

## CS1007: Object Oriented Design and Programming in Java

Lecture #18  
Dec 1

Shlomo Hershkop  
*shlomo@cs.columbia.edu*

## Outline

- Frameworks
  - Approach
  - Requirements
  - Code examples
  
- Reading: Chapter 8.2-8.5

## Graph editor framework

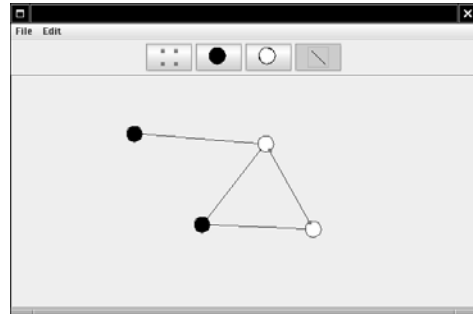
- Traditional approach: programmer starts from scratch for every editor type
- Framework approach: Programmer extends graph, node, edge classes
- Framework handles UI, load/save, ...
- Our framework is kept simple
- Violet uses extension of this framework

## Requirements

- What are the GUI requirements?
- What are the programming requirements?

## UI

- Toolbar on top
- Grabber button for selecting nodes/edges
- Buttons for current node/edge type
- Menu
- Drawing area



## Mouse Control

- Click on empty space: current node inserted
- Click on node or edge: select it
- Drag node when current tool an edge: connect nodes
- Drag node when current tool not an edge: move node

## Divide the work

- Divide code between
  - framework
  - specific application
- Rendering is app specific (e.g. transistor)
- Hit testing is app specific (odd node shapes)
- Framework draws toolbar
- Framework does mouse listening

## Adding nodes

- Framework draws toolbar
- How does it know what nodes/edges to draw?
- App gives a list of nodes/edges to framework at startup
- How does app specify nodes/edges?
  - Class names? ("Transistor")
  - Class objects? (Transistor.class)
  - Node, Edge objects? (new Transistor())

- Objects are more flexible than classes
- ```
new CircleNode(Color.BLACK)
new CircleNode(Color.WHITE)
```
- When user inserts new node, the toolbar node is cloned
  - Node prototype = node of currently selected toolbar button;
  - Node newNode = (Node) prototype.clone();
  - Point2D mousePoint = current mouse position;
  - graph.add(newNode, mousePoint);

## Framework Classes

- Framework programmer implements Node/Edge interfaces
- draw draws node/edge
- getBounds returns enclosing rectangle (to compute total graph size for scrolling)
- Edge.getStart, getEnd yield start/end nodes
- Node.getConnectionPoint computes attachment point on shape boundary
- Edge.getConnectionPoints yields start/end coordinates (for grabbers)
- clone overridden to be public

## Code

- AbstractEdge class for convenience
  - Programmer implements Node/Edge type or extends AbstractEdge
- ```
Ch8/graphed/Node.java
Ch8/graphed/Edge.java
Ch8/graphed/AbstractEdge.java
```

- Graph collects nodes and edges
- Subclasses override methods

```
public abstract Node[] getNodePrototypes()
public abstract Edge[] getEdgePrototypes()
```

Ch8/graphed/Graph.java

## Framework UI

- GraphFrame: a frame that manages the toolbar, the menu bar, and the graph panel.
- ToolBar: a panel that holds toggle buttons for the node and edge icons.
- GraphPanel: a panel that shows the graph and handles the mouse clicks and drags for the editing commands.
- Application programmers need not subclass these classes

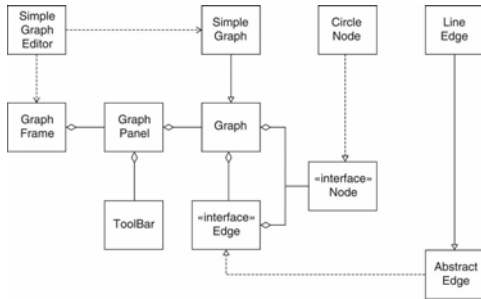
## Framework instance

- Simple application
- Draw black and white nodes
- Join nodes with straight lines

## Shopping List

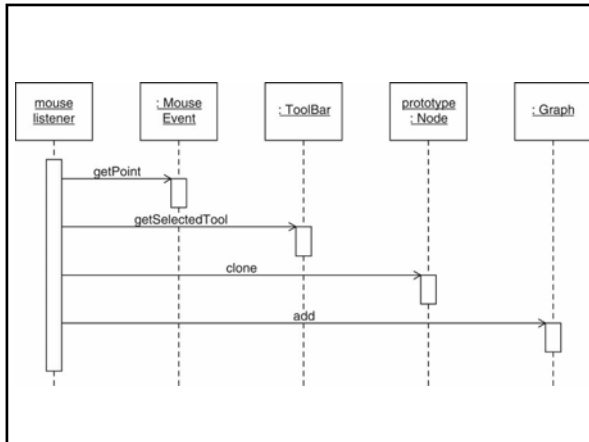
- For each node and edge type, define a class that implements the Node or Edge interface type
- Supply all required methods, such as drawing and containment testing.
- Define a subclass of the Graph class and supply `getNodePrototypes`, `getEdgePrototypes`
- Supply a class with a main method

## Big Picture



## Code

Ch8/graphed/SimpleGraph.java  
 Ch8/graphed/SimpleGraphEditor.java  
 Ch8/graphed/CircleNode.java  
 Ch8/graphed/LineEdge.java



## Adding new edges

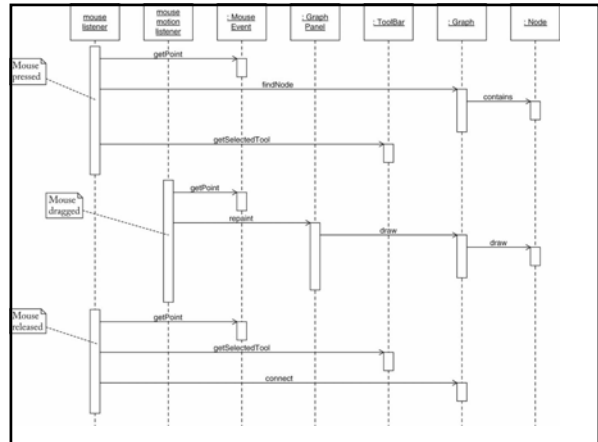
- First check if mouse was pressed inside existing node

```

public Node findNode(Point2D p)
{
    for (int i = 0; i < nodes.size(); i++)
    {
        Node n = (Node) nodes.get(i);
        if (n.contains(p)) return n;
    }
    return null;
}
    
```

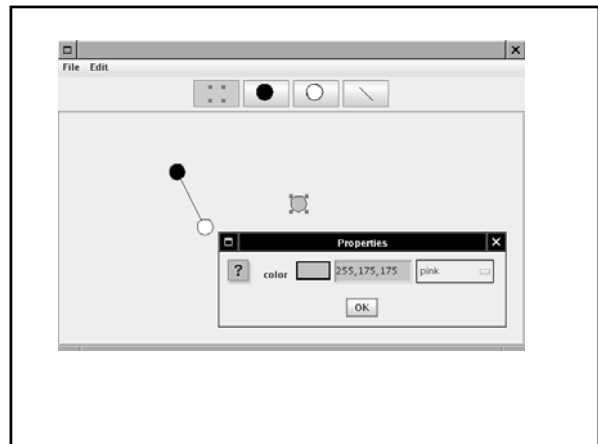
## New edges

- mousePressed:
  - Check if mouse point inside node
  - Check if current tool is edge
  - Mouse point is start of rubber band
- mouseDragged:
  - Mouse point is end of rubber band; repaint
- mouseReleased:
  - Add edge to graph



## Extending the framework

- Edit node/edge properties
  - Node colors
  - Edge styles (solid/dotted)
- Framework enhancement:  
Edit->Properties menu pops up property dialog



## How?

- How to implement the dialog?

## Idea

- Solved in chapter 7--bean properties!
- CircleNode exposes color property:  
`Color getColor()`  
`void setColor(Color newValue)`
- Property editor automatically edits color!

## Others

- Add dotted lines
- Define enumerated type LineStyle
- Two instances LineStyle.SOLID, LineStyle.DOTTED
- Add lineStyle property to LineEdge
- LineStyle has method getStroke()
- LineEdge.draw calls getStroke()
- Supply property editor for LineStyle type
- Property editor now edits line style!

## Next time

- Rest of chapter 8
- Will also start some advanced topics
- Will be releasing extra credit assignment