Computer Science 210: Data Structures

Intro to Java Graphics

Summary

- Today
  - GUIs in Java using Swing
  - in-class: a Scribbler program

- READING:
  - browse Java online Docs, Swing tutorials

GUIs in Java

- Java comes with two libraries for creating GUIs
  - Swing and Awt
  - Swing is implemented on top of AWT

- All your programs (that have a GUI) will start by importing Swing and Awt classes

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

- Swing/Awt provide definition of standard classes used in GUIs
  - panels, labels, frames, buttons, scroll bars, text labels etc
  - all classes in Swing start with J
    - JButton, JComboBox,JDesktopIcon,JSeparator,JSlider,JScrollPane,JLabel, JProgressBar, JTable etc
  - called **components**

GUIs in Java

- Components
  - JButton, JComboBox, JDesktopIcon, JSeparator, JSlider, JScrollPane, JLabel, JProgressBar, JTable etc
- Components are organized in a hierarchy
  - at the top level, a component that handles windows
    - top-level containers: JFrame, JDialog, JApplet
    - we'll use JFrame
  - the window may contain panels that contain buttons and labels and so on
  - components that are not top-level containers must be attached to some other component

Example

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


//a class that handles a window
public class MyClass extends JFrame  {

    // instance variables
    ....

    public MyClass() {
        super("My window");
        setSize(400, 400);

        //exit when closing the window
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setVisible(true);
    }

};
```

Handling the mouse

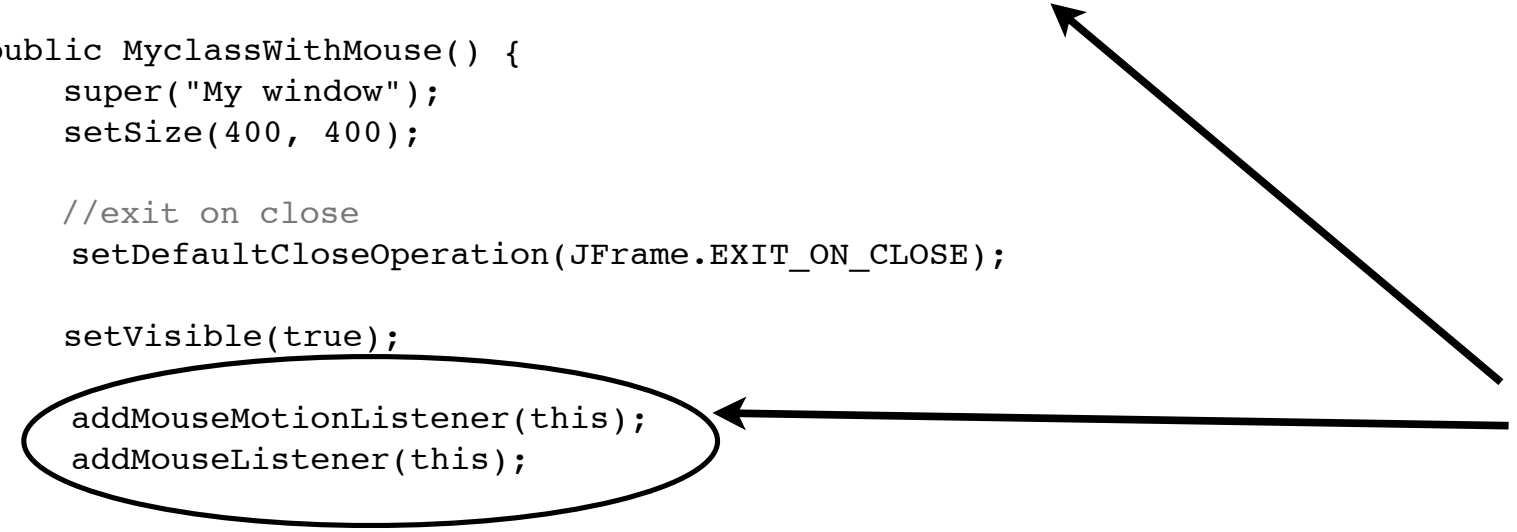
- Mouse events are sent to the objects that “registered” to “listen” to the mouse
- The mouse events are sent through a set of methods with predefined names

- To handle the mouse
  1. the class must implement one or both of these interfaces
     - MouseMotionListener
     - MouseListener

  2. the object must register itself as a mouse “listener”
     - the mouse events will be sent to all objects that are registered as “listeners” by calling the methods defined by the interface
     - Note: if the registration is in the constructor of the class, then every instance of the class will “listen” to the mouse

- In general
  - mouse events --> register as a mouse listener, etc
  - timer events --> register as a time listener
  - for each type of event, there exists a corresp method to register as listener
  -

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

//a class that handles the mouse
public class MyclassWithMouse extends JFrame implements MouseInputListener {

    public MyclassWithMouse() {
        super("My window");
        setSize(400, 400);

        //exit on close
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setVisible(true);

        addMouseMotionListener(this);
        addMouseListener(this);
    }

    public void mousePressed(MouseEvent e) {}

    public void mouseDragged(MouseEvent e) {}

    public void mouseReleased(MouseEvent e) {}

    public void mouseClicked(MouseEvent e) {}

    public void mouseEntered(MouseEvent e) {}

    public void mouseExited(MouseEvent e) {}

    public void mouseMoved(MouseEvent e) {}

};
```



Drawing in a window

- To draw you need a canvas

```
Graphics  g ;
```

- Need to grab the canvas of the JFrame

```
Graphics g = this.getGraphics();
```

- Methods supported by class Graphics
  - `drawLine(Point p1, Point p2)`
  - `drawImage(..)`
  - `drawOval..`
  - `drawPolygon..`
  - `drawRect..`
  - `getColor, setColor..`
  - `getFont, setFont..`

- Java coordinate system:
  - (0,0) upper left corner

In-class work

- Test mouse functionality
  - write code in the various mouse methods and check when they get called

- Develop a program that lets the user scribble on the window
  - record the mouse clicks
  - when pressing the mouse you want to start drawing; if you keep the mouse pressed and drag it around, you want the movement to be shown on screen, until the mouse is released.
  - in addition to the skeleton above, you need some instance variables to record position
    - you can use integers, or class Point provided by Java

The painting mechanism in Swing

- Problem: render/paint the right things at the right time
- Swing: any component has (inherits) a method called paint
  - `public void paint(Graphics g)`
  - the component should place the rendering code inside paint()
  - paint() is invoked every time it's time to paint
- A call to paint() can be triggered:
  - by the system
    - the component is made visible
    - the component is resized
    - the component needs to be repaired (i.e. some other window that was previously obscuring this component has moved away)
  - by the the application
    - when the program decides it needs to re-paint the component
- When the system invokes paint() on a component, it pre-configures a Graphics object with the current Graphics context and passes it as argument to paint()

The painting mechanism in Swing

- Programs should place the rendering code inside paint()
  - override paint()
- Programs should avoid placing rendering code at any point where it might be invoked outside paint
  - Why? Because such code may be invoked at times when it is not appropriate to paint -- for instance, before the component is visible or has access to a valid Graphics object.

- Programs should NOT invoke paint() directly.
- Instead, use
  - `public void repaint()`

- In fact, Swing components should override
  - `public void paintComponent(Graphics g)`

- Paint mechanism is complicated
- We'll keep GUIs simple
  - GUIs are a tool for the class, not the focus

- Here is an example of a paint() method which renders a filled circle in the bounds of a component:

```
public void paint(Graphics g) {
        //clear the screen
        super.paint();

        // Dynamically calculate size information of the component
        Dimension size = getSize();

        // diameter
        int d = Math.min(size.width, size.height);
        int x = (size.width - d)/2;
        int y = (size.height - d)/2;

       // draw circle (color already set to foreground)
       g.fillOval(x, y, d, d);
       g.setColor(Color.black);
       g.drawOval(x, y, d, d);
   }
```

Class work

- re-write Scribbler
  - place all render code in paint()
  - call repaint() when appropriate