1.00 Lecture 17

Introduction to Swing

Reading for next time: Big Java: sections 9.7-9.11

Online hyperlinked Swing tutorial:
http://download.oracle.com/javase/tutorial/uiswing/

Swing

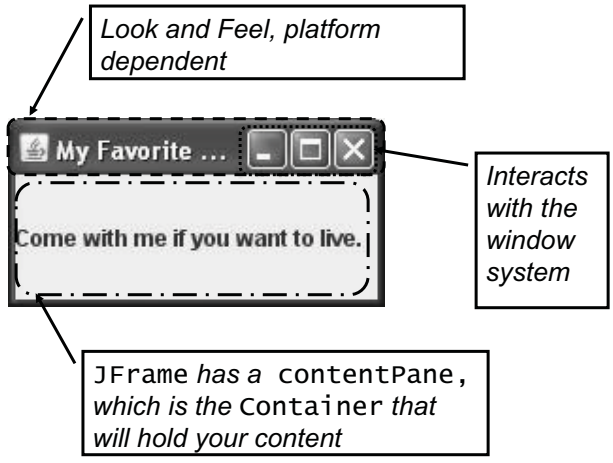
- **Java package of user interface classes for windows, menus, scroll bars, buttons, drawing...**
- **Independent of hardware and operating system (as long as they can paint a window)**
  - **Swing gains independence but loses performance by not relying on native drawing calls**
  - **Has Windows, Mac, other look and feel options**
- **Supersedes Java Abstract Window Toolkit (AWT) though it still uses many non-drawing classes from that package. You will usually:**

```
import java.awt.*;
import javax.swing.*;
```

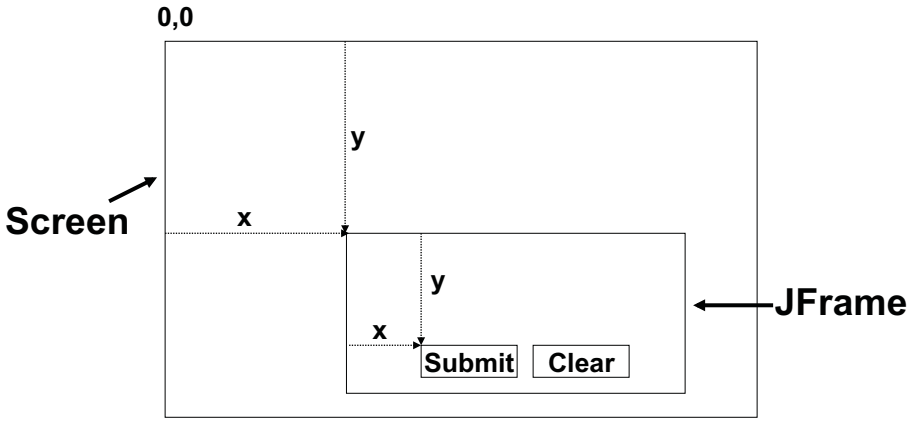
The 3 Flavors of GUI Objects

- **Top Level Windows:**
  - **Containers that are not contained by any other containers**
  - **They can be iconified or dragged, and interact with the native windowing system**
  - **Example: `JFrame`, `JDialog`**
- **`JComponents`: present information or interact with the user**
  - **Examples: labels (`JLabel`), buttons (`JButton`), text fields (`JTextField`)**
  - **`JFrame` and `JDialog` are not `JComponents`**
- **Containers:**
  - **Some `JComponents` are designed to hold other components, not to present info or interact with the user**
  - **Examples: `JPanel` , `JScrollPane`, `Container`**

Anatomy of a `JFrame`



Coordinates



Measured in pixels (e.g. 640 by 480, 1024 by 768, etc.)
By tradition, upper left hand corner is origin (0,0)
X axis goes from left to right, y from top to bottom

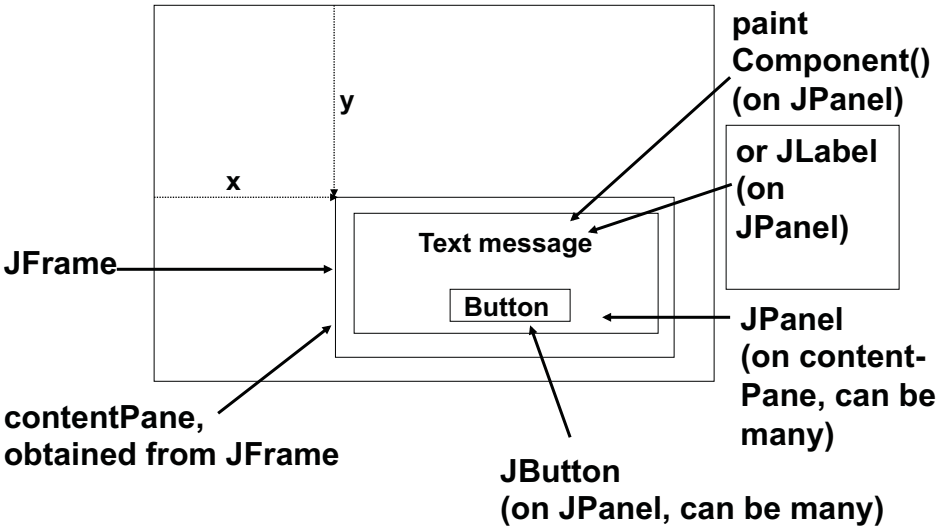
Exercise 1: Empty JFrame

```
// Download, read and run this program
import javax.swing.*;

public class SwingTest {
    public static void main(String[] args) {
        // Create new frame
        JFrame frame= new JFrame();
        // Tells program to exit when user closes this frame
         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         // Frame has 0 default size; give it a size
         frame.setSize(500, 400);      // setSize(int x, int y)
         // Frame invisible by default; make it visible
         frame.setVisible(true);
    }
    // main() ends but Swing "thread" stays alive
}

// Run the program; see what it draws
```

Frames, Panes and Panels



Color

- **Swing has:**
  - **13 predefined colors:** `Color.x` **where x is**
    `orange, pink, cyan, magenta, yellow, black, blue, white, gray, lightGray, darkGray, red, green`
  - **We can create our own colors**
    `Color ugly= new Color(30, 90, 120);`
  - **This uses red-green-blue (RGB) values 0-255**
  - **Color has multiple constructors (see Javadoc)**

Exercise 2: Panel with Color

```
// Continue to write SwingTest
import java.awt.*;            // 1. Import AWT
import javax.swing.*;

public class SwingTest {
    public static void main(String args[]) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500,400);
        Container contentPane= frame.getContentPane(); // No J
        // 2. Create new JPanel object with default constructor
        // 3. Call its setBackground() method with Color yellow
        // 4. Use Container method add() to add panel to
        // contentPane. Panel is 1st argument;
        // BorderLayout.CENTER is 2nd argument
        frame.setVisible(true);
    }
}
```

How to Do Custom Drawing

- **Standard Swing components like `JPanel` and `JButton` use `paintComponent()` to draw themselves.**
- **If you want to do custom drawing, extend a container class, usually `JPanel`, and override `paintComponent()`**
  - **Don't confuse `paintComponent()` with `paintComponents()` (Note the extra "s".)**
- **Use calls from the 2D API in `paintComponent()` to draw what you want on the `JPanel` background.**

Custom Drawing, cont.

- To draw on a JPanel, use inheritance:
  - Create a subclass of `JPanel` or other component to do what you want
  - Redefine (override) the `paintComponent` method in your subclass (Not `paintComponents`)
  - `paintComponent()` has a `Graphics` object as argument
  - `Graphics` object stores data on fonts and colors, and has drawing methods that you can use
  - Add an object of your subclass to the content pane
- Java Graphics class can draw lines, ellipses…
  - Very limited: single thickness, no rotation, etc.
  - Java's Graphics2D class is much more functional
  - Swing draws all components using Java classes and methods in the packages `java.awt.*` and `java.awt.geom.*`.

Exercise 3: AreaPanel

```
// 1. Write initial version of class AreaPanel:
import java.awt.*;
import javax.swing.*;
public class AreaPanel extends JPanel {
    public void paintComponent(Graphics g) {
        // Have JPanel paintComponent do default operations
        // such as background color, etc.
        super.paintComponent(g);
        Graphics2D g2= (Graphics2D) g;
        g2.drawString("Area of rectangle", 125, 150);
        // The last two arguments of drawString indicate
        // that the message should be drawn starting at
        // (x,y)= (125,150)
    }
}
// 2. Modify SwingTest main(), and run it:
//     Change JPanel panel= new JPanel();
//     To AreaPanel panel= new AreaPanel();
//     Move setBackground() to the AreaPanel paintComponent()
```

2D Shapes

- `Shape` is an interface defined in java.awt, but the classes that implement `Shape` are all defined in `java.awt.geom`.
- Shapes all come in two versions, one with high precision coordinates and one with low, e.g.:

```
Ellipse2D.Double  // high precision
Ellipse2D.Float   // low precision
```

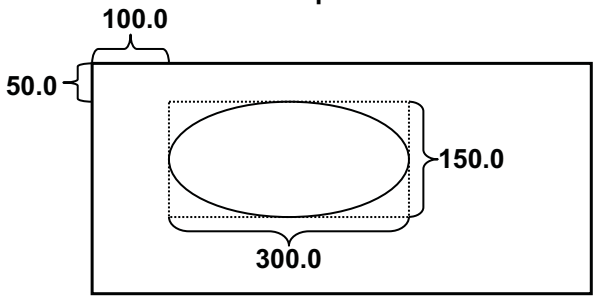
- Each shape has different constructor arguments, doubles or floats depending on whether they are high precision or low.

Creating an Ellipse

- To create an ellipse in `paintComponent()` use the `Ellipse2D.Double` class in `java.awt.geom`:

```
Shape ellipse= new Ellipse2D.Double( double x,
  double y, double width, double height );
```

- x and y define the upper left of the bounding box, width and height the aspect ratio and dimensions of the ellipse.
- Tell the `Graphics2D` object `g2` to `draw()` the ellipse with the position and dimensions pictured below:

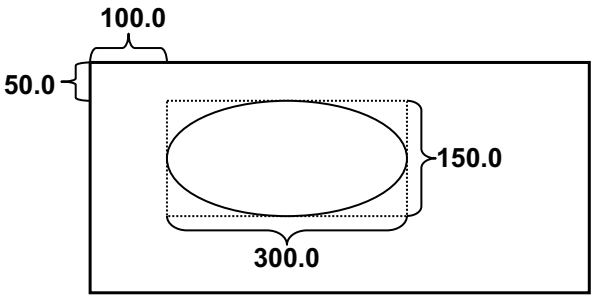



```
Shape e= new
Ellipse2D.Double(
100,50,300,150);

g2.draw(e);
```

Fill Ellipse; Draw Rectangle, Line

- To fill a shape, substitute the `fill()` method for `draw()`.
- If you want to draw in a different color, use the `Graphics2D` method `setPaint()` using any color as an argument



- To create a rectangle

```
Shape rect= new Rectangle2D.Double( double x,
  double y, double width, double height );
```

- To create a line

```
Line2D li= new Line2D.Double( double x1, double
y1, double x2, double y2 );
```

2D Shapes Provided in Java

- Predefined shapes include:
  - `Arc2D`
  - `CubicCurve2D`
  - `Ellipse2D`
  - `QuadCurve2D`
  - `Rectangle2D`
  - `RoundRectangle2D`
  - all with `Double` and `Float` versions
- `Line2D` is not strictly a `Shape` because you can only draw it, not fill it.
- To draw lines or shapes with greater thickness, change the "pen" with `setStroke()`:
  - `g2.setStroke(new BasicStroke(3)); // 3 pixels wide`

Exercise 4: Drawing

- Add code to `AreaPanel`'s `paintComponent` method to:
  - Create one `Ellipse2D`, `Rectangle2D`, and `Line2D` object
    - Make the ellipse be a circle
  - Make each object a different color, e.g.,
    `g2.setPaint(Color.black)`
  - Fill the rectangle object; draw the ellipse object
  - Show the area of the ellipse and rectangle, in pixels, using `Graphics2D.drawString()`
- Import `java.awt.geom.*` to have `Ellipse2D`, etc.

Fonts

- Standard constructor:

```
Font myFont =
  new Font( String name, int style, int size );
```

- Font name: safe approach is to use a logical font name, one of
  - `"SansSerif", "Serif", "Monospaced", "Dialog", "DialogInput", "Symbol"`
- Four font styles are present: Font.y where y is
  - `PLAIN, BOLD, ITALIC`
  - `Font.BOLD + Font.ITALIC  // Combines fonts`
- Size is point size; 12 corresponds to standard printed text
- Components that display text (like a `JLabel`) have a `setFont()` method that takes a `Font` object as an argument

Exercise 5: Font

- Change the font in AreaPanel to:
  - Monospaced
  - Bold
  - 20 point
- By creating a new Font object
- And using g2.setFont()
  - Argument is a Font object

Graphics 2D Attributes

- Much of the power of the 2D API comes from the user's ability to set attributes of the Graphics2D object known collectively as the rendering context:
  - `public void setStroke(Stroke s)// BasicStroke b`
  - `public void setPaint(Paint p)  // Color c`
  - `public void setFont(Font f)`
  - `// Combine new pixels with existing pixels`
    `public void setComposite(Composite c)`
  - `// Appearance: antialiasing, etc.`
    `public void setRenderingHints(Map m)`
  - `// Scale, rotate, translate (covered later)`
    `public void setTransform(Transform t)`
- Look these up in Javadoc; you should be getting comfortable using it

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012