

Programming Languages and Techniques (CIS120)

Lecture 34

April 18, 2013

Swing II: Layout & Designing a GUI app

How is HW10 going so far?

1. not started
2. started reading
3. got an idea for what game to write
4. started coding

```
List<Integer> list = new LinkedList<Integer>();  
list.add(1);  
list.add(2);  
list.add(3);  
  
Iterator<Integer> it = list.iterator();  
    while (it.hasNext()) {  
        int v = it.next();  
        if (it.next() > 1) {  
            System.out.print(v + " ");  
        }  
    }  
}
```

What is the result?

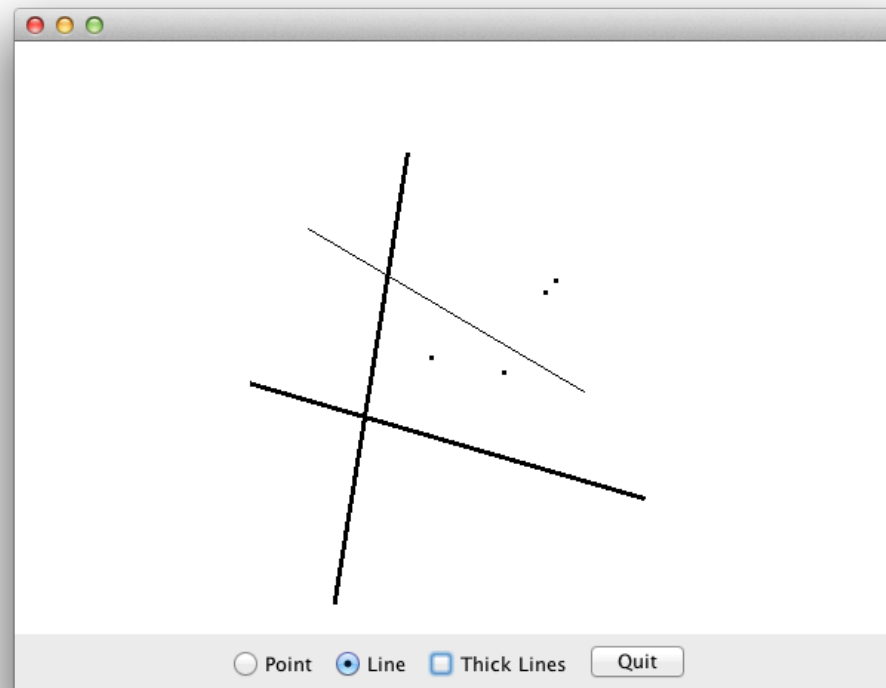
1. 2 and 3 are printed
2. 1 and 3 are printed
3. 1 is printed, then NoSuchElementException occurs
4. 2 is printed, then NoSuchElementException occurs
5. NullPointerException occurs
6. None of the above

LayoutManagers

Swing Programming Demo

What layout would you use for Paint?

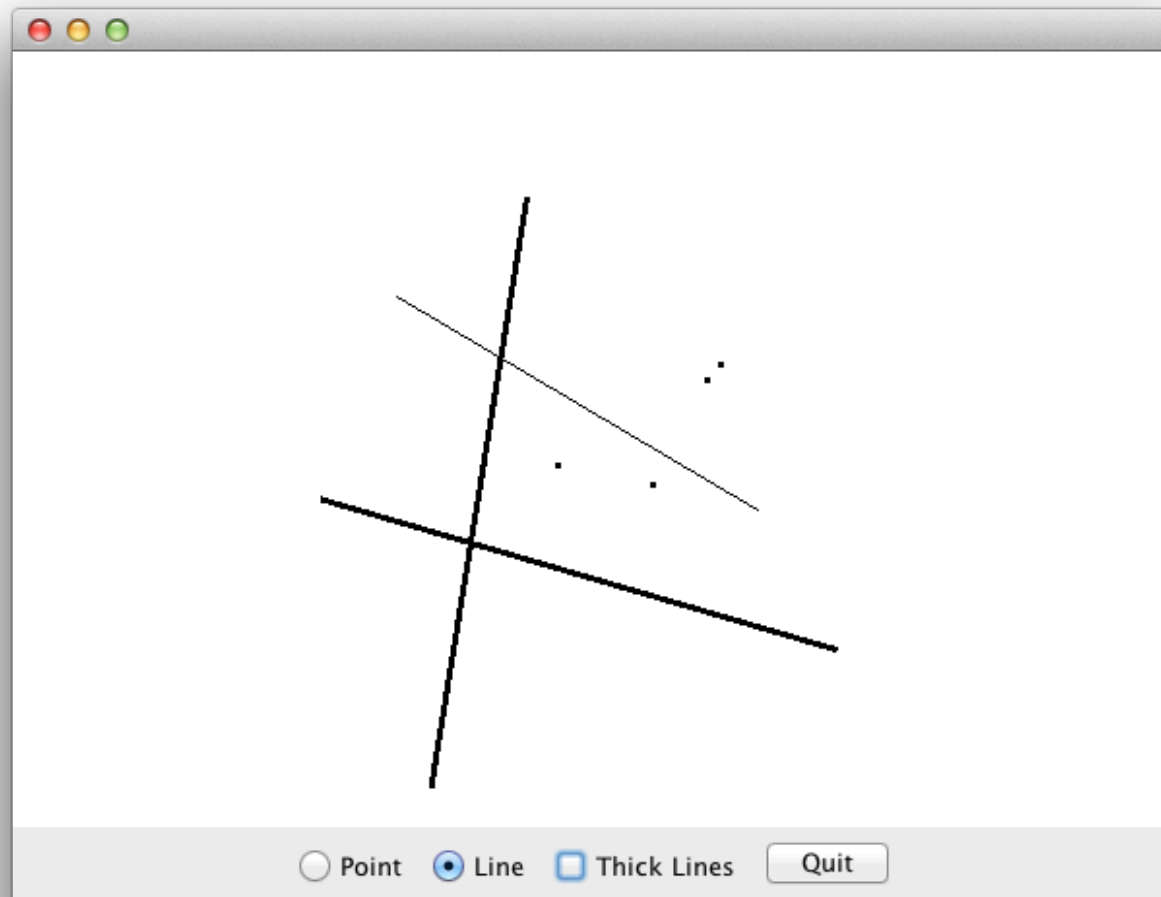
1. FlowLayout
2. GridLayout
3. BorderLayout
4. Something else?



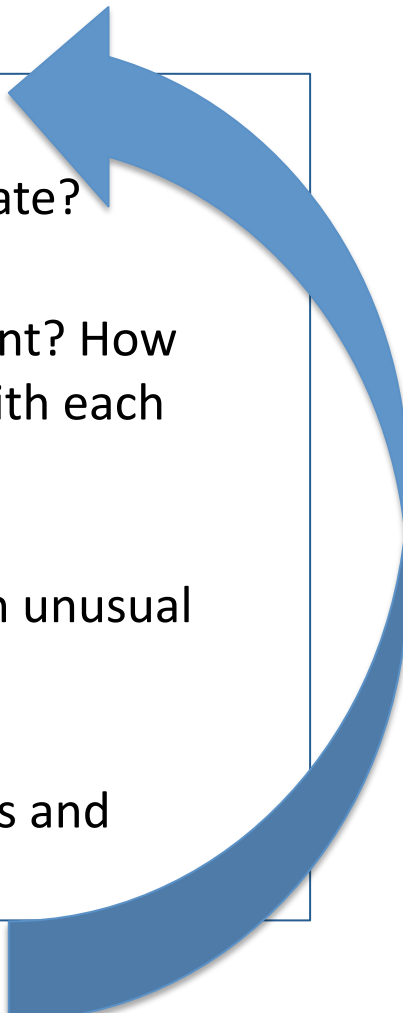
Design Exercise

Java Paint

Java Paint



Design Recipe

1. Understand the problem
What are the relevant concepts and how do they relate?
 2. Formalize the interface
How should the program interact with its environment? How should different parts of the program interact with each other?
 3. Write test cases
How does the program behave on typical inputs? On unusual ones? On erroneous ones?
 4. Implement the required behavior
Often by decomposing the problem into simpler ones and applying the same recipe to each
- 

5. Revise / Refactor / Edit

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
 - 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

How should we represent shapes?

1. Using a class:

```
public class Shape {  
    String name; // i.e. "Point" or "Line"  
    int x; int y;  
    int thickness;  
}
```

2. Using an enum:

```
public enum Shape {  
    Point, Line, Ellipse  
}
```

3. Using an interface:

```
public interface Shape { ... }  
public class Point implements Shape { ... }  
public class Line implements Shape { ... }  
public class Ellipse implements Shape { ... }
```

4. Something else?

OCaml Version of Paint

```
type shape =  
  | Points   of Gctx.color * int * point list  
  | Line     of Gctx.color * int * point * point  
  
let repaint (g:Gctx.t) : unit =  
  let draw_shape (s:shape) : unit =  
    begin match s with  
      | Points (c,t,ps) -> ...  
      | Line (c,t,p1,p2) -> ...  
    end in  
  Deque.iterate draw_shape paint.shapes;  
  begin match paint.preview with  
  | None -> ()  
  | Some d -> draw_shape d  
end
```

Datatypes define the structure of information.

Drawing operation is defined externally to the datatype and uses case analysis to dispatch.

The “main” loop looks very similar.

Java Version of Paint

```
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

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 draw

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)