Midterm

• Hope you had some fun taking it
• Will review it in class (if you couldn’t make it to class, please grab someone’s notes…if you are in class, please take notes)
• Will try to review fundamental concepts in each question.
midterm

• Try to balance your time, and topics

• No reason to ask about random stuff we haven't touched

• No reason to ask stuff you will get wrong

• No power rush from tricking students

• These exams are a lesson for everyone

Review Midterm

• Question 1

• General definitions

• Looking for specific point, hopefully your answer
  – Legible
  – Correct
  – You understand what you wrote 😊
Question 2

• Understanding packages

• Understanding directory relationships to packages

• Picking up on final keyword in method definition

Q3

• == vs. .equals

• One compares address of referenced objects

• One compares contents of specific object

• Challenge: Can 2 different classes get a true for the == operation?
Interfaces question

• Interface definition

• Use of interface

• Different between the two compare interfaces

• Why both?

Question 5

• Understanding method chaining

• Will walk/talk through the code
Copy question

- Related to references
- Pretty much covered in the course

Recursion question

- Very open ended
- What ideas did you use?
idea

• Base case:
  • 1) a==b
    – Return a
  • 2) Need to check a<b
    – Return 0

• Else
• Return a + sum(a+1,b-1) + b

How to do it better?
Call helper method

• Can use more args 😊

• Would carry around the sum in third arg

• Help(a, b, 0)

• Recursion: Help(a+1, b-1, a+b)

Random issues: Return values

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

Example:

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

Problem:

```java
public boolean Something(int a, int b) {
    return 5 + a * 10 == b;
}
```
Random II: return live iterator

• Most of the time, when you consider Iterator, think of implimented class
• Can also return an anonymous inline Iterator:

• Code:
  – Represent a group of items (will use an array)
  – Give back an iterator on request

Random III: 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.
Bottom line

• Will be returning the exams next class
• Will post grades on to courseworks

• Please stop by OH to discuss any concerns etc you might have

• Make sure to get started early on the next homework (released by next class)

Post midterm

• Will be covering:
  – Start chapter 5
    • Patterns of programming
    • Object design and considerations
    • Many examples
    • Application to homework: writing Othello game
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

Patterns

• Advantage:
  – If can group what you are doing as a specific pattern
    • Can reuse related patterns (code)
    • Can study for optimization purposes…one discovery can be easily applied to many different types of stuff, if we’ve grouped them into patterns
    • Easier to think about huge projects if we have a few patterns to talk about

  • Seemed to have worked in architecture
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

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.
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
Division of Labor

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

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

• Will cover more examples with catch/finally
• Will cover more coded examples with iterators
• Will cover observer examples/code

• Read chapter 5.1-5.6