# Programming Languages and Techniques (CIS120)

Lecture 35

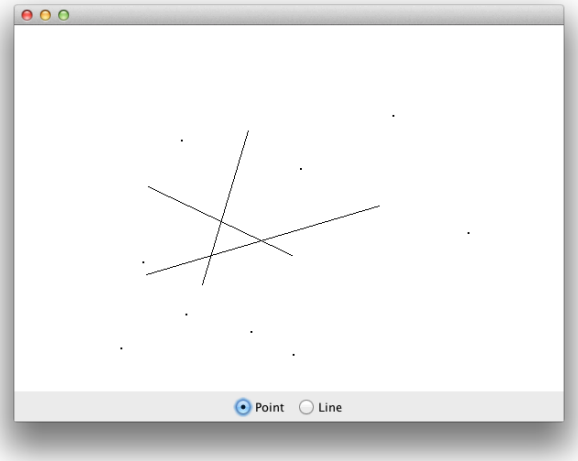
April 16, 2012

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

# 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?

# Mouse Interaction

## Basic Mouse Interaction

- Copy OCaml structure and add *modes* to the model

```
public enum Mode {
  PointMode, LineStartMode, LineEndMode
}
private Mode mode = Mode.PointMode;
```

- Button press switches between PointMode and LineStartMode
- Mouse click in PointMode ➔ add a new point to the list of shapes
- Mouse click in LineStartMode ➔ remember location, switch to LineEndMode
- Mouse click in LineEndMode ➔ add a new line to list of shapes, switch to LineStartMode

# Drag-and-drop

- Implement drag-and-drop lines
  1. When the button is pressed, record the current mouse location as a point, change the mode to LineEndMode and store the current point
  2. As the mouse is dragged with the button pressed, set preview to be a line from the stored point to the current position of the mouse
  3. When the button is released, update the mode to LineStartMode, set preview to null, and add the new line to the list of actions

- In OCaml, single event listener for *all* events
- In Java, things are a bit more sophisticated…

# Two kinds of mouse listeners

```
interface MouseListener extends EventListener {
  public void mouseClicked(MouseEvent e);
  public void mouseEntered(MouseEvent e);
  public void mouseExited(MouseEvent e);
  public void mousePressed(MouseEvent e);
  public void mouseReleased(MouseEvent e);
}
```

```
interface MouseMotionListener extends EventListener {
  public void mouseDragged(MouseEvent e);
  public void mouseMoved(MouseEvent e);
}
```

# Lots of boilerplate

- There are seven methods in the two interfaces.
- We only want to do something interesting for three of them.
- Need "trivial" implementations of the other four to implement the interface…

```
public void mouseMoved(MouseEvent e) { return; }
public void mouseClicked(MouseEvent e) { return; }
public void mouseEntered(MouseEvent e) { return; }
public void mouseExited(MouseEvent e)  { return; }
```

- Solution: MouseAdapter class
- Implements all seven methods trivially
- Subclasses override only the ones they want.

# Adapter classes:

- Swing provides a collection of abstract event adapter classes
- These adapter classes implement listener interfaces with empty, do-nothing methods
- To implement a listener class, we extend an adapter class and override just the methods we need

```
private class Mouse extends MouseAdapter {
   public void mousePressed(MouseEvent e) { … }
   public void mouseReleased(MouseEvent e) { … }
   public void mouseDragged(MouseEvent e) { … }
}
```