Swing, Part 4 of 4

Christopher Simpkins
chris.simpkins@gatech.edu

WindowListeners, Adapters, Graphics and Drawing

- `WindowListener` and `WindowAdapter`
- `ImageIcon`s
- Graphics
- Colors
- Fonts and `drawString`

java.awt.event.WindowListener

```
public interface WindowListener extends EventListener {
    public void windowOpened(WindowEvent e);
    public void windowClosing(WindowEvent e);
    public void windowClosed(WindowEvent e);
    public void windowIconified(WindowEvent e);
    public void windowDeiconified(WindowEvent e);
    public void windowActivated(WindowEvent e);
    public void windowDeactivated(WindowEvent e);
}
```

See the API docs for details on each of these methods.

- To implement `WindowListener`, you'd have to provide implementations for each method.
- In most cases you're only interested in a few events.

java.awt.event.WindowAdapter

`WindowAdapter` provides empty implementations of each method in `WindowListener` (and a couple of other interfaces).

```
public abstract class WindowAdapter
  implements WindowListener, WindowStateListener, WindowFocusListener {
    public void windowOpened(WindowEvent e) {}
    public void windowClosing(WindowEvent e) {}
    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
    public void windowStateChanged(WindowEvent e) {}
    public void windowGainedFocus(WindowEvent e) {}
    public void windowLostFocus(WindowEvent e) {}
}
```

- For simple window event listeners, extending `WindowAdapter` is easier than implementing `WindowListener`.
- The standard library provides similar adapter classes for other large interfaces, such as `MouseListener` (`MouseAdapter`).

Using `java.awt.event.WindowAdapter`

`WindowAdapter` allows us to deal with only the single method from `WindowListener` that we actually care about:

```
private class JackWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        int choice = JOptionPane.showConfirmDialog(
            Jack.this,
            "Do you really want to exit?",
            "Exit for reals?",
            JOptionPane.OK_CANCEL_OPTION);
        if (choice == JOptionPane.YES_OPTION) {
            System.exit(0);
        }
    }
}
```

Then you'd have the following lines in your constructor:

```
  // Need to set DO_NOTHING_ON_CLOSE so we can handle window closing
  setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);

  // Confirm exit before exiting program
  addWindowListener(new JackWindowListener());
```

ImageIcons

An *icon* is a fixed-sized picture.

- Swing components such as `JButton`s, `JMenuItem`s and `JLabel`s can display `ImageIcon`s.
- Typical sizes for button and menu item icons are 16x16 or 32x32. Oracle provides a set of ready-to-use icons in the Java look and feel Graphics Repository
- `ImageIcon`s can be used for displaying any image, not just icons for GUI components. (But you wouldn't use `ImageIcon` as the basis for an image-editing application.)

Create an `ImageIcon` by passing the name of an image file to the constructor:

```
ImageIcon jackIcon = new ImageIcon("JACK-HEARTS.png");
```

Take a look at Jack.java for simple examples of using `ImageIcon`s and `WindowAdapter`.

The `paint` Method

All `java.awt.Component`s have a `paint` method which you can override to draw arbitrary graphics on the screen. The general form of such an overriden `paint` method, shown here with a `JFrame` is:

```
public class MyFrame extends JFrame {
    // ...
    public void paint(Graphics g) {
        super.paint(g);
        // Custom drawing commands here ...
    }
}
```

A component's `paint` is called whenever the component

- is first made visible on the screen,
- is resized, or
- has damage that needs to be repaired. (For example, something that previously obscured the component has moved, and a previously obscured portion of the component has become exposed).

The `Graphics` Object

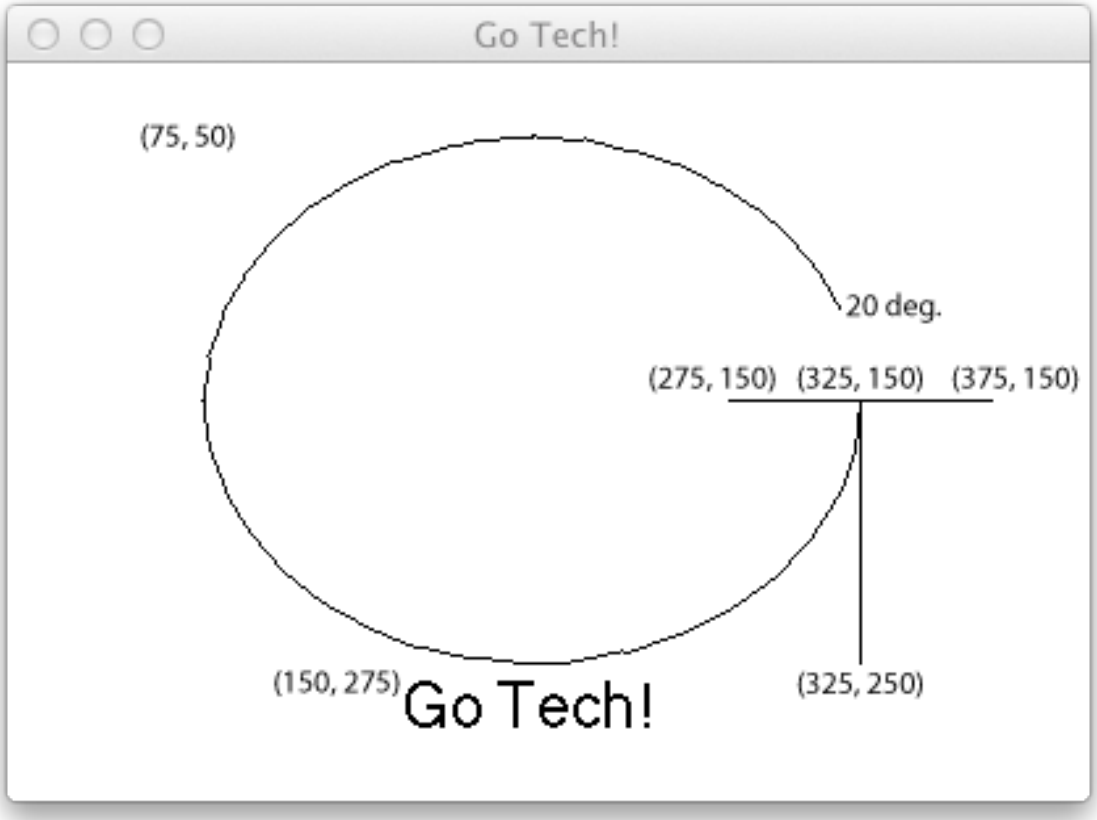
The `java.awt.Graphics` instance that's passed into the `paint` method keeps track of where the component is on the screen and provides component-relative drawing methods to draw on the component as well as methods to set background color and more. Here's a simple demonstration of several `Graphics` methods. What does this paint?

```
public void paint(Graphics g) {
    super.paint(g);
    g.setColor(new Color(207, 181, 59));
    g.drawArc(75, 50, 250, 200, 20, 340);
    g.drawLine(275, 150, 375, 150);
    g.drawLine(325, 150, 325, 250);
    g.setFont(new Font("Helvetica", Font.BOLD, 24));
    g.drawString("Go Tech!", 150, 275);
}
```

Try it out: GtFrame.java

Coordinates on a `Graphics` Object

Here are the important points we used to draw the `GtFrame`:



Key points:

- (75, 50) is upper-left corner of the bounding rectangle for the arc.
- `drawArc` uses the 3-o'clock position as 0 degrees and sweeps counter-clockwise.

Closing Thoughts on Swing

- There's much more to Swing and GUI programming.
- You've learned the basics of GUIs and seen a good example of an object-oriented framework (Swing).
- Underneath it all, it's just programs made of the same stuff we've learned already - data and processes, variables and control structures, classes and methods.
- If you want to become a Java GUI programming ninja, learn JavaFX. JavaFX is the future of GUI programming in Java.