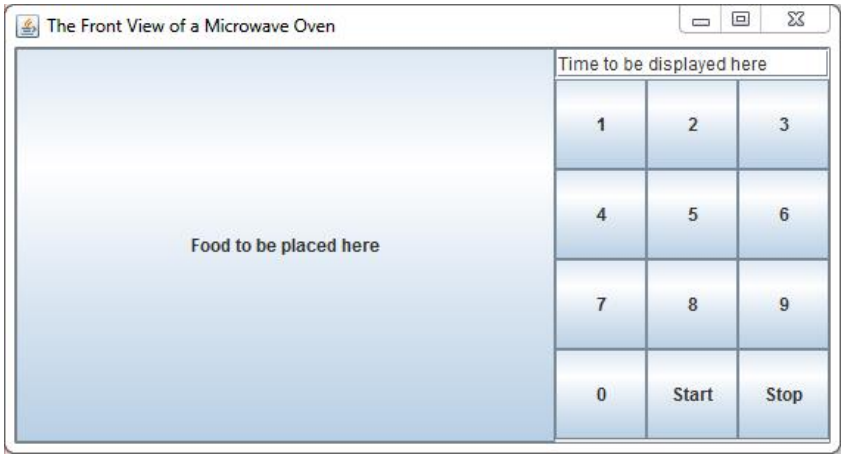
- Panels act as sub-containers for grouping user interface components.
- It is recommended that you place the user interface components in panels and place the panels in a frame. You can also place panels in a panel.
  - Decompose into manageable sub-interface
  - Easy for management
- You can use new JPanel() to create a panel with a default FlowLayout manager or new JPanel(LayoutManager layout) to create a panel with the specified layout manager.
- Use the add(Component) method to add a component to the panel.

- For example,
  - JPanel p = new JPanel(); // FlowLayout by default
  - p.add(new JButton("OK"));

Testing Panels Example

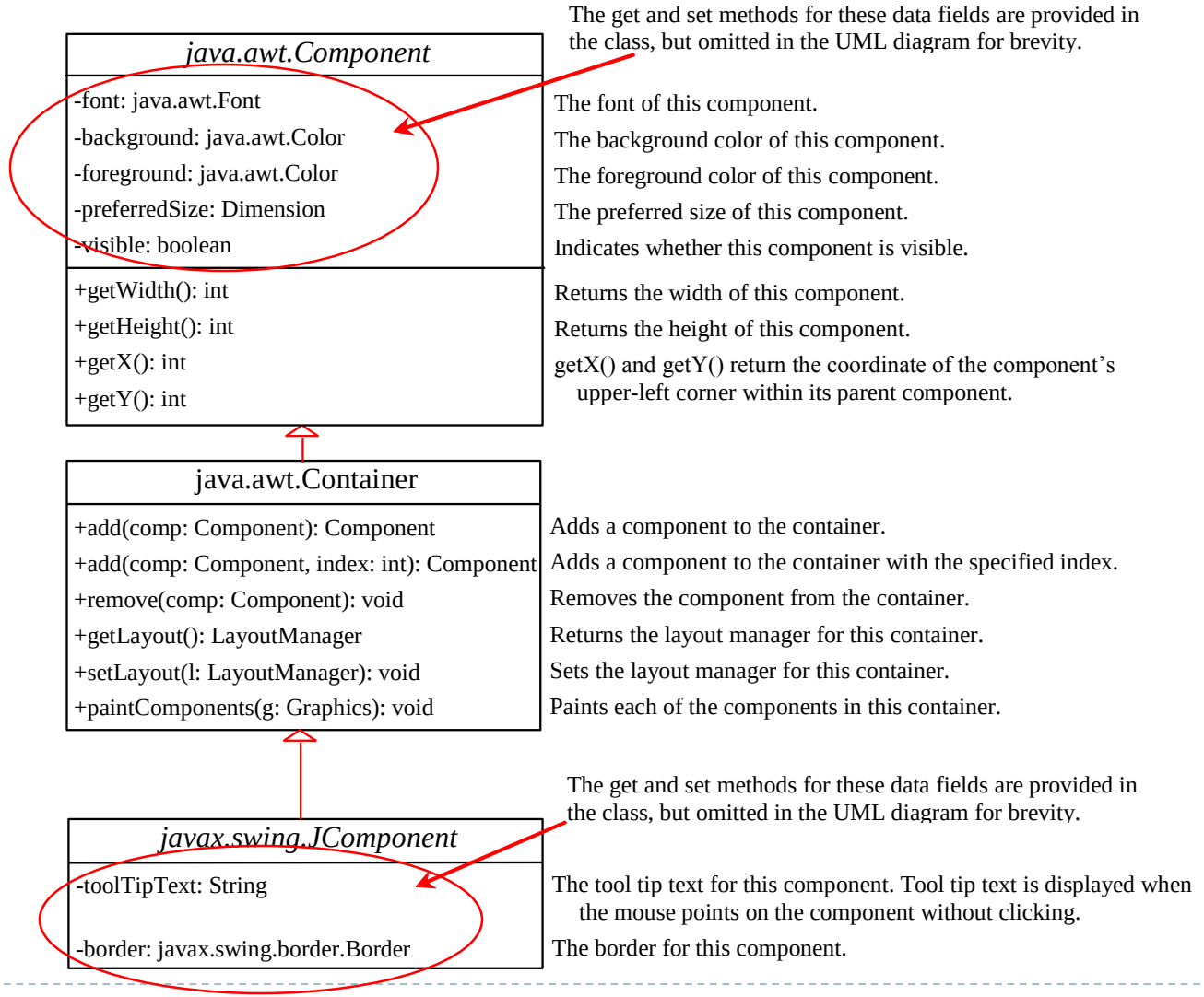
- This example uses panels to organize components. The program creates a user interface for a Microwave oven.





TestPanels

Common Features of Swing Components



Borders

- You can set a border on any object of the JComponent class. Swing has several types of borders. To create a titled border, use
  - new TitledBorder(String title);
- To create a line border, use
  - new LineBorder(Color color, int width);
- where width specifies the thickness of the line.
- For example, the following code displays a titled border on a panel:
  - JPanel panel = new JPanel();
  - panel.setBorder(new TitledBorder("My Panel"));

TestSwingCommonFeatures

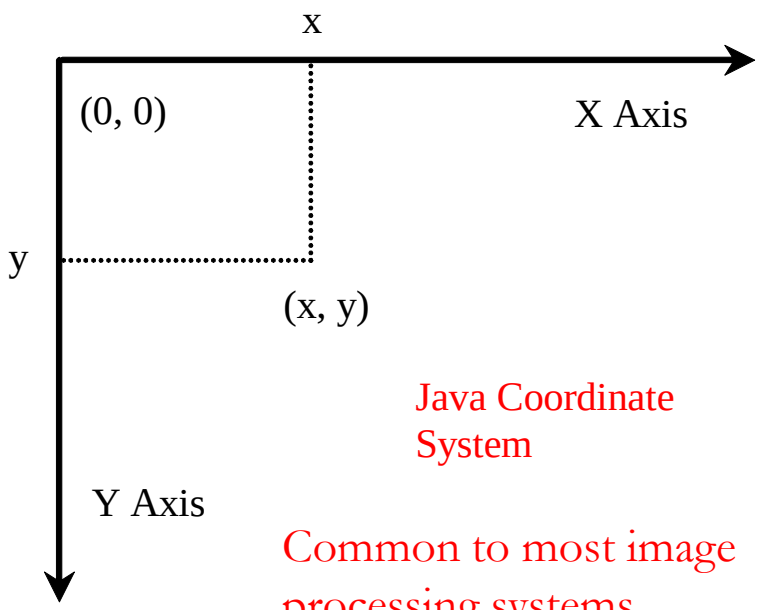
Image Icons

- Java uses the javax.swing.ImageIcon class to represent an icon. An icon is a fixed-size picture; typically it is small and used to decorate components. Images are normally stored in image files.
- You can use new ImageIcon(filename) to construct an image icon. For example, the following statement creates an icon from an image file us.gif in the image directory under the current class path:

  - ImageIcon icon = new ImageIcon("image/us.gif");

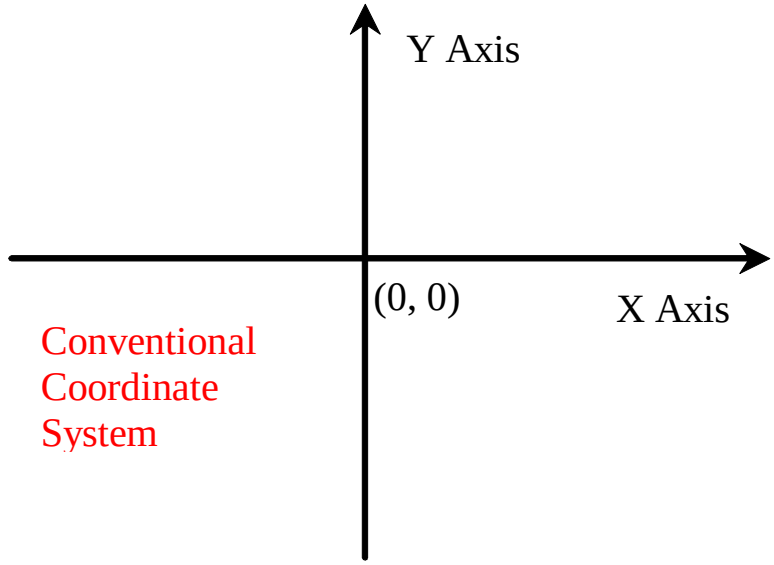
TestImageIcon

Java Coordinate System



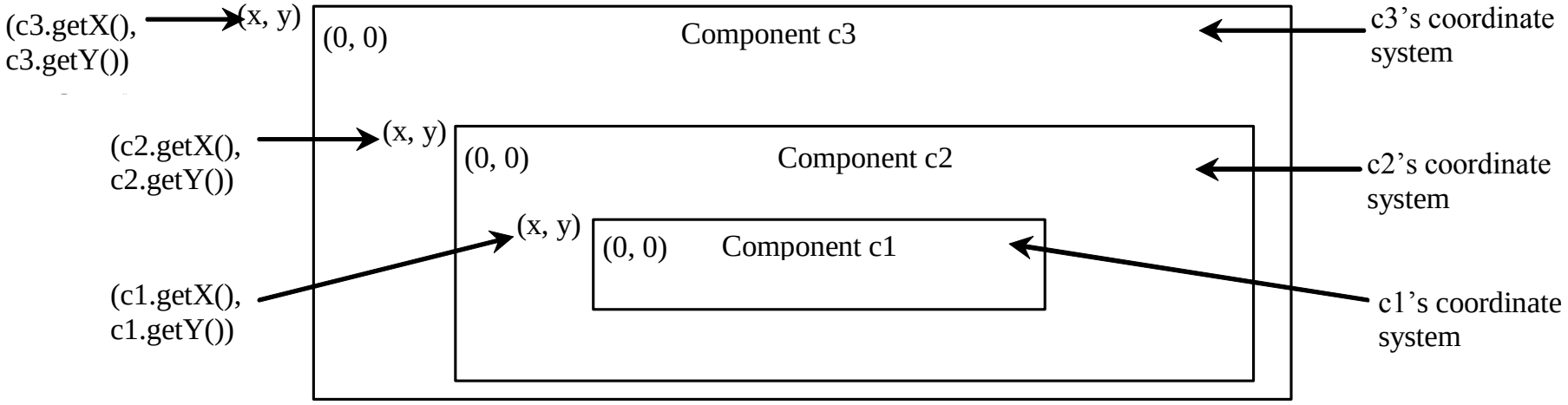
Java Coordinate System

Common to most image processing systems



Conventional Coordinate System

Each GUI Component Has its Own Coordinate System



Obtaining Graphics Object

- The Graphics class is an abstract class that provides a device-independent graphics interface for displaying figures and images on the screen on different platforms.

- Whenever a component (e.g., a button, a label, a panel) is displayed, a Graphics object is created for the component on the native platform. This object can be obtained using the getGraphics() method (inherited from JComponent). For example, the graphics context for a **label object jlblBanner** can be obtained using

  - Graphics graphics = jlblBanner.getGraphics();

- You can then apply the methods in the Graphics class to draw things on the label's graphics context.

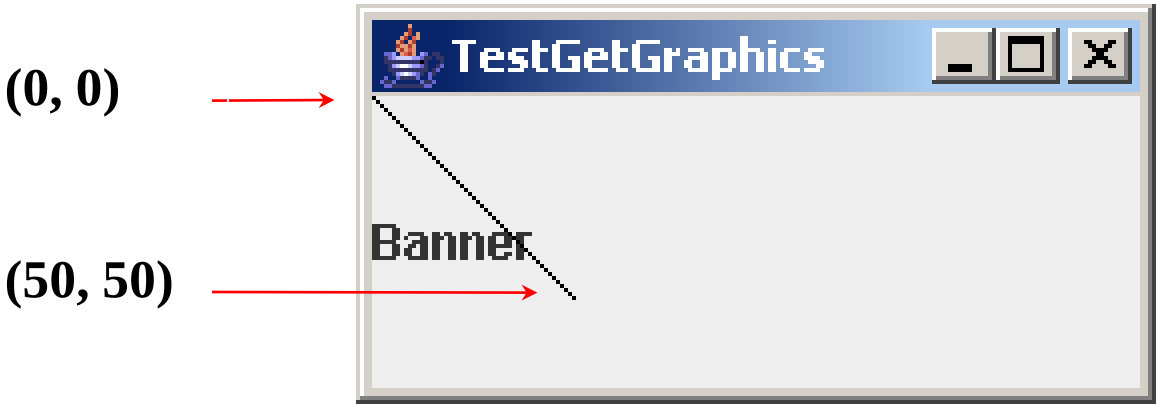
The Graphics Class

- You can use the Graphics class to draw:
  - strings
  - lines
  - rectangles
  - ovals
  - arcs
  - polygons
  - polylines

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.

A Drawing Example

- Draw a line and a text



TestGetGraphics

Problems With the Preceding Example

- If you resize the frame, the line is gone. Why?

- To fix this problem, you need to know its cause. When you resize the frame, the JVM invokes the paintComponent() method of a Swing component (e.g., a JLabel) to redisplay the graphics on the component. Since you did not draw a line in the paintComponent() method, the line is gone when the frame is resized.
- To permanently display the line, you need to draw the line in the paintComponent() method.

```
public class JLabel {
        …
        protected void paintComponent(Graphics g) {
                // draw label text
                g.drawString(…);
                // fill background color, shading, border … etc.
        }
        …
}
```

The paintComponent() Method

- The paintComponent method is defined in JComponent, and its header is as follows:
  - protected void paintComponent(Graphics g)
- This method, defined in the JComponent class, is invoked whenever the component is first displayed or redisplayed.
- The Graphics object g is created automatically by the JVM for every visible GUI component. The JVM obtains the Graphics object and passes it to invoke paintComponent.
- In order to draw things on a component (e.g., a JLabel), you need to declare a class that extends a Swing GUI component class and overrides its paintComponent method to specify what to draw. The previous problem can be solved by rewriting paintComponent.

TestPaintComponent

Drawing Geometric Figures on Panels

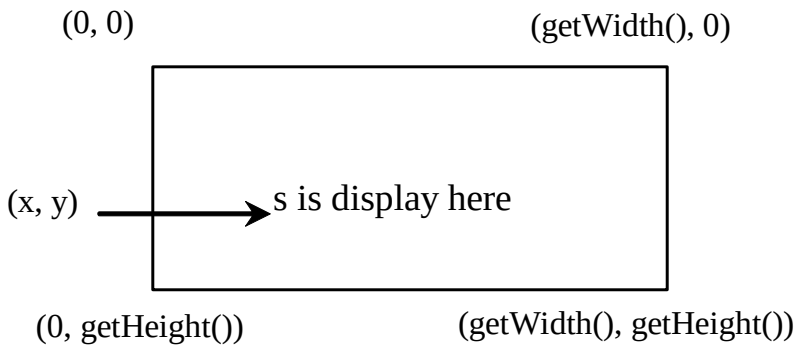
- JPanel can be used to draw graphics (including text) and enable user interaction.
  - Drawing Strings
  - Drawing Lines
  - Drawing Rectangles
  - Drawing Ovals
  - Drawing Arcs
  - Drawing Polygons

- To draw in a panel, you create a new class that extends JPanel and override the paintComponent method to tell the panel how to draw things. You can then display strings, draw geometric shapes, and view images on the panel.

TestPanelDrawing

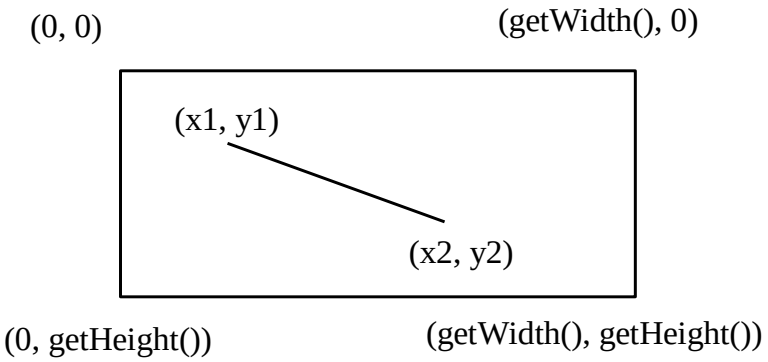
Drawing Strings and Lines

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



The baseline of the leftmost character is at position (*x*, *y*).

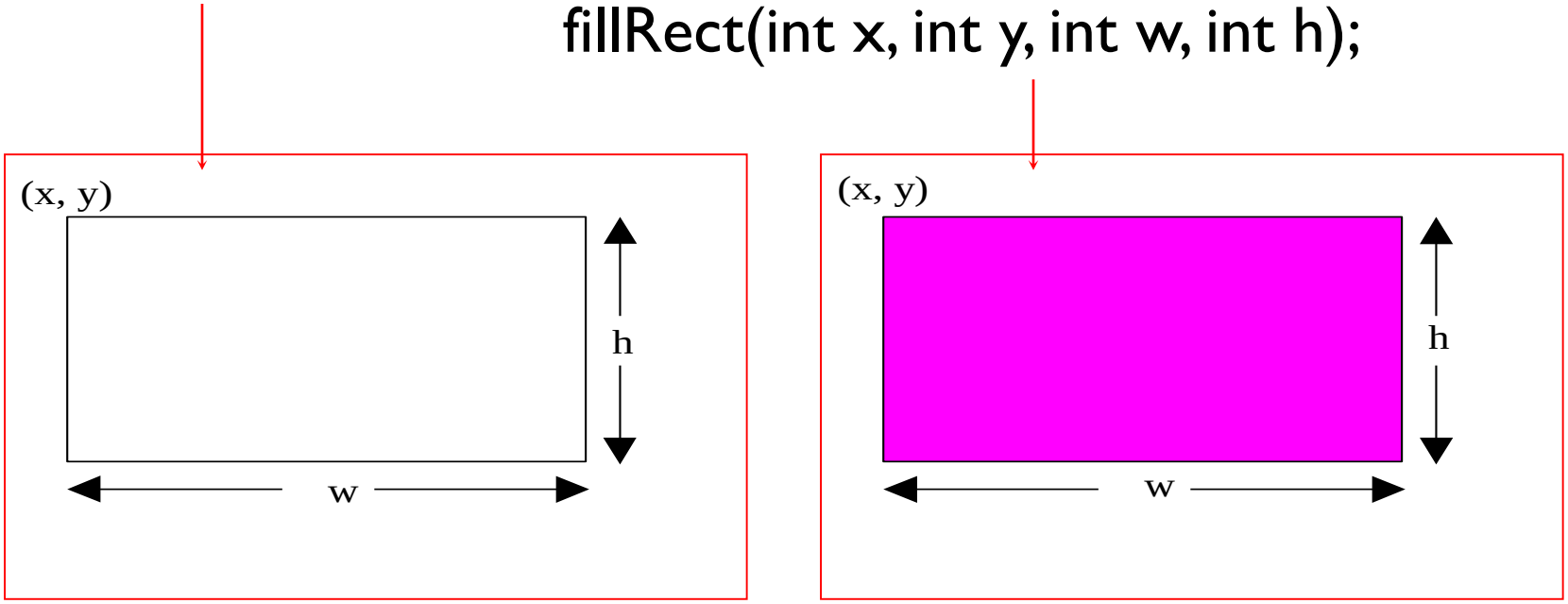
drawLine(int x1, int y1, int x2, int y2);



Drawing Rectangles

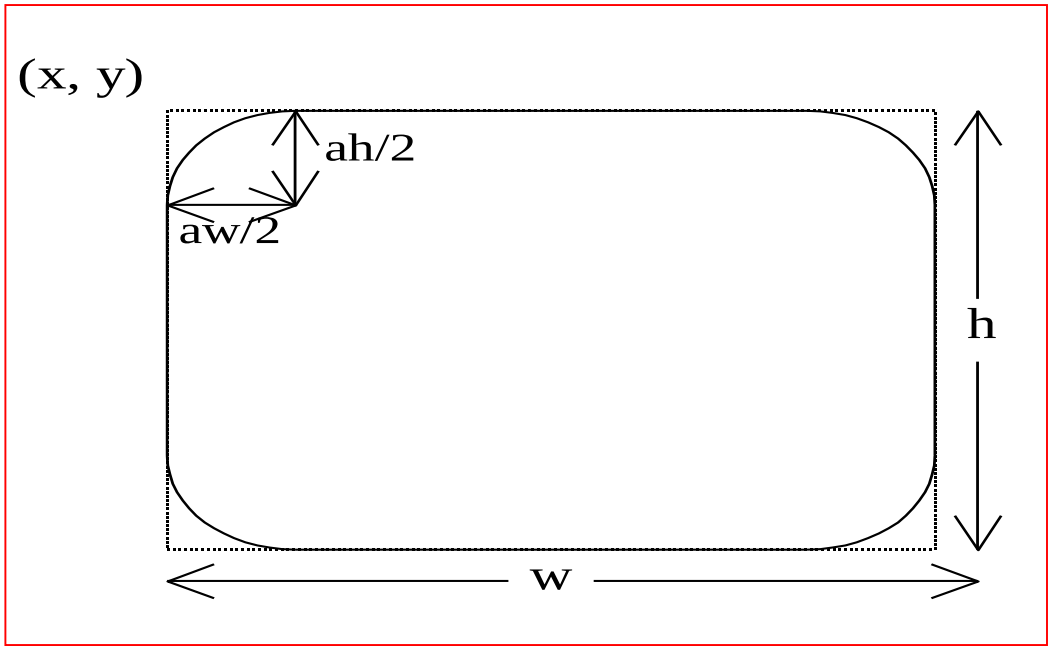
drawRect(int x, int y, int w, int h);

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



Drawing Rounded Rectangles

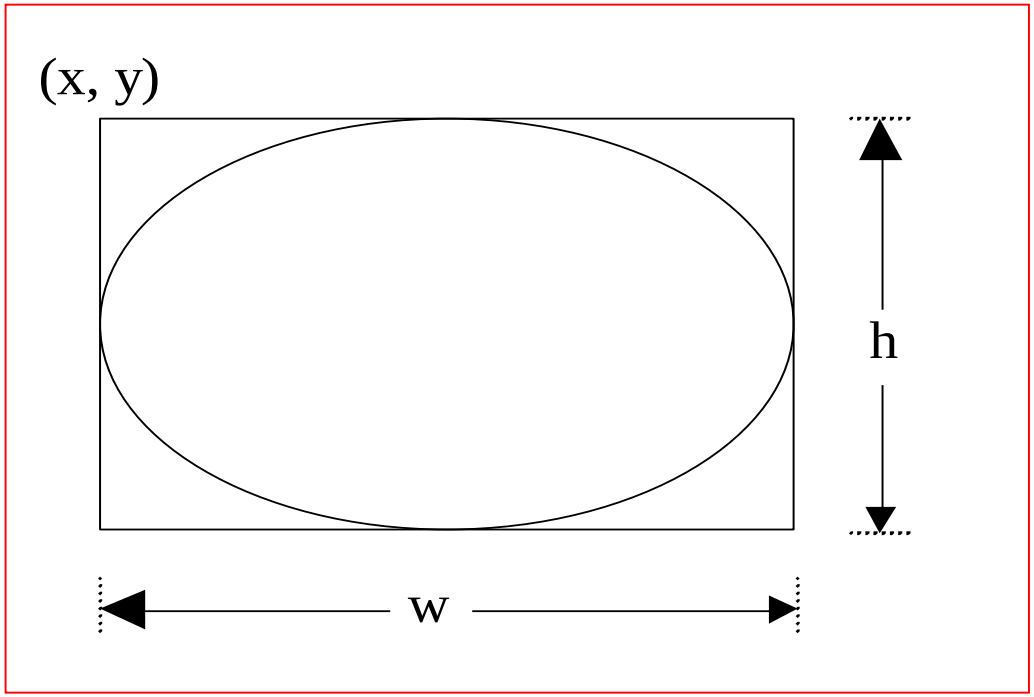
- drawRoundRect(int x, int y, int w, int h, int aw, int ah);
- fillRoundRect(int x, int y, int w, int h, int aw, int ah);



ah = arc height
aw = arc width

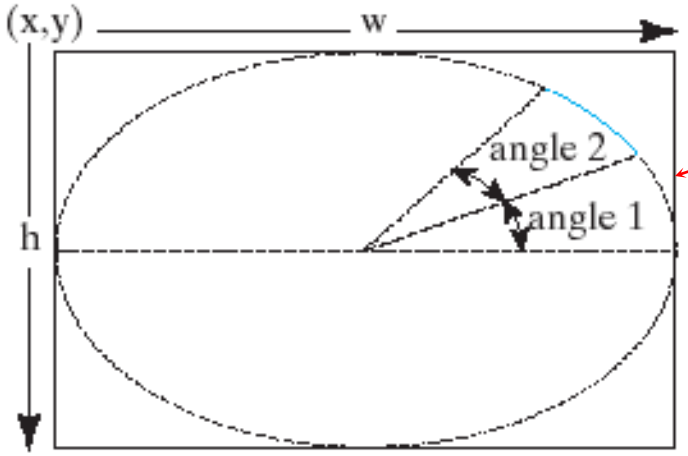
Drawing Ovals

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

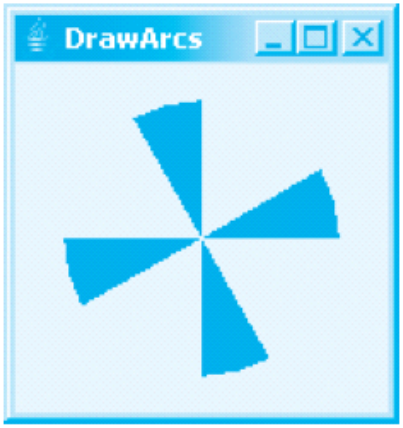


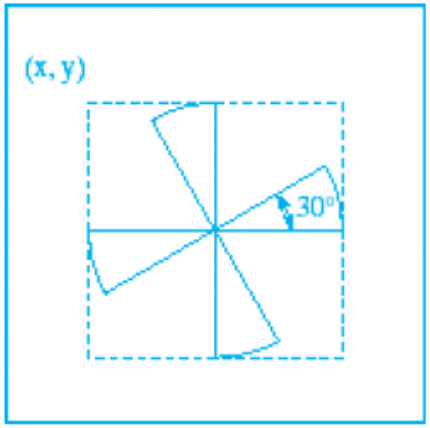
Drawing Arcs

- 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



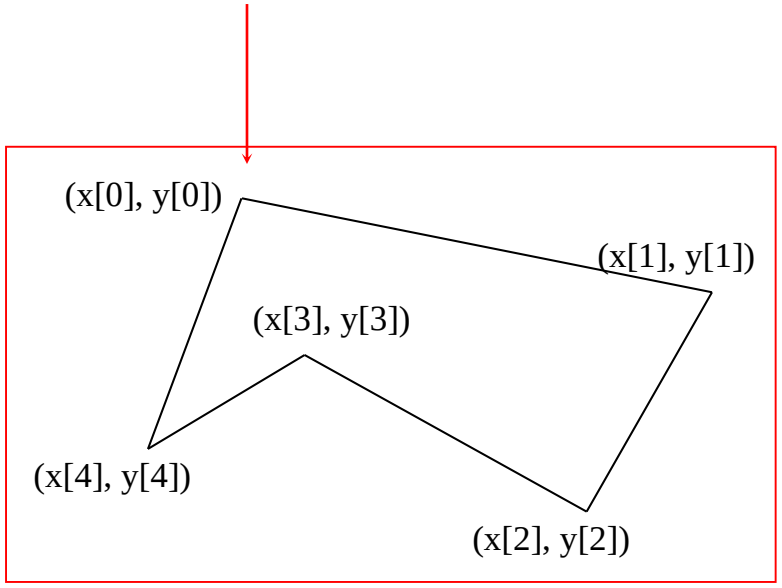


DrawArcs

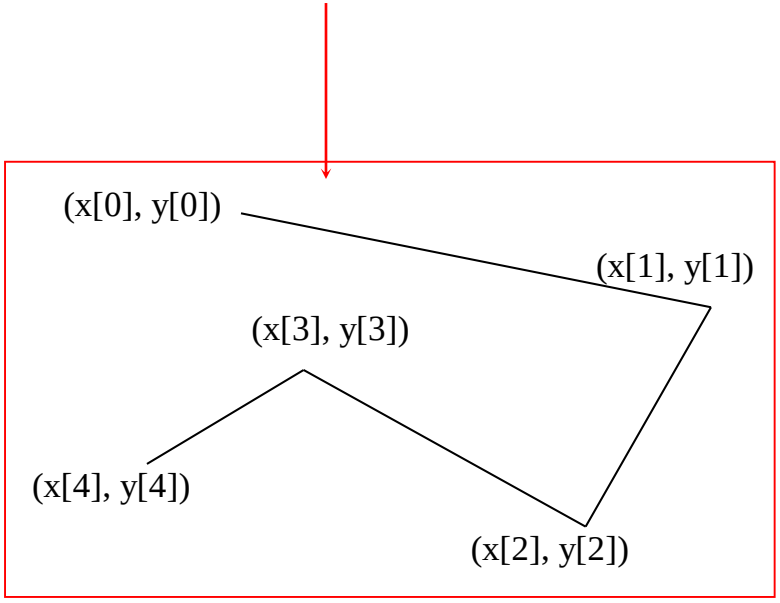
Drawing Polygons and Polylines

- int[] x = {40, 70, 60, 45, 20};
- int[] y = {20, 40, 80, 45, 60};

g.drawPolygon(x, y, x.length);

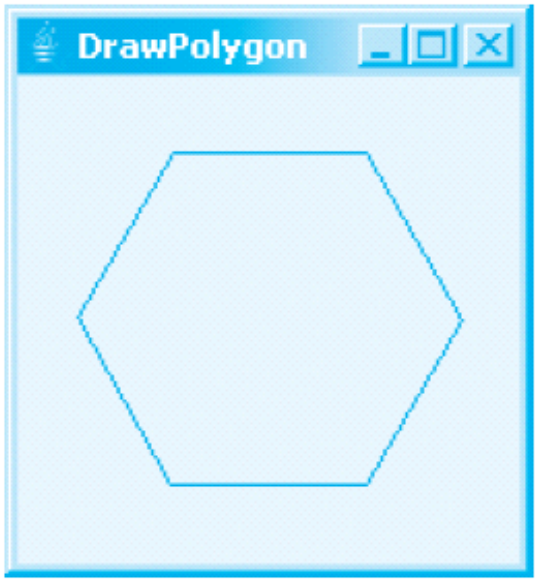


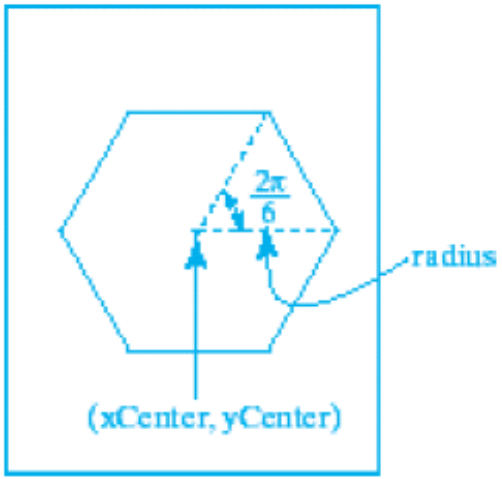
g.drawPolyline(x, y, x.length);



Using the Polygon Class

- You can also use the Polygon class to draw polygons:
  - Polygon polygon = new Polygon();
  - polygon.addPoint(40, 59);
  - polygon.addPoint(40, 100);
  - polygon.addPoint(10, 100);
  - g.drawPolygon(polygon);





DrawPolygon

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 imageIcon = 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 (e.g. JPanel) on which the image is displayed (usually `this`). 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