Lecture 3: Java Graphics & Events

CS 62
Fall 2017
Kim Bruce & Alexandra Papoutsaki

Text Input

- **Scanner** class
  - Constructor: `myScanner = new Scanner(System.in);`
  - can use file instead of `System.in`
    - `new Scanner(new File("filename"))`
  - Read values:
    - `myScanner.nextInt()` -- returns an `int`
    - `myScanner.nextDouble()` -- returns a `double`
    - `myScanner.nextLine()` -- returns `String` -- to end of line
  - see documentation for more

For more details

- See document on course web page associated with lecture.
- See GUI cheat sheet in documentation and handouts section.

Overview

- Graphical User Interfaces (GUI)
  - `JFrame` (window), `JPanel` (grouping)
  - `JButton`, `JTextField`, `JSlider`, `JChooser`, ...
- Graphics
  - Drawing items on the screen
- Events
  - Generated by mouse actions, button clicks etc.
  - Use `MouseListener`, `MouseMotionListener`, `ActionListener`, etc. to respond
Graphical User Interfaces (GUIs)

- **AWT**: The Abstract Windowing Toolkit is found in the package `java.awt`.
  - Heavyweight components
  - Implemented with native native code written for that particular computer
  - The AWT library was written in six weeks!

- **Swing**: Java 1.2 extended AWT with the `javax.swing` package.
  - Lightweight components
  - Written in Java

### JFrame

- `javax.swing.JFrame` inherits from `java.awt.Frame`
  - The outermost container in an application.
  - To display a window in Java:
    - create a `JFrame`
    - set the size
    - set the location
    - set it visible

```java
import javax.swing.JFrame;

public class MyFirstGUI extends JFrame{
    public MyFirstGUI() {
        super("First Frame");
        setSize(500, 300);
        setLocation(100, 100);
        setVisible(true);
    }

    public static void main(String[] args) {
        MyFirstGUI mfgui = new MyFirstGUI();
    }
}
```

### Screen Location

Assume 1440x900 screen resolution

- Increase Y
- Increase X

- `NewFrame`
Positioning a window

```java
import javax.swing.JFrame;
import java.awt.Dimension;
import java.awt.Toolkit;

public class MySecondGUI extends JFrame {
    public MySecondGUI() {
        super("Second Frame");
        setSize(500, 300);
        Toolkit toolkit = getToolkit();
        Dimension size = toolkit.getScreenSize();
        setLocation(size.width/2 - getWidth()/2,
                    size.height/2 - getHeight()/2);
        setVisible(true);
    }
}
```

Closing a GUI

- The default operation of the quit button is to set the visibility to false
  - The program does not terminate!
- `setDefaultCloseOperation` can be used to control this behavior.
  ```java
  mgui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  ```
  - Exits the application using `System.exit(0)`
- More options (hide, do nothing, etc).

Basic Controls

Interactive Displays
### Graphics

- Create objects you want to draw:
  - `Rectangle2D.Double`, `Line2D.Double`, etc.
    - Constructors take x, y coords and dimensions, but don’t actually draw items.
- All drawing takes place in `paint` method using a “graphics context”
- Triggered implicitly by uncovering window or explicitly by calling `repaint` method.
  - Adds repaint event to event queue – eventually draws it

### Graphics context

- All drawing is done in “paint” method of component
- `public void paint(Graphics g)`
  - g is a Graphics context provided by system
  - “pen” that does the drawing
  - Programmer calls repaint(), not paint!!
- Need to import classes from `java.awt.*`, `java.geom.*`, `javax.swing.*`
- See `MyGraphicsDemo`

### General Graphics Applications

- Create an extension of component (either `JPanel`, `JFrame`, or `JApplet`) and implement `paint` method in the subclass.
- See main method of demo to get window to show
- At start of paint method cast `g` to `Graphics2D` to get access to new methods
- Call `repaint()` on component every time you make a change.
  - Causes OS to schedule call of paint in event queue
  - Called automatically if window obscured and revealed

### Geometric Objects

- Objects from classes `Rectangle2D.Double`, `Line2D.Double`, etc. from `java.awt.geom`
  - There are also float versions
  - Constructors take params x, y, width, height, but don’t draw object
- `Rectangle2D.Double`
- `RoundRectangle2D.Double`
- `Ellipse2D.Double`
- `Arc2D.Double`
- `Line2D.Double`...
Methods

- `myObj.setFrame(x, y, width, height)`: can move object
- `g2.draw(myObj)`: gives outline
- `g2.fill(myObj)`: gives filled version
- `g2.drawString("a string", x, y)`: draws string

MyGraphicsDemo

- Class extends `JFrame`, which creates window.
- Constructor calls `super` with title of window.
- `main` method creates object, sets size, visibility, and enables go-away box.
- `paint` method creates and draws objects.
BorderLayout

PostItApplication

• More sophisticated.

• JFrame contains two JPanels.

• JFrame uses BorderLayout, so add controls to JPanel in SOUTH, drawing canvas in CENTER of the JFrame.

• DrawingCanvas extends JPanel -- contains paint method
  • Note use of ArrayList to hold PostIts

PostIt Class

• Represents the rectangles being dragged:
  • Contains accessor and mutator methods to allow it to be manipulated by drawing program.
  • Could add features (title bar, go-away box) without affecting PostItApplication code.

PostItApplication

• PostItApplication class responsible for
  • setting up the GUI
  • Responding to button pressed and menu selections
  • Sets up ArrayList of items on canvas.
  • Class has 3 inner classes
    • DrawingCanvas
    • DrawingMouseListener
    • DrawingMouseMotionListener
    • Inner classes have access to private features of containing class
**Inner Classes**

- **DrawingPanel** extends **JPanel**
  - Associates listeners for mouse actions on the panel
  - Responsible for repainting the screen
- **DrawingMouseListener** and **DrawingMouseMotionListener**
  - Responsible for responding to mouse actions by changing the items in the ArrayList.

**Event-Driven Programming**

**Handling Mouse Events**

- If want program to react to mouse press, click, or release on a component
  - send `addMouseListener(ml)` to component (usually in the constructor of the component)
  - See `PostItApplication.java`
  - For motion or drag, send `addMouseMotionListener(ml)`
- When user presses mouse on a component
  - Computer looks for registered "MouseListener" for component or its containers.
  - If found, sends `mousePressed(evt)` to listener

**Listener**

- object designated as mouse listener must
  - implement `MouseListener` (implement `mousePressed`, `mouseReleased` & `mouseClicked` or
    - extend `MouseAdapter` (which has default implementations of all 3)
  - Second is easier unless class already extends another. Can only extend one class in Java
- Similarly, for mouse motion listener
  - implement `MouseMotionListener` or
    - extend `MouseMotionAdapter`
Gui Objects & Events

- Similar to handling mouse events, but must also install components in a container.
- See GUI cheat sheet in Documentation & Handouts.

Listeners in PostItApplication

- Main class (this) is listener for button and choice. Set up when GUI items constructed.
- Special listener objects for mouse actions. Set up by DrawingCanvas since listening for actions on that object.

List Operations

- Review list operations from library interface List in Java 8 documentation.
- Bailey's List is slightly different.
- Think about how to implement with array.
- size, isEmpty, get, set functions

ArrayList

- See Bailey’s ArrayList
- Similar to Java 8’s ArrayList
- Instance variables:
  - elts array instance variable,
  - eltsFilled number of slots filled.
- Some operations very cheap:
  - size, isEmpty, get, set take constant time (no search)
- Others more expensive