CS1007: Object Oriented Design and Programming in Java

Lecture #11
Oct 27

Shlomo Hershkop
shlomo@cs.columbia.edu

1

Outline

- Review of midterm
  - Please feel free to ask if you don't see something covered.
- On to chapter 5

- Reading: Chapter 5-5.4.1
- Next time: 5.4.2-5.8

2

Midterm

- Please see me if you are concerned about your grade..office hours or email me for appointment.
- Some fundamental concepts missing
- Will try to review and will start to address them in the course

3

Public/Private classes

- Each .java file must contain one public/abstract class
- But can contain many private classes

- NOTE: if you got the private class marked wrong….please see me, I thought more people were aware of this fact.

4

Constructors

- Are called when you instantiate object
- If nothing defined will default to () empty constructor
- Never returns anything
- Can call other constructors IF
  - first line of constructor
- Usage:
  - this(argument list)

5

Return values

- Java will in some case make believe there are parenthesis around something

Example:

```
public boolean Something(int a,int b){
  return 5 + a == b;
  }
```

Problem:

```
public boolean Something(int a,int b){
  return 5 + a * 10  == b;
  }
```

6

Other issues with code example

- When to cast and when not
  - Explicit vs. implicit
- Can define many methods and not use them at all
  - Might be bad programming
  - Not error

7

3 way switch

- Simple class but had to keep track of current state
- Example:
  ThreeWaySwitch A = new ….
- ThreeWaySwitch B = new ..
  A.switch(5);
  B.swtich(1);

8

Exception handling

- If you are throwing an exception in the method, you need to make it part of the signature
  - This doesn't apply if it is contained…i.e. within and handled by try/catch block.
- Finally clause always handled (unless you kill the program).
  - Many people didn't understand the order of this.

9

Inheritance / Exception

- Order of catching exceptions important
  - More general should always be caught last
  - Else will have unreachable code.

10

Course plan – After Midterm

- Take home lesson: this is a chance to make sure you are on track….please see me with any concerns
- Will balance Object oriented design (correct way) with fundamentals (what you can use).
  i.e not just how to use a hammer correctly but also what kinds of hammers are available.

11

Patterns

- Many times when programming large projects:
  - Notice certain underlying patterns
  - Example: email file

    many different ways of representing email messages

    but: end user will want to treat them the same way!
  - Haha! A pattern

12

Iterator Pattern

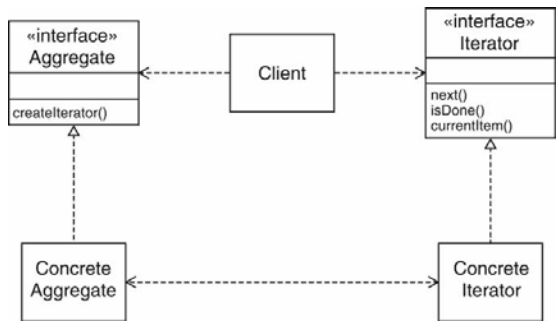
- Covered before midterm
- Collection of elements
- Users want to examine elements
- We don't want to expose the underlying implementation
- Ability to allow multiple independent access

13

Iterators pattern

- Define a general iterator that fetches one element at a time
- Each iterator object keeps track of the position of the next element
- If there are several collection/iterator variations, it is best if the collection and Iterator classes realize common interface types.

14



15

GUI programming

- Strong graphical component in java language.
- Very easy to create graphics and graphical components
- Because it is based on an Object Oriented Approach
- Again pattern based programming

16

Observer Patterns

- In many applications will have multiple views of the same data
- When you edit one part, affects other parts of the view
- Eclipse

17

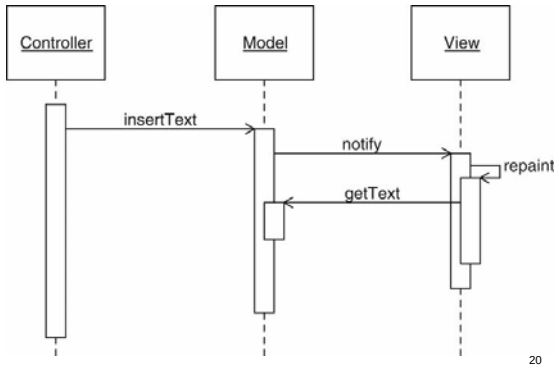
Division of Labor

- Model: data structure, no visual representation
- Views: visual representations
- Controllers: user interaction

18

- Views/controllers update model
- Model tells views that data has changed
- Views redraw themselves

19



20

Observer Pattern II

- Model notifies views when something interesting happens
- Button notifies action listeners when something interesting happens
- Views attach themselves to model in order to be notified
- Action listeners attach themselves to button in order to be notified
- Generalize: Observers attach themselves to subject

21

Consider

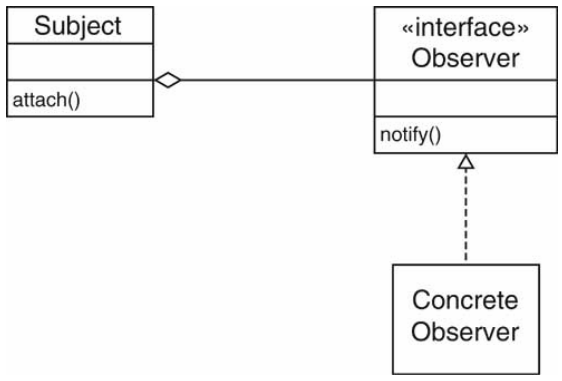
1. An object, called the subject, is source of events
2. One or more observer objects want to be notified when such an event occurs.

22

Idea

- Define an observer interface type. All concrete observers implement it.
- The subject maintains a collection of observers.
- The subject supplies methods for attaching and detaching observers.
- Whenever an event occurs, the subject notifies all observers.

23



24

Layout Managers

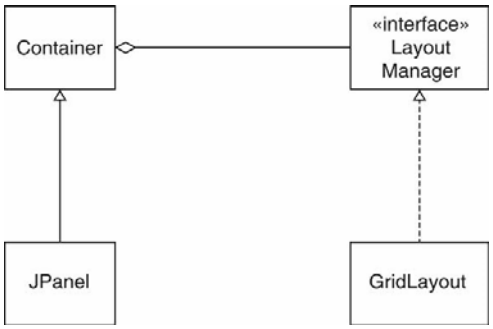
- Set layout manager

```
JPanel keyPanel = new JPanel();
keyPanel.setLayout(new GridLayout(4, 3));
```

- Add components

```
for (int i = 0; i < 12; i++)
    keyPanel.add(button[i]);
```

25



26

GUI for Voicemail system

- Same backend as text-based system
- Only Telephone class changes
- Buttons for keypad
- Text areas for microphone, speaker

27

28

Keys

- Arrange keys in panel with GridLayout:

```
JPanel keyPanel = new JPanel();
keyPanel.setLayout(new GridLayout(4, 3));
for (int i = 0; i < 12; i++)
{
   JButton keyButton = new JButton(...);
   keyPanel.add(keyButton);
   keyButton.addActionListener(...);
}
```

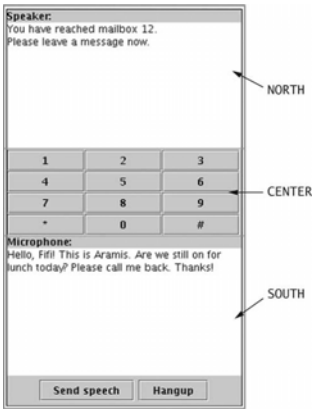
29

- Panel with `BorderLayout` for speaker

```
JPanel speakerPanel = new JPanel();
speakerPanel.setLayout(new BorderLayout());
speakerPanel.add(new JLabel("Speaker:"),
  BorderLayout.NORTH);
speakerField = new JTextArea(10, 25);
speakerPanel.add(speakerField,
  BorderLayout.CENTER);
```

30

31



32

- Ch5/code/mailgui/Telephone.java.html

33

Custom Layouts

- Form layout
- Odd-numbered components right aligned
- Even-numbered components left aligned
- Implement `LayoutManager` interface type

34

LayoutManager Interface

```
public interface LayoutManager
{
   void layoutContainer(Container parent);
   Dimension minimumLayoutSize(Container parent);
   Dimension preferredLayoutSize(Container parent);
   void addLayoutComponent(String name, Component
  comp);
   void removeLayoutComponent(Component comp);
}
```

35

Form Layout

- Ch5/layout/FormLayout.java
- Ch5/layout/FormLayoutTester.java
- Note: Can use GridBagLayout to achieve the same effect

36

Plan

- Pluggable strategy for layout management
- Layout manager object responsible for executing concrete strategy
- Generalizes to Strategy Design Pattern
- Other manifestation: Comparators

```
Comparator<Country> comp = new CountryComparatorByName();
Collections.sort(countries, comp);
```

37

Objective

1. A class can benefit from different variants for an algorithm
2. Clients sometimes want to replace standard algorithms with custom versions

38

Solution

- Define an interface type that is an abstraction for the algorithm
- Actual strategy classes realize this interface type.
- Clients can supply strategy objects
- Whenever the algorithm needs to be executed, the context class calls the appropriate methods of the strategy object

39

In short

- PLUG AND PRAY

40

Next Time

- Do Reading (through 5.8)
- Review midterm to make sure you understand what went wrong.

41