Programming Languages and Techniques (CIS120)

Lecture 35

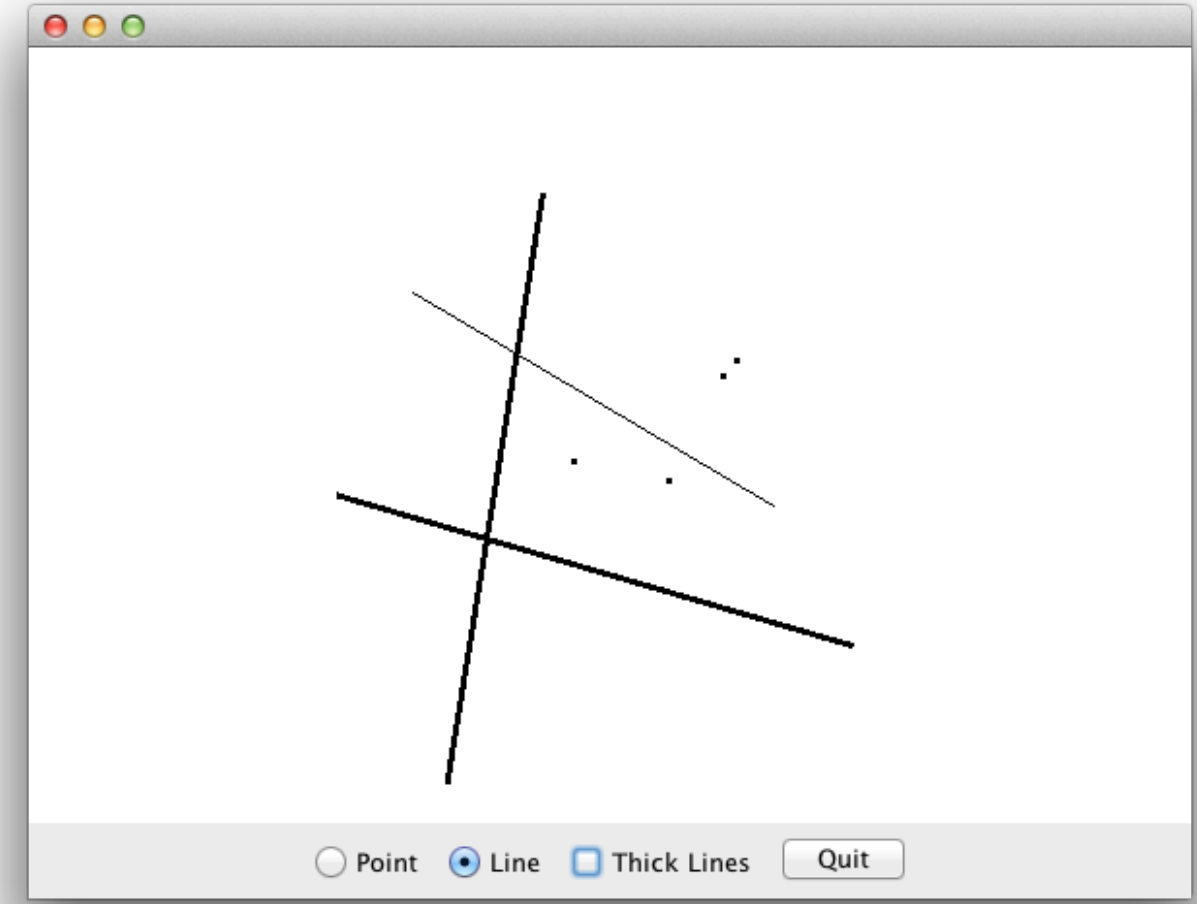
April 21, 2014

Swing III: Paint demo, Mouse Interaction

Announcements

- HW 10 has a HARD deadline
  - You must submit by midnight, April 30th
  - Demo your project to your TA during reading days
- TAs have advice for your game
  - source control tutorial
  - collisions
  - difficulty guide
- Friday's lecture is a BONUS lecture
  - Not directly related to game project or Java libraries
  - No clicker quizzes
  - Fun!

Java Paint



Basic structure

- Main class for the application (the *MODEL)*
- Drawing area, i.e. Canvas (the *VIEW)*
- Control panel (the *CONTROLLER)*

- Model *stores* the state of the application
  - Collection of shapes to draw
  - Preview shape (if any…)
  - The current color (will always be BLACK today)
  - The current line thickness
- View *displays* the state
  - overrides paintComponent method and draws each shape in the list
- Controller *updates* the state
  - Radio buttons for selecting shape to draw
  - Line thickness checkbox, undo and quit buttons

The Model

Dynamic Dispatch for Drawing

```
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 : shapes)
            s.draw(gc);
        if (preview != null)
            preview.draw(gc);
  }
}
```

Canvas uses dynamic dispatch to draw the shapes

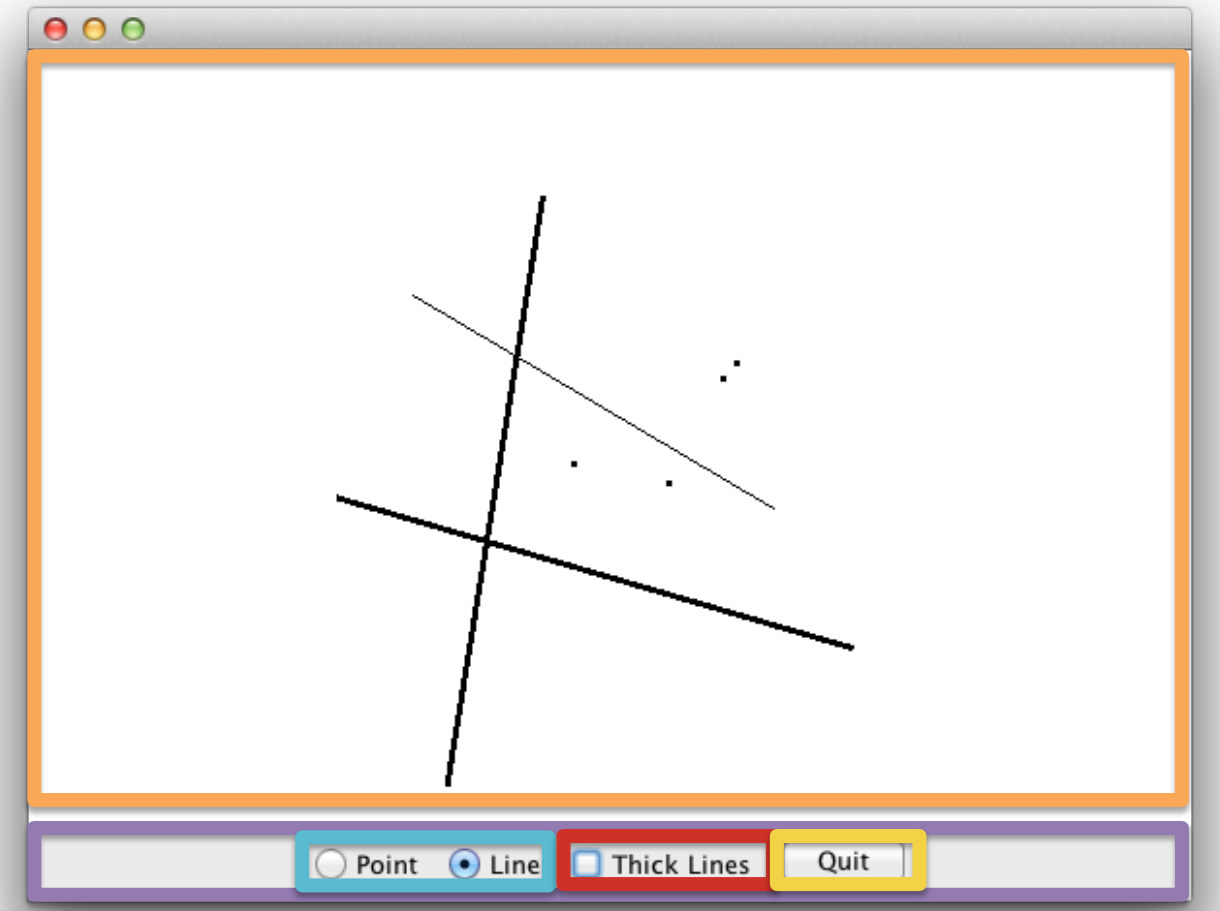
```
public class Paint {

    /** Preview shape for drag and drop */
    private Shape preview = null;
    /** Current drawing color */
    private Color  color  = Color.BLACK;
    /** Current drawing thickness */
    private Stroke stroke = thickStroke;

    /** Stroke for drawing shapes with thin lines */
    public final static Stroke thinStroke  = new BasicStroke(1);
    /** Stroke for drawing shapes with thick lines */
    public final static Stroke thickStroke = new BasicStroke(3);

    /** The shapes that will be drawn on the canvas. */
    private ??? shapes = new ????();
```

What Collection class should we use to store the shapes ?

1. TreeSet<Shape>
2. TreeMap<Shape>
3. LinkedList<Shape>
4. Iterable<Shape>
5. None of the above

The Controller

updates the state of the model



Canvas
(canvas)

JPanel
(modeToolbar)

JRadioButton
(point,line)

JCheckbox
(thick)

JButton
(quit)

Demo: Controller Layout

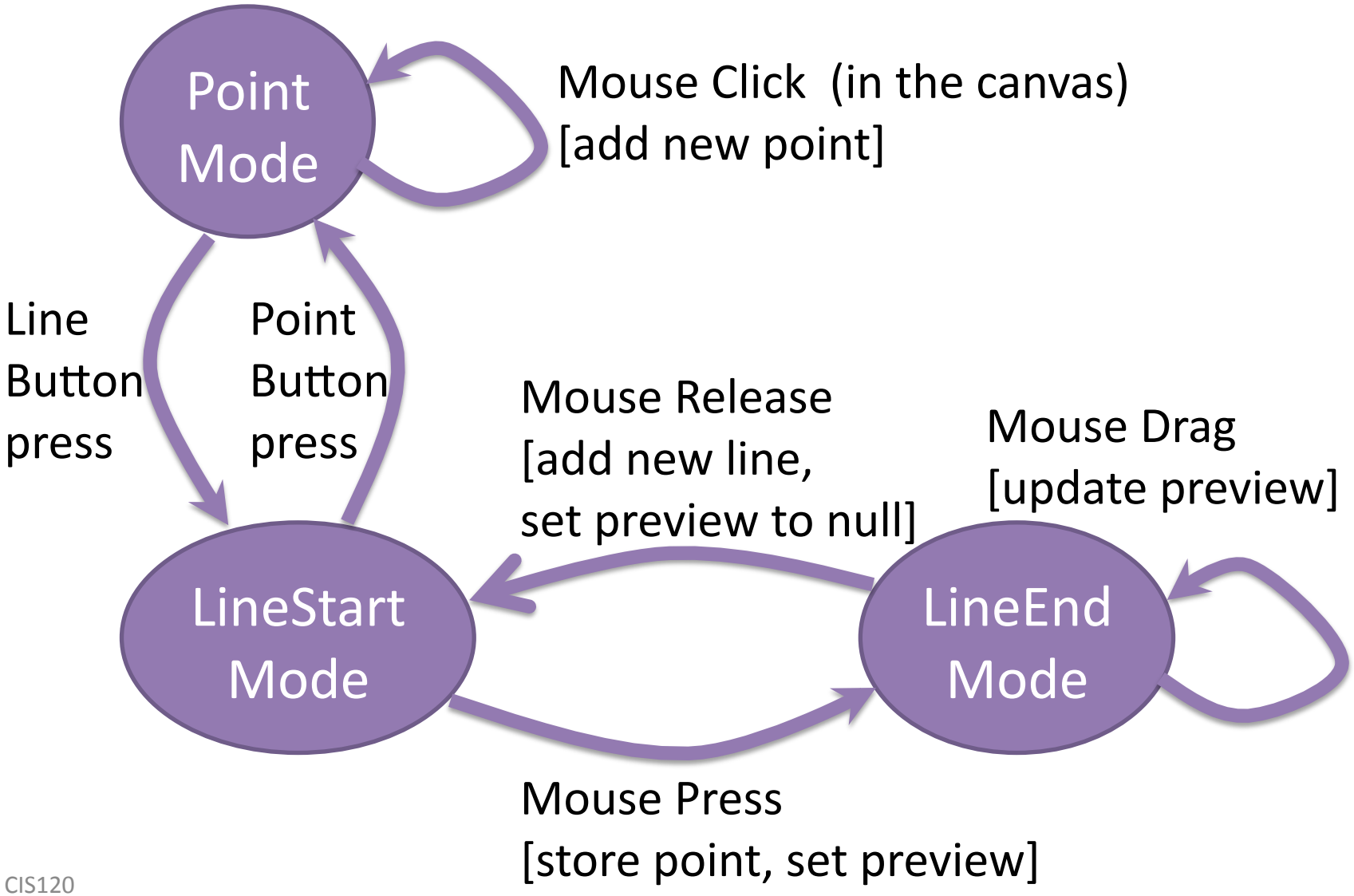
Paint.java

Mouse Interaction

- How do we specify what shapes to draw on the canvas?

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

Interaction



Two interfaces for 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…

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) { … }
}
```