Programming Languages and Techniques (CIS120)

Lecture 36

April 18, 2012

Swing IV: Mouse and Keyboard Input

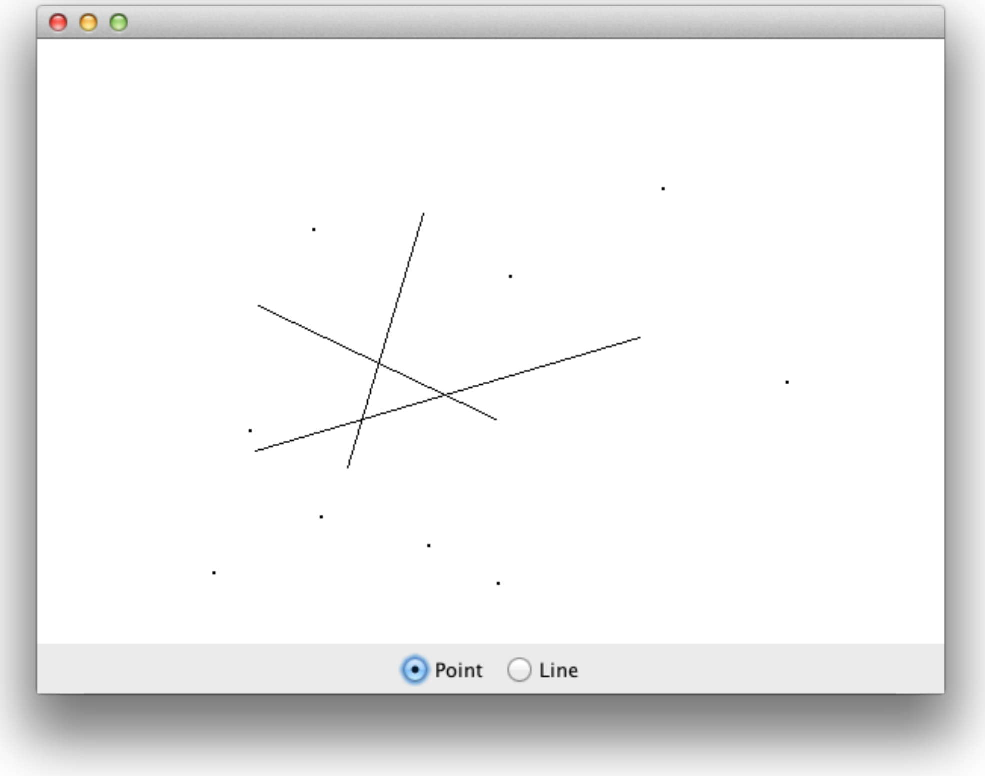
Announcements

- Lab this week is review (BRING QUESTIONS)

- Game Project is out, due Tuesday April 24th
  - If you want to do a game other than one of the ones listed, send email to tas120@seas.upenn.edu

- Final Exam
  - Date: Tuesday, May 8th
  - Time: 9:00 AM-11:00 AM
  - Place: SKIR AUD
  - Review session: TBA during finals

Paint



Mouse and Keyboard interaction

Basic structure

- Main frame for application (class Paint) the *MODEL*
- Drawing panel (class Canvas, inner class of Paint) the *VIEW*
- Control panel (class JPanel)
  - Contains radio buttons for interacting with the program
  - (part of) the *CONTROL*

- Paint class contains the state of the program
  - List of shapes to draw
  - The current color (will always be BLACK today)
  - References to UI components: canvas, modeToolbar

- How can users update that state?

Keyboard Interaction

Keyboard Interaction

- How to make the program responsive to keyboard input?
- Concept: keyboard focus
  - A “Focusable” UI Component is one that can respond to keyboard input
  - Java method “requestFocusInWindow” gives the focus to a particular component
  - Registered KeyListeners for the component react when it is in focus

- KeyListener Interface
  - void **keyPressed(KeyEvent e)**
    Invoked when a key has been pressed
  - void **keyReleased(KeyEvent e)**
    Invoked when a key has been released
  - void **keyTyped(KeyEvent e)**
    Invoked when a key has been typed

- Use KeyAdapter to easily make an instance of this interface

Paint: Comparison with OCaml

How does our treatment of shape drawing in Java compare with the OCaml GUI project?

Java Design Summary

```
public interface Shape {
    public void draw(Graphics gc);
}
```

Interface describes what shapes can do

```
public class PointShape implements Shape { … }
public class LineShape implements Shape { … }
```

Classes describe how to draw themselves

```
private class Canvas extends JPanel {
   public void paintComponent(Graphics gc) {
        super.paintComponent(gc);
        for (Shape s : actions)
            s.draw((Graphics2D)gc);
        if (preview != null)
            preview.draw((Graphics2D)gc);
  }
}
```

Canvas uses dynamic dispatch to draw the shapes

```
type point = int * int
type shape =
  | Point    of Gctx.color * int * point
  | Line     of Gctx.color * int * point * point

(* Repaint function for displaying the canvas. *)
let repaint (g:Gctx.t) : unit =
  let actions = List.rev paint.shapes in
  let drawit d =
    begin match d with
    | Point (c,t,p) ->
        Gctx.draw_points (set_params g c t) p
    | Line (c,t,p1,p2) ->
        Gctx.draw_line (set_params g c t) p1 p2
    end in
  List.iter drawit actions
```

Comparison with OCaml

- How does our treatment of shape drawing in the Java Paint example compare with the OCaml GUI project?

- Java:
  - Interface Shape for drawable objects
  - Classes implement that interface
  - Canvas uses dynamic dispatch to draw the shapes
  - Add more shapes by adding more implementations of "Shape"
- OCaml
  - Datatype specifies variants of drawable objects
  - Canvas uses pattern matching to draw the shapes
  - Add more shapes by adding more variants, and modifying drawit

Datatypes vs. Objects

Datatypes

- Focus on how the data is stored
- Easy to add new operations
- Hard to add new variants

- Best for: situations where the structure of the data is fixed (i.e. BSTs)

Objects

- Focus on what to do with the data
- Easy to add new variants
- Hard to add new operations

- Best for: situations where the interface with the data is fixed (i.e. Shapes)

What about Modes?

Is Enum the best way to represent them?