Computer Science 210: Data Structures
Summary

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

• READING:
  • Java online Docs
GUIs in Java

• use Swing
  • toolkit for designing GUIs
  • implemented on top of AWT (another toolkit)
  • provides uniform look across platforms
  • provides customized looks, etc

• Swing provides components
  • panels, labels, frames, buttons, scroll bars, text labels etc

• all classes in Swing start with J
• components are organized in a hierarchy
  • at the top level, a component that handles windows
    • we’ll use JFrame
    • also JDialog, JApplet

• the window may contain panels that contain buttons and labels and so on
• components must be attached to some other component
import javax.swing.*;
import java.awt.*;

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

    // instance variables
    ....

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

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

        setVisible(true);
    }

};
Handling the mouse

- the class must implement one or both
  - MouseMotionListener
  - MouseListener

- the class must register as a mouse listener
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

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

    public GraphicsAndMouse() {
        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

- draw on a canvas
  
  ```java
  Graphics g;
  ```

- need to grab the canvas of JFrame
  
  ```java
  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 a few 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 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 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 context and passes it as argument
Here is a simple example of a paint() which renders a filled circle in the bounds of a component:

```java
public void paint(Graphics g) {
    // Dynamically calculate size information
    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);
}
```
The painting mechanism

- 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()`

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

- overriding `paint()` is ok
- paint mechanism is complicated
- we’ll keep GUIs simple
- GUIs are a tool for our class, not the focus
in-class work

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