Programming Languages and Techniques (CIS120)

Lecture 35

November 30th, 2015

Design Patterns
Model / View / Controller
Chapter 31
Game project grading

• Game Design Proposal Milestone Due:  (12 points)
  Tuesday December 1\textsuperscript{st} at 11:59pm
  – (Should take about 1 hour)

• Final Program Due:  (88 points)
  Tuesday December 8\textsuperscript{th} at 11:59pm
  – Submit zipfile online, submission \textit{only} checks if your code compiles

• Grade based on demo with your TA during reading days
  – Make sure that you test your program in Moore 100, especially if you use outside libraries
  – Grading rubric on the assignment website
  – Recommendation: don’t be too ambitious.

• \textbf{NO LATE SUBMISSIONS PERMITTED}
How is the Game Project going so far?

1. not started
2. got an idea
3. submitted design proposal
4. started coding
5. it's somewhat working
6. it's mostly working
7. debugging / polishing
8. done!
Course Trajectory

• Today: Thinking about software design at a larger scale
  – (Much more than we can cover)

• Java loose ends
  – Hashing and hashCode

• Touching on Java advanced concepts
  – Garbage collection
  – Concurrency
Design Patterns

• Design Patterns
  – Influential OO design book published in 1994
  – Identifies many common situations and "patterns" for implementing them in OO languages

• Some we have seen explicitly:
  – e.g. *Iterator* pattern

• Some we've used but not explicitly described:
  – e.g. The Broadcast class from the Chat HW uses the *Factory* pattern

• Some are workarounds for OO's lack of some features:
  – e.g. The *Visitor* pattern is like OCaml's fold + pattern matching
GUI Design Patterns

Model/View/Controller

Adapters
Example 1: Java Paint
Paint Program Structure

• Main frame for application (class Paint)
  – List of shapes to draw
  – The current color
  – The current line thickness

• Drawing panel (class Canvas, inner class of Paint)

• Control panel (class JPanel)
  – Contains radio buttons for selecting shape to draw
  – Line thickness checkbox, undo and quit buttons

• Connections between Preview shape (if any...)
  – Preview Shape: View <-> Controller
  – MouseAdapter: Controller <-> Model
Example 2: CheckBox

Views

CheckBox

Model

Selected?
Pressed?

true or false

Controllers

setSelected
mouseListener
keyListener

Class JToggleButton.ToggleButtonModel

boolean isSelected() Checks if the button is selected.
void setPressed(boolean b) Sets the pressed state of the button.
void setSelected(boolean b) Sets the selected state of the button.
Example 3: Chat Server

Views
- getChannels
- getUsers
- getOwner
... 

Model
- Internal Representation
  - owners: Map<Channel, Users>
  - users: Map<Channel, Set<Users>>
  ...

Controllers
- createChannel
- joinChannel
- invite
- kick
...

ServerModel
Example 4: Web Pages

- **Model**: Document Object Model (DOM)
- **View**: Model-view-controller
- **Controller**: JavaScript API

```
addEventListener()
```
MVC Pattern

Model

Updates

Manipulates

Controller

Adjusts

Uses

Sees

View(s)

User

Manipulates

Uses

Sees

View(s)

Model

Updates

Controller

Adjusts

User

Manipulates

Uses

Sees

View(s)
MVC Benefits?

• Decouples important "model state" from how that state is presented and manipulated
  – Suggests where to insert interfaces in the design
  – Makes the model testable independent of the GUI

• Multiple views
  – e.g. from two different angles, or for multiple different users

• Multiple controllers
  – e.g. mouse vs. keyboard interaction
MVC Variations

• Many variations on MVC pattern

• Hierarchical / Nested
  – As in the Swing libraries, in which JComponents often have a "model" and a "controller" part

• Coupling between Model / View or View / Controller
  – e.g. in Paint the Model and the View are quite coupled because the model carries most of the information about the view
Adapters

MouseAdapter
KeyBoardAdapter
Mouse Interaction in Paint

- **Point Mode**: Mouse Click (in the canvas) [add new point]
- **Line Start Mode**: Mouse Press [store point, set preview shape]
- **Line End Mode**: Mouse Drag [update preview]
- **Mouse Release**: [add new line, set preview to null]
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);
}

CIS120
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...

```java
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

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