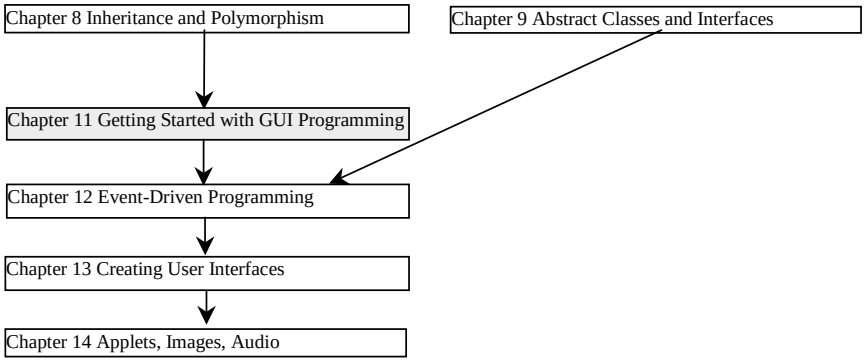
Chapter 11 Getting Started with GUI Programming

Prerequisites for Part III

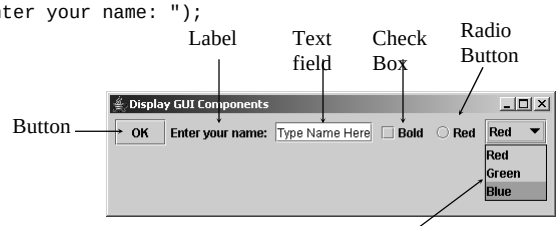


Objectives

- To distinguish simple GUI components (§11.2).
- To describe the Java GUI API hierarchy (§11.3).
- To create user interfaces using frames, panels, and simple UI components (§11.4).
- To understand the role of layout managers (§11.5).
- To use the FlowLayout, GridLayout, and BorderLayout managers to layout components in a container (§11.5).
- To specify colors and fonts using the Color and Font classes (§11.6-11.7).
- To use JPanel as subcontainers (§11.8).
- To paint graphics using the paintComponent method on a panel (§11.9).
- To draw strings, lines, rectangles, ovals, arcs, and polygons using the drawing methods in the Graphics class (§11.9).
- To center display using the FontMetrics Class (§11.10).
- To develop a reusable component MessagePanel to display a message on a panel (§11.11).
- To develop a reusable component StillClock to emulate an analog clock (§11.12 Optional).

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: ");
```



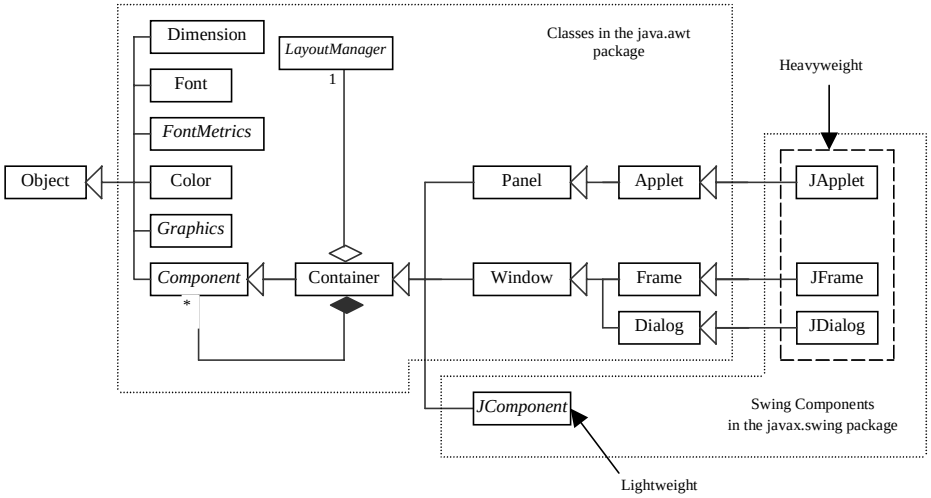

```
// Create a text field with text "Type Name Here"
JTextField jtfName = new JTextField("Type Name Here");

// 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"});
```


Swing vs. AWT

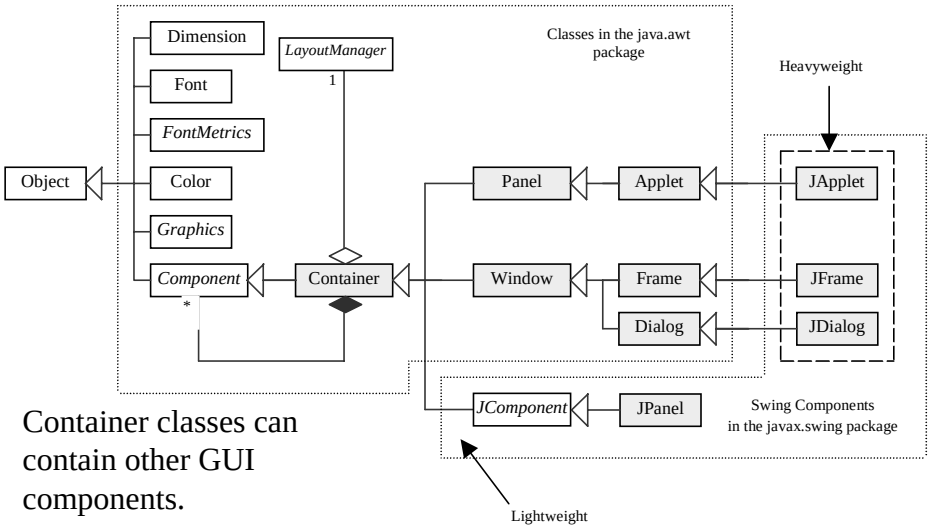
So why do the GUI component classes have a prefix *J*? Instead of JButton, why not name it simply Button? In fact, there is a class already named Button in the java.awt package.

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 Java code, except for components that are 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*.

GUI Class Hierarchy (Swing)

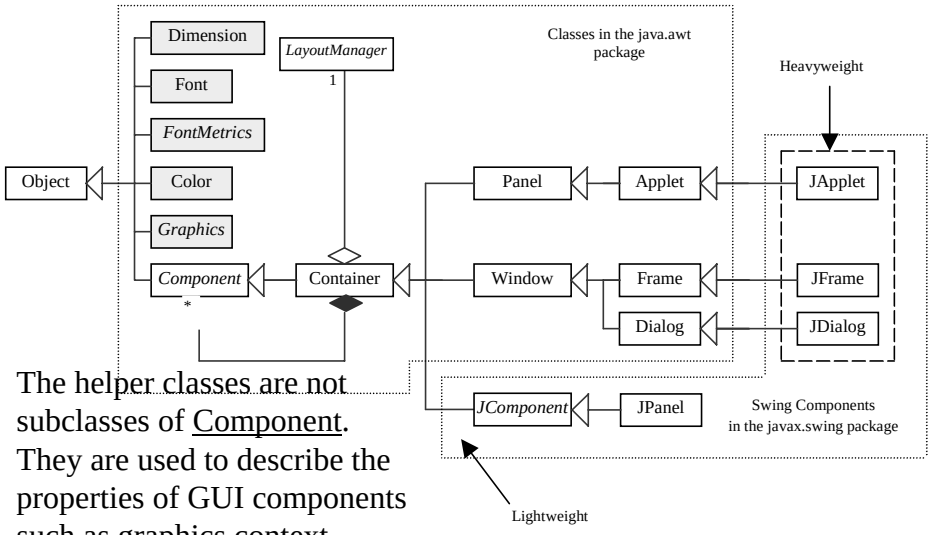


Container Classes



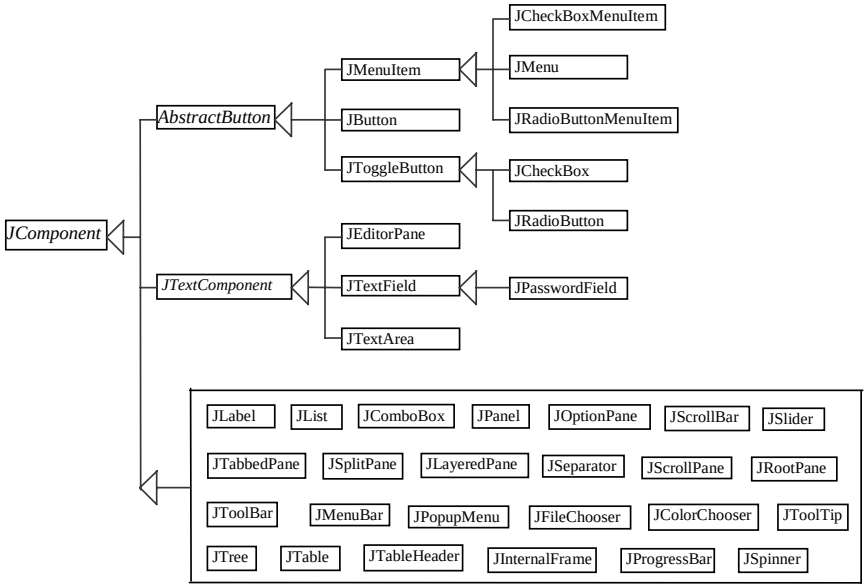
Container classes can contain other GUI components.

GUI Helper Classes

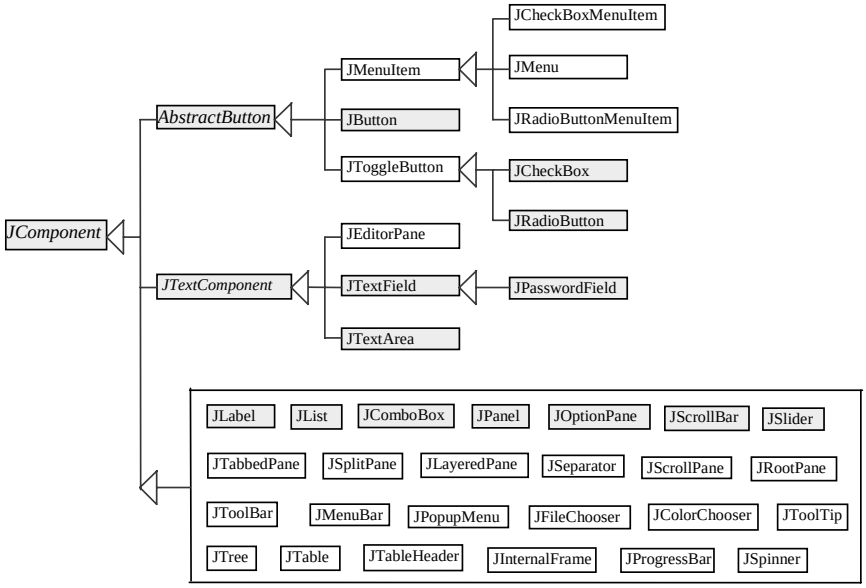


The helper classes are not subclasses of Component. They are used to describe the properties of GUI components such as graphics context, colors, fonts, and dimension.

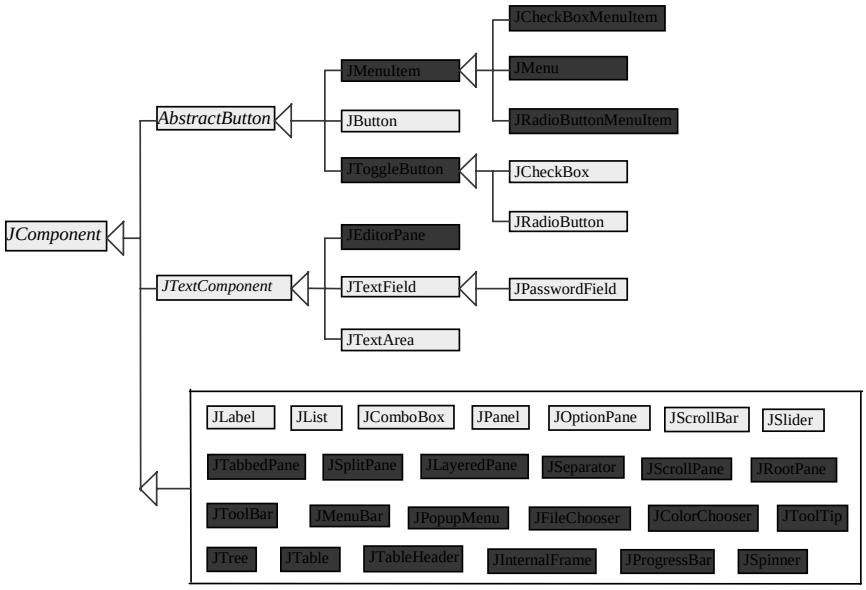
Swing GUI Components



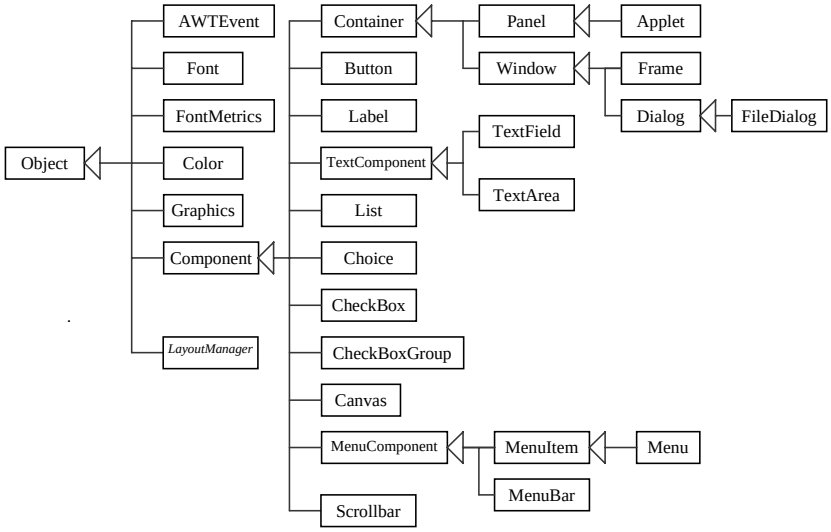
Components Covered in the Custom Core



Components Covered in the Comprehensive Version



AWT (Optional)



Frames

- Frame is a window that is not contained inside another window. Frame is the basis to contain other user interface components in Java GUI applications.
- The Frame class can be used to create windows.
- For Swing GUI programs, use JFrame class to create widows.

Creating Frames

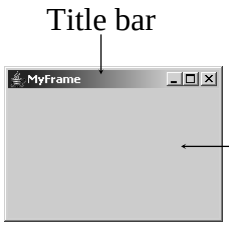
```
import javax.swing.*;
public class MyFrame {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Test Frame");
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
  }
}
```

NOTE: You must have JDK 1.3 or higher to run the slides.

Run

Adding Components into a Frame

```
// Add a button into the frame
frame.getContentPane().add(
  new JButton("OK"));
```



MyFrameWithComponents

Run

NOTE

The content pane is a subclass of Container. The statement in the preceding slide can be replaced by the following two lines:

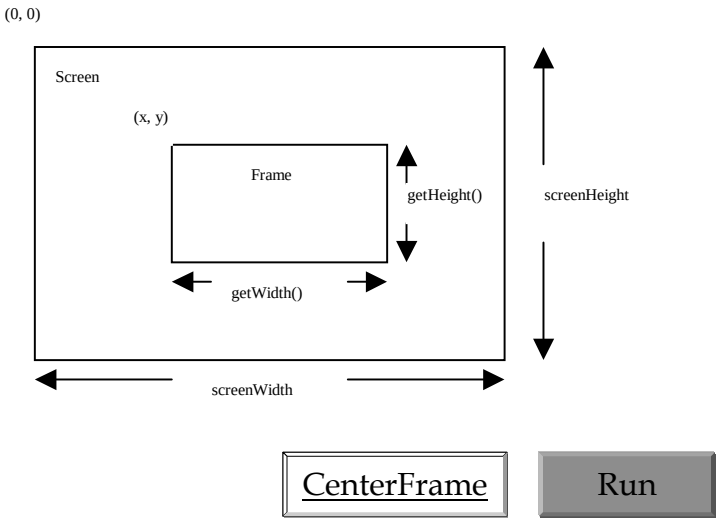
```
Container container = frame.getContentPane();
container.add(new JButton("OK"));
```

You may wonder how a Container object is created. It is created when a JFrame object is created. A JFrame object uses the content pane to hold components in the frame.

Centering Frames

By default, a frame is displayed in the upper-left corner of the screen. To display a frame at a specified location, you can use the setLocation(x, y) method in the JFrame class. This method places the upper-left corner of a frame at location (x, y).

Centering Frames, cont.



Layout Managers

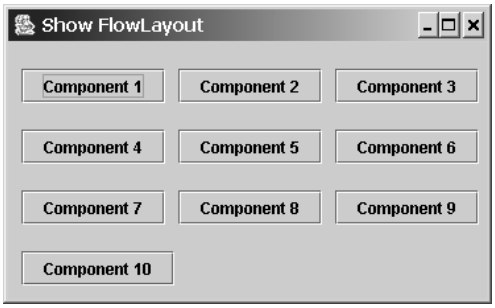
- Java's layout managers provide a level of abstraction to automatically map your user interface on all window systems.
- The UI components are placed in containers. Each container has a layout manager to arrange the UI components within the container.
- Layout managers are set in containers using the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

Kinds of Layout Managers

- FlowLayout (Chapter 11)
- GridLayout (Chapter 11)
- BorderLayout (Chapter 11)
- Several other layout managers will be introduced in Chapter 23 Chapter 23, “Containers, Layout Managers, and Borders”

Example 11.1 Testing the `FlowLayout` Manager

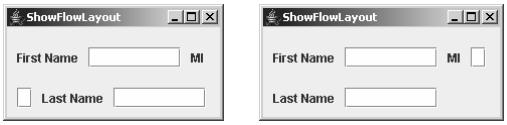
The components are arranged in the container from left to right in the order in which they were added. When one row becomes filled, a new row is started.



ShowFlowLayout Run

Example 11.1 (New) Testing the `FlowLayout` Manager

Write a program that adds three labels and text fields into the content pane of a frame with a FlowLayout manager.



ShowFlowLayout Run

`FlowLayout` Constructors

- `public FlowLayout(int align, int hGap, int vGap)`

  Constructs a new `FlowLayout` with a specified alignment, horizontal gap, and vertical gap. The *gaps* are the distances in pixel between components.

- `public FlowLayout(int alignment)`

  Constructs a new `FlowLayout` with a specified alignment and a default gap of five pixels for both horizontal and vertical.

- `public FlowLayout()`

  Constructs a new `FlowLayout` with a default center alignment and a default gap of five pixels for both horizontal and vertical.

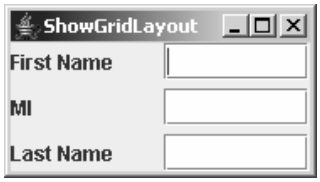
Example 11.2
Testing the `GridLayout` Manager

The `GridLayout` manager arranges components in a grid (matrix) formation with the number of rows and columns defined by the constructor. The components are placed in the grid from left to right starting with the first row, then the second, and so on.

ShowGridLayout | Run

Example 11.2
(New) Testing the `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.



ShowGridLayout | Run

`GridLayout` Constructors

- `public GridLayout(int rows, int columns)`

  Constructs a new `GridLayout` with the specified number of rows and columns.

- `public GridLayout(int rows, int columns, int hGap, int vGap)`

  Constructs a new `GridLayout` with the specified number of rows and columns, along with specified horizontal and vertical gaps between components.

Example 11.3 Testing the `BorderLayout` Manager

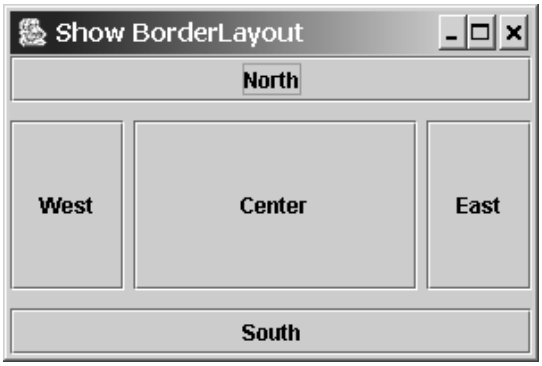
The `BorderLayout` manager divides the container into five areas: East, South, West, North, and Center. Components are added to a `BorderLayout` by using the add method.

`add(Component, constraint)`, where `constraint` is `BorderLayout.EAST`, `BorderLayout.SOUTH`, `BorderLayout.WEST`, `BorderLayout.NORTH`, or `BorderLayout.CENTER`.

ShowBorderLayout

Run

Example 11.3, cont.



ShowBorderLayout | Run

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

Thirteen standard colors (black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow) are defined as constants in java.awt.Color.

The standard color names are constants, but they are named as variables with lowercase for the first word and uppercase for the first letters of subsequent words. Thus the color names violate the Java naming convention. Since JDK 1.4, you can also use the new constants: BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, and YELLOW.

Setting Colors

You can use the following methods to set the component's background and foreground colors:

```
setBackground(Color c)
setForeground(Color c)
```

Example:

```
jbt.setBackground(Color.yellow);
jbt.setForeground(Color.red);
```

The Font Class

Font Names

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

Font Style

Font.PLAIN (0), Font.BOLD (1), Font.ITALIC (2), and Font.BOLD + Font.ITALIC (3)

```
Font myFont = 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);
```


Finding All Available Font Names

```
GraphicsEnvironment e =
  GraphicsEnvironment.getLocalGraphicsEnvironment();
String[] fontnames =
   e.getAvailableFontFamilyNames();
for (int i = 0; i < fontnames.length; i++)
  System.out.println(fontnames[i]);
```

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.
- To add a component to JFrame, you actually add it to the content pane of JFrame. To add a component to a panel, you add it directly to the panel using the add method.

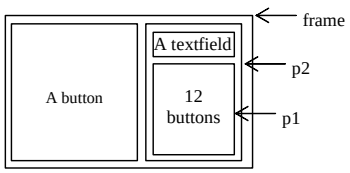
Creating a JPanel

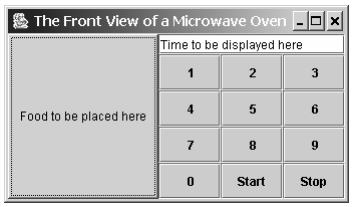
You can use new JPanel() to create a panel with a default FlowLayout manager or new JPanel(LayoutManager) to create a panel with the specified layout manager. Use the add(Component) method to add a component to the panel. For example,

```
JPanel p = new JPanel();
p.add(new JButton("OK"));
```

Example 11.4 Testing Panels

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





Drawing on Panels

- JPanel can be used to draw graphics (including text) and enable user interaction.
- 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.

The paintComponent Method

The paintComponent method is defined in JComponent, and its header is as follows:

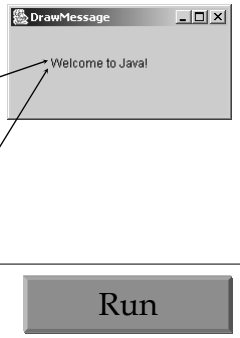
protected void paintComponent(Graphics g)

The Graphics object g is created automatically by the JVM for every visible GUI component. This object controls how information is drawn. You can use various drawing methods defined in the Graphics class to draw strings and geometric figures. For example, you can draw a string using the following method in the Graphics class:

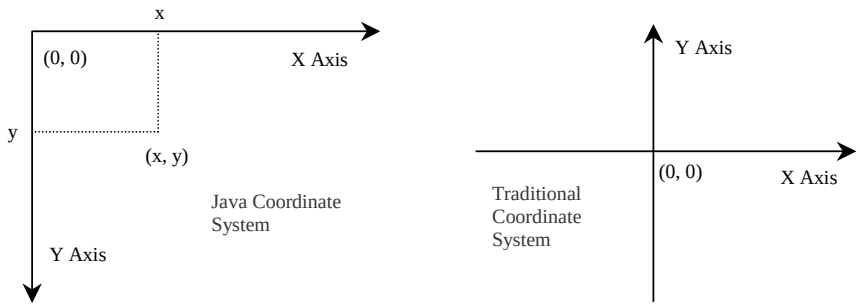
public void drawString(String string, int x, int y)

Drawing on Panels, cont.

```
public class DrawMessage extends JPanel {
  /** Main method */
  public static void main(String[] args) {
    JFrame frame = new JFrame("DrawMessage");
    frame.getContentPane().add(new DrawMessage());
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 200);
    frame.setVisible(true);
  }

  /** Paint the message */
  protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    g.drawString("Welcome to Java!", 40, 40);
  }
}
```



Java Coordinate System



NOTE

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. The Graphics class is implemented on the native platform in the JVM. When you use the paintComponent method to draw things on a graphics context g, this g is an instance of a concrete subclass of the abstract Graphics class for the specific platform. The Graphics class encapsulates the platform details and enables you to draw things uniformly without concerning specific platforms.

NOTE

Whenever a component is displayed, a Graphics object is created for the component. The Swing components use the paintComponent method to draw things. The paintComponent method is automatically invoked to paint the graphics context when the component is first displayed or whenever the component needs to be redisplayed. Invoking super.paintComponent(g) is necessary to ensure that the viewing area is cleared before a new drawing is displayed.

NOTE

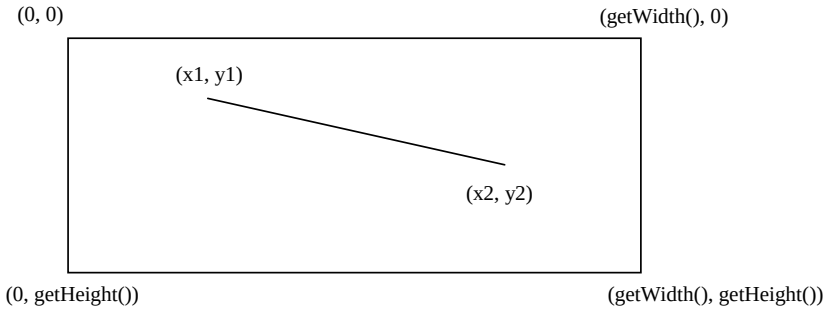
To draw things, normally you create a subclass of JPanel and override its paintComponent method to tell the system how to draw. In fact, you can draw things on any GUI component.

Drawing Geometric Figures

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

Drawing Lines

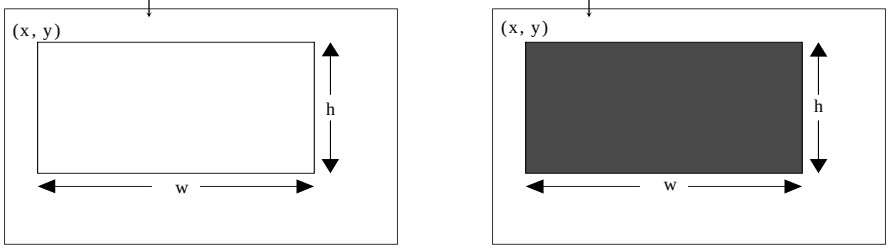
drawLine(int x1, int y1, int x2, int y2);



Drawing Rectangles

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

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

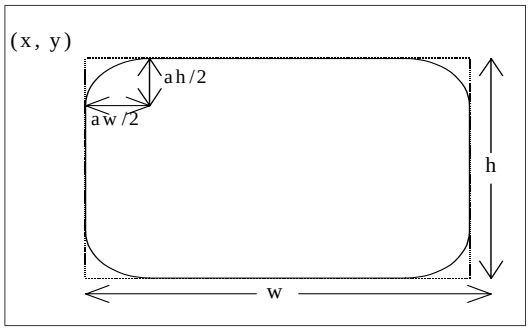


DrawRectangles Run

Drawing Rounded Rectangles

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

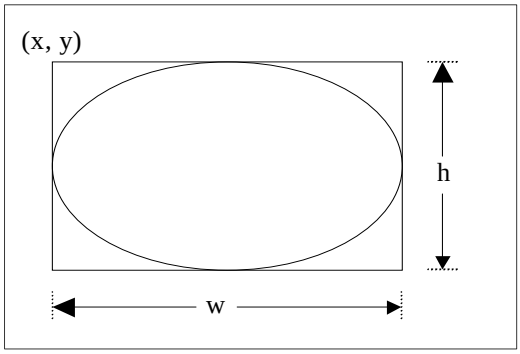
fillRoundRect(int x, int y, int w, int h, int aw, int ah);



Drawing Ovals

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

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

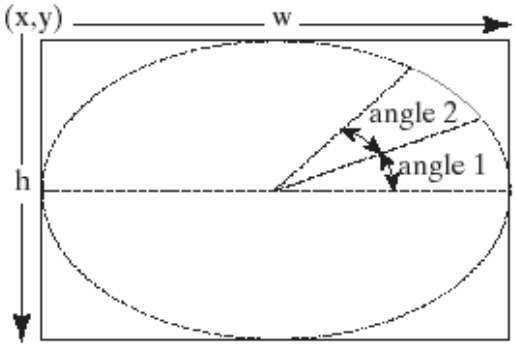


DrawOvals

Run

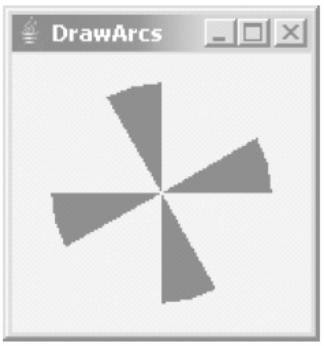
Drawing Arcs

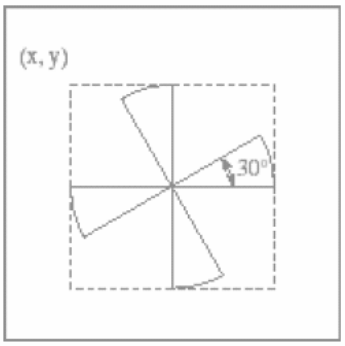
drawArc(int x, int y, int w, int h, int angle1, int angle2);
fillArc(int x, int y, int w, int h, int angle1, int angle2);



Angles are in degree

Drawing Arcs Example



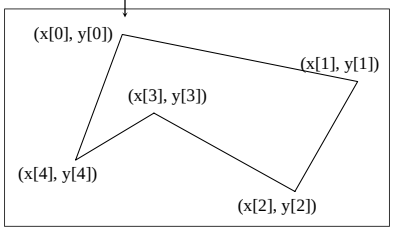


DrawArcs

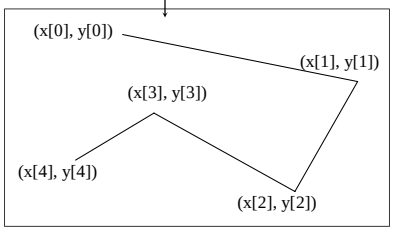
Run

Drawing Polygons and Polylines

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



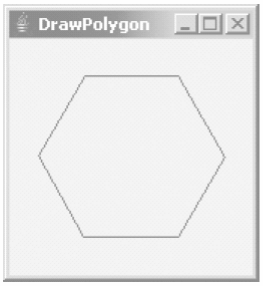

```
g.drawPolyline(x, y, x.length);
```

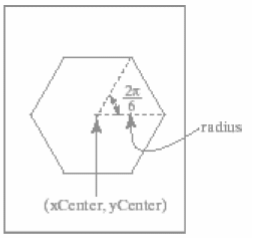


Drawing Polygons Using the Polygon Class

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


Drawing Polygons Example





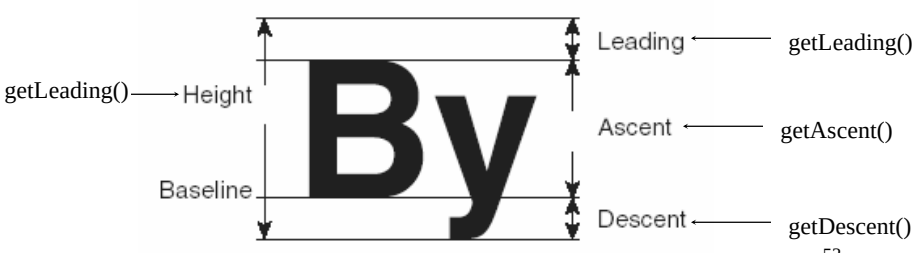
DrawPolygon

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

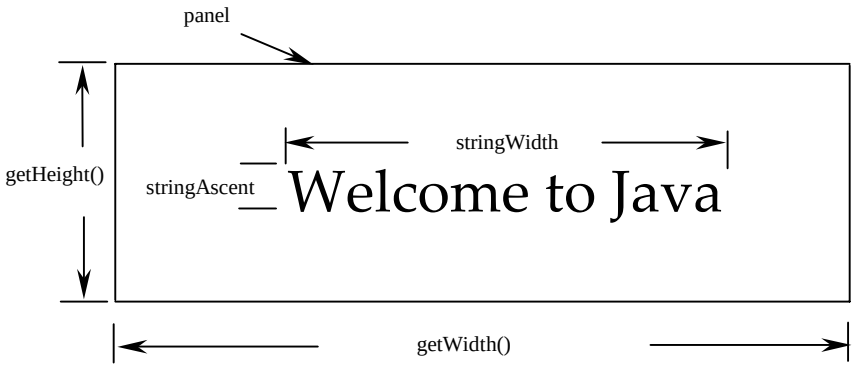
FontMetrics is an abstract class. To get a FontMetrics object for a specific font, use the following getFontMetrics methods defined in the Graphics class:

- public FontMetrics getFontMetrics(Font f)

Returns the font metrics of the specified font.

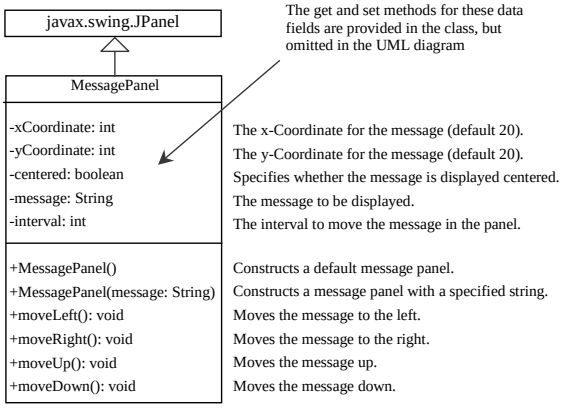
- public FontMetrics getFontMetrics()

Returns the font metrics of the current font.



CenterMessage Run

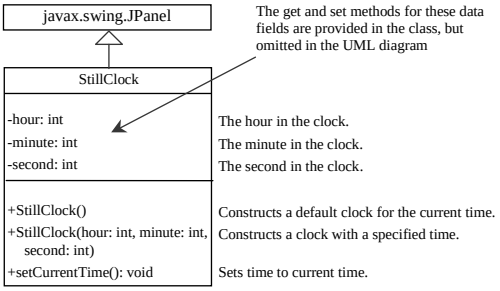
Case Study I

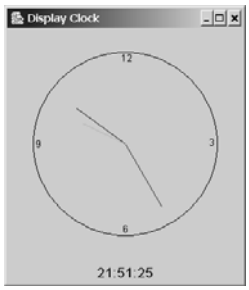


This case study develops a useful class that displays a message in a panel. The class enables the user to set the location of the message, center the message, and move the message with the specified interval.

MessagePanel TestMessagePanel Run

Case Study II





StillClock | DisplayClock | Run

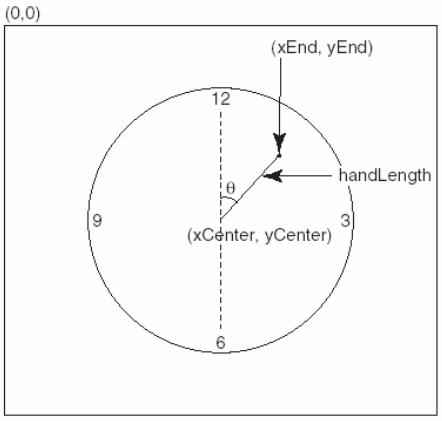
Drawing Clock

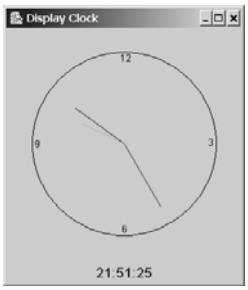
$xEnd = xCenter + handLength \times \sin(\theta)$

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

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

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



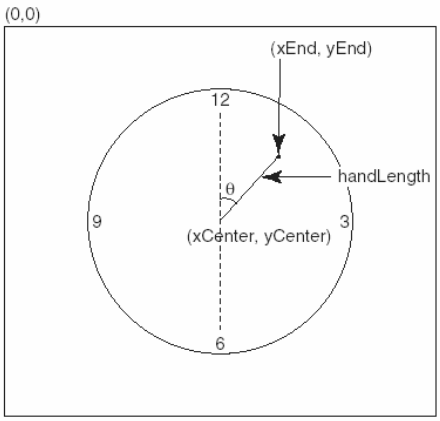


Drawing Clock, cont.

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

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

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



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 $(hour + minute/60 + second/(60 \times 60))) \times (2\pi/12)$