Announcements

• Project 3 Due:
  • Wednesday, Dec. 1 - 8:00am

Material

• More on reuse, inheritance, coupling
• Patterns: observer
Line Drawing App

- Create a simple application for drawing lines:
- Highlight current line being drawn
- Allow current line endpoint to be dragged with mouse

- Examine code
- Refactor code to provide line drawing for any application
LinesPanel

- User-defined subclass of JPanel
- Handles mouse events internally
- Depends on Vector and LineObject
- Restricted to single view

How do we design reusable components that are “extensible”?

- answer: abstraction
LinesPanel

- What services does it need from outside of its code?
  - Collection of end points (start-end)
  - Add to the collection
  - Know when to redraw
    - (if collection changed on its own accord)
  - Support for currently drawing line
    - rubberbanding of action

- Initial design:

<table>
<thead>
<tr>
<th>LinesModel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector lineObj</td>
</tr>
<tr>
<td>LineObject dragging</td>
</tr>
<tr>
<td>setDraggingLine()</td>
</tr>
<tr>
<td>getDraggingLine()</td>
</tr>
<tr>
<td>doneDragging()</td>
</tr>
<tr>
<td>addLineObject()</td>
</tr>
<tr>
<td>iterator()</td>
</tr>
</tbody>
</table>
• Use a LinesModel with a LinesPanel

LinesModel m = new LinesModel();
JPanel j = new LinesPanel(m);
JPanel p = new LinesPanel(m);  

• Internally, LinesPanel uses LinesModel to store the lines desired  

• Examine code
• Encapsulation
  LinesPanel as a stand alone class
  More reuseable

• Separation of data and code is good at times
  Can we separate the vector of objects from the method?
  If we do, we can have two panels with the same data, 2 views
  How to refresh the “other”?

• Limitation: how do LinesPanel objects know when the
  LinesModel is modified?
• LinesModel needs to let all views that are drawing this model update their representations

• Occurs twice
  • when mouse is moved
  • when lines are added

• How can the LinesModel tell objects outside of its bounds that things changed?
  • What have you seen already that follows this same behavior?

Observer Pattern
• The LinesModel class uses a LinesListener to notify “interested” parties that things have changed
  public interface LinesListener {
    public void refreshDragging();
    public void lineAdded(LineObject l);
  }

• Classes must register with the LinesModel
  public void addLinesListener(LinesListener ll)
New design

- Re-design

```
LinesModel
Vector lineObjs
LineObject dragging
setDraggingLine()
getDraggingLine()
doneDragging()
addLineObject()
iterator()
```

```
LinesModel2
Vector listeners
setDraggingLine()
addLineObject()
addLinesListener()
doneDragging()
```

```
<<interface LinesListener>>
- refreshDragging()
  lineAdded()
```

USES
Summary

• View of listeners from the inside
• No special code required
  • vector of objects
  • iterate over them and call them

• Extensibility of code is better
  • reuse with low coupling