Creating GUI Objects

To develop reusable components (§11.11).
To distinguish simple GUI components (§11.2).
To describe the Java GUI API hierarchy (§11.13).
To specify colors and fonts using the Color and Font classes (§11.6-11.7).
To specify colors and fonts using the Color and Font classes (§11.6).

Creating GUI Objects

To develop a reusable component StillClock to emulate an analog clock (§11.12 Optional).
To develop a reusable component MessagePanel to display a message on a panel (§11.11).

Swing vs. AWT

Swing vs. AWT

Creating GUI Objects

Objectives

Swing vs. AWT

- To distinguish simple GUI components (§11.2).
- To describe the Java GUI API hierarchy (§11.13).
- To use the FlowLayout, GridBagLayout, and BorderLayout managers to layout components in a container (§11.5).
- To use the FlowLayout, GridBagLayout, and BorderLayout managers to layout components in a container (§11.15).
- To use JPanel as subcontainers (§11.8).
- To use 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 distinguish simple GUI components (§11.2).
- To develop a reusable component StillClock to emulate an analog clock (§11.12 Optional).

Swing vs. AWT

Creating GUI Objects

Objectives

Swing vs. AWT

- To distinguish simple GUI components (§11.2).
- To describe the Java GUI API hierarchy (§11.13).
- To use the FlowLayout, GridBagLayout, and BorderLayout managers to layout components in a container (§11.5).
- To use JPanel as subcontainers (§11.8).
- To use 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).

Swing vs. AWT

Creating GUI Objects

Objectives

Swing vs. AWT

- To distinguish simple GUI components (§11.2).
- To describe the Java GUI API hierarchy (§11.13).
- To use the FlowLayout, GridBagLayout, and BorderLayout managers to layout components in a container (§11.5).
- To use JPanel as subcontainers (§11.8).
- To use 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).

Swing vs. AWT

Creating GUI Objects

Objectives

Swing vs. AWT

- To distinguish simple GUI components (§11.2).
- To describe the Java GUI API hierarchy (§11.13).
- To use the FlowLayout, GridBagLayout, and BorderLayout managers to layout components in a container (§11.5).
- To use JPanel as subcontainers (§11.8).
- To use 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).

Swing vs. AWT

Creating GUI Objects

Objectives

Swing vs. AWT

- To distinguish simple GUI components (§11.2).
- To describe the Java GUI API hierarchy (§11.13).
- To use the FlowLayout, GridBagLayout, and BorderLayout managers to layout components in a container (§11.5).
- To use JPanel as subcontainers (§11.8).
- To use 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).

Swing vs. AWT

Creating GUI Objects

Objectives

Swing vs. AWT

- To distinguish simple GUI components (§11.2).
- To describe the Java GUI API hierarchy (§11.13).
- To use the FlowLayout, GridBagLayout, and BorderLayout managers to layout components in a container (§11.5).
- To use JPanel as subcontainers (§11.8).
- To use 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).
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

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 windows.
Creating Frames

```java
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.

Adding Components into a Frame

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

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.

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, “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.

Example 11.2
Testing the GridLayout Manager

Several other layout managers will be introduced in Chapter 23, “Containers, Layout Managers, and Borders.”

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.

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
(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.
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.

Example 11.3, cont.

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

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:

setBackgroundColor(Color c)
setForegroundColor(Color c)

Example:
jbt.setBackgroundColor(Color.YELLOW);
jbt.setForegroundColor(Color.RED);
The Font Class

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.

Example 11.4 Testing Panels

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

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"));

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:

```java
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:

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

### Drawing on Panels, cont.

```java
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);
    }
}
```

Run (40, 40)

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);

(0, 0)  (getWidth(), 0)

(x1, y1)  (x2, y2)

(0, getHeight())  (getWidth(), getHeight())

Drawing Rectangles

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

(x, y)  (x + w, y)

(w, h)  (x, y + 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);

(x, y)  (x + w, y)

(w/2, h/2)  (x, y + h)

Angles are in degree

Drawing Ovals

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

(x, y)  (x + w, y)

(h, w)  (x, y + 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
Drawing Polygons and Polylines

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

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:

- `getAscent()`
- `getDescent()`
- `getLeading()`
- `getHeight()`
- `stringWidth(String str)`

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.
Welcome to Java

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.

Case Study II

Drawing Clock

xEnd = xCenter + handLength × sin(θ)
yEnd = yCenter - handLength × cos(θ)

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 × sin(θ)
yEnd = yCenter - handLength × cos(θ)

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