Session 04
Review: GUI - Graphic, Frame, Events

Adapted from D. Liang's "Introduction to Java Programming 8th Ed."
Creating GUI Objects

```java
// 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
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
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 JFrame 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);
    }
}
```
Adding Components into a Frame

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

JFrame Class

```java
javax.swing.JFrame
+JFrame()
+JFrame(title: String)
+setSize(width: int, height: int): void
+setLocation(x: int, y: int): void
+setVisible(visible: boolean): void
+setDefaultCloseOperation(mode: int): void
+setLocationRelativeTo(c: Component): void
+pack(): void
```

- Creates a default frame with no title.
- Creates a frame with the specified title.
- Specifies the size of the frame.
- Specifies the upper-left corner location of the frame.
- Sets true to display the frame.
- Specifies the operation when the frame is closed.
- Sets the location of the frame relative to the specified component.
  - If the component is null, the frame is centered on the screen.
- Automatically sets the frame size to hold the components in the frame.
Java 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
setColor(color: Color): void
setFont(font: Font): void
drawString(s: String, x: int, y: int): void
drawLine(x1: int, y1: int, x2: int, y2: int): void
drawRect(x: int, y: int, w: int, h: int): void
fillRect(x: int, y: int, w: int, h: int): void
drawRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int): void
fillRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int): void
draw3DRect(x: int, y: int, w: int, h: int, raised: boolean): void
fill3DRect(x: int, y: int, w: int, h: int, raised: boolean): void
drawOval(x: int, y: int, w: int, h: int): void
fillOval(x: int, y: int, w: int, h: int): void
drawArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void
fillArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void
drawPolygon(xPoints: int[], yPoints: int[], nPoints: int): void
fillPolygon(xPoints: int[], yPoints: int[], nPoints: int): void
drawPolygon(g: Polygon): void
fillPolygon(g: Polygon): void
drawPolyline(xPoints: int[], yPoints: int[], nPoints: int): void
```

Sets a new color for subsequent drawings.
Sets a new font for subsequent drawings.
Draws a string starting at point (x, y).
Draws a line from (x1, y1) to (x2, y2).
Draws a rectangle with specified upper-left corner point at (x, y) and width w and height h.
Draws a filled rectangle with specified upper-left corner point at (x, y) and width w and height h.
Draws a rounded-corner rectangle with specified arc width aw and arc height ah.
Draws a filled rounded-corner rectangle with specified arc width aw and arc height ah.
Draws a 3-D rectangle raised above the surface or sunk into the surface.
Draws a filled 3-D rectangle raised above the surface or sunk into the surface.
Draws an oval bounded by the rectangle specified by the parameters x, y, w, and h.
Draws a filled oval bounded by the rectangle specified by the parameters x, y, w, and h.
Draws an arc conceived as part of an oval bounded by the rectangle specified by the parameters x, y, w, and h.
Draws a filled arc conceived as part of an oval bounded by the rectangle specified by the parameters x, y, w, and h.
Draws a closed polygon defined by arrays of x and y coordinates. Each pair of (x[i], y[i]) coordinates is a point.
Draws a filled polygon defined by arrays of x and y coordinates. Each pair of (x[i], y[i]) coordinates is a point.
Draws a closed polygon defined by a Polygon object.
Draws a filled polygon defined by a Polygon object.

**Drawing Strings**

```java
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);

### 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**

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

**Drawing Polygons Using the Polygon Class**

```java
Polygon polygon = new Polygon();
polygon.addPoint(40, 59);
polygon.addPoint(40, 100);
polygon.addPoint(10, 100);
polygon.addPoint(10, 100);
g.drawPolyline(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()`: The height of the ascent, which is the distance from the baseline to the top of the tallest character in the string.
- `getAscent()`: The height of the descent, which is the distance from the baseline to the bottom of the lowest character in the string.
- `getLeading()`: The leading, which is the distance between the baseline and the baseline of the next line of text.
- `getHeight()`: The height of the string.
- `getWidth()`: The width of the string.

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, x: int, y: int, bgcolor: Color, observer: ImageObserver): void
  drawImage(Image, x: int, y: int, 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.

Same as the preceding method except that it does not specify a background color.

Draws a scaled version of the image that can fill all of the available space in the specified rectangle.

Same as the preceding method except that it provides a solid background color behind the image being drawn.
Procedural vs. Event-Driven Programming

- *Procedural programming* is executed in procedural order
- In *event-driven programming*, code is executed upon activation of events

Events

- An *event* can be defined as a type of signal to the program that something has happened
- The event is generated by external user actions such as mouse movements, mouse clicks, and keystrokes, or by the operating system, such as a timer
Event Classes

- AWTEvent
- EventObject
  - AdjustmentEvent
  - ComponentEvent
  - TextEvent
  - ItemEvent
  - ActionEvent
  - InputEvent
  - WindowEvent
  - MouseEvent
  - KeyEvent
  - ContainerEvent
  - FocusEvent
  - InputEvent
  - PaintEvent
  - WindowEvent
  - ListSelectionEvent
  - ChangeEvent

Event Information

- An event object contains whatever properties are pertinent to the event
- You can identify the source object of the event using the `getSource()` instance method in the `EventObject` class
- The subclasses of `EventObject` deal with special types of events, such as button actions, window events, component events, mouse movements, and keystrokes
- Table 15.1 (of the text book) lists external user actions, source objects, and event types generated
### Selected User Actions

<table>
<thead>
<tr>
<th>User Action</th>
<th>Source Object</th>
<th>Event Type Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click a button</td>
<td>JButton</td>
<td>ActionEvent</td>
</tr>
<tr>
<td>Click a check box</td>
<td>JCheckBox</td>
<td>ItemEvent, ActionEvent</td>
</tr>
<tr>
<td>Click a radio button</td>
<td>JRadioButton</td>
<td>ItemEvent, ActionEvent</td>
</tr>
<tr>
<td>Press return on a text field</td>
<td>JTextField</td>
<td>ActionEvent</td>
</tr>
<tr>
<td>Select a new item</td>
<td>JComboBox</td>
<td>ItemEvent, ActionEvent</td>
</tr>
<tr>
<td>Window opened, closed, etc.</td>
<td>Window</td>
<td>WindowEvent</td>
</tr>
<tr>
<td>Mouse pressed, released, etc.</td>
<td>Component</td>
<td>MouseEvent</td>
</tr>
<tr>
<td>Key released, pressed, etc.</td>
<td>Component</td>
<td>KeyEvent</td>
</tr>
</tbody>
</table>

### The Delegation Model

(a) A generic source component with a generic listener

```
source: SourceClass
+addXListener(listener: XListener)

XListener
+handler(event: XEvent)
```

Register by invoking `source.addXListener(listener)`.

(b) A JButton source component with an ActionListener

```
source: JButton
+addActionListener(listener: ActionListener)

ActionListener
+actionPerformed(event: ActionEvent)
```

Register by invoking `source.addActionListener(listener)`.
Internal Function of a Source Component

source: SourceClass
+addXListener(XListener listener)

An event is triggered

Invoke
listener1.handler(event)
listener2.handler(event)
...
listeners.handler(event)

Keep it a list

(a) Internal function of a generic source object

source: JButton
+addActionListener(ActionListener listener)

An event is triggered

Invoke
listener1.actionPerformed(event)
listener2.actionPerformed(event)
...
listeners.actionPerformed(event)

Keep it a list

(b) Internal function of a JButton object

The Delegation Model: Example

```java
JButton jbt = new JButton("OK");
ActionListener listener = new OKListener();
jbt.addActionListener(listener);
```
### Selected Event Handlers

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Listener Interface</th>
<th>Listener Methods (Handlers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionEvent</td>
<td>ActionListener</td>
<td>actionPerformed(ActionEvent)</td>
</tr>
<tr>
<td>ItemEvent</td>
<td>ItemListener</td>
<td>itemStateChanged(ItemEvent)</td>
</tr>
<tr>
<td>WindowEvent</td>
<td>WindowListener</td>
<td>windowClosing(WindowEvent), windowOpened(WindowEvent),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowIconified(WindowEvent), windowDeiconified(WindowEvent),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowActivated(WindowEvent), windowDeactivated(WindowEvent),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowOpened(WindowEvent), windowIconified(WindowEvent),</td>
</tr>
<tr>
<td>ContainerEvent</td>
<td>ContainerListener</td>
<td>componentAdded(ContainerEvent), componentRemoved(ContainerEvent)</td>
</tr>
<tr>
<td>MouseEvent</td>
<td>MouseListener</td>
<td>mousePressed(MouseEvent), mouseReleased(MouseEvent),</td>
</tr>
<tr>
<td>KeyEvent</td>
<td>KeyListener</td>
<td>keyPressed(KeyEvent), keyReleased(KeyEvent),</td>
</tr>
</tbody>
</table>

#### java.awt.event.ActionEvent

- **java.util.EventObject**
  - `+getSource(): Object`
    - Returns the object on which the event initially occurred.

- **java.awt.event.AWTEvent**
  - **java.awt.event.ActionEvent**
    - `+getActionCommand(): String`
      - Returns the command string associated with this action. For a button, its text is the command string.
    - `+getModifiers(): int`
    - `+getWhen(): long`
      - Returns the timestamp when this event occurred. The time is the number of milliseconds since January 1, 1970, 00:00:00 GMT.