Designing Classes
More Practice with Conditionals

Lecture 13
Some slides borrowed from Kurt Eiselt and Beth Simon.

Readings
This Week’s Reading: Review Ch 1-4
(that were previously assigned)

(Reminder: Readings are absolutely vital for learning this stuff!)

Labs and Tutorials
This week is Lab #4.

Next week is Lab #5! (Some on-line material had out-of-date schedule – fixed now.)

Midterms – Save the Dates!
- Midterm #1 is 5:30-6:30pm on February 10 (Tuesday) in Woodward IRC 2
- Midterm #2 is 6-7pm on March 11 (Wednesday) in Woodward IRC 2

Midterm Study Tips:
- Old midterms on-line.
- Try programming without notes.
- Try programming without computer!

Programming Assignment 1
- Assignment 1 is up on WebCT!
  - Click on the “Assignments” icon.
  - Due at NOON, February 17 (Tuesday), via electronic hand in.
    - It may take me a couple days to setup the electronic hand in, so if you’re really fast, please wait a day or two.
  - Start early!

Learning Goals
By the end of the next several lectures you will be able to...
- Create your own classes, with:
  - Public and private fields and methods
  - Helpful documentation that works with javadoc
  - Basic principles of abstraction and encapsulation (information hiding)
- Explain why abstraction and information hiding are important.
Learning Goals

By the end of class today you will be able to...

- Methodically approach the task of designing a class.
- Have more confidence creating your own classes.

Designing a Die Class

- Let’s create a class to represent a die (as in rolling dice, not other meanings of “die”):
  - What should we call it?
  - What methods should we provide?
    - What sort of things do you do to a die?
  - What instance variables (or attributes, fields) do we need?
  - How do we test it? What should we make sure we can do?

Class Design

- Two types of things in every class:
  - things we KNOW about that object
  - thing we can DO to/with that object
- This is how you start designing a class

Designing a Die Class: Let’s do it...

- Let’s create a class to represent a die (as in rolling dice, not other meanings of “die”):
- Design before you implement.
  - You might have to adjust your design a bit as you implement, but that’s OK.
  - Use UML if you want, or just variable declarations and method headers.
- What do we KNOW about a die? (attributes)
- What do we DO with a die? (methods)
- What should our tester try?

Implementing Die

```java
/**
 * Provides a simple model of a die (as in pair of dice).
 */
public class Die {
    /*
   竞争优势
    */
}
```

Random Numbers

- Random class in java.util package
  - public Random()
    - Constructor
  - public float nextFloat()
    - Returns random number between 0.0 (inclusive) and 1.0 (exclusive)
  - public int nextInt()
    - Returns random integer ranging over all possible int values
  - public int nextInt( int num )
    - Returns random integer in range 0 to (num-1)
Trying It Out

- Die class has no main method.
- Best is to write another class that instantiates some objects of your new class and tries them out.
- Sometimes called a “tester” or “testbench”

Implementing RollDice

```java
public class RollDice {
    public static void main(String[] args) {
    }
}
```

Questions?

Mileage Computer

- Design a class for a trip computer in a car that computes the gas mileage
  - Every 1m, a distance sensor sends a signal to the computer.
  - Every 1 second, a fuel sensor sends a signal to the computer indicating how many ml of gas were used during the last second.
  - The measurements can be reset
  - We can query the computer for the gas mileage (reported as l/100km) averaged since the last reset.

Let’s do it...

- Design before you implement.
  - You might have to adjust your design a bit as you implement, but that’s OK.
  - Use UML if you want, or just variable declarations and method headers.
- What do we KNOW about a die? (attributes)
- What do we DO with a die? (methods)
- What should our tester try?

Mileage Computer

- Design a class for a trip computer in a car that computes the gas mileage
  - Every 1m, a distance sensor sends a signal to the computer.
  - Every 1 second, a fuel sensor sends a signal to the computer indicating how many ml of gas were used during the last second.
  - The measurements can be reset
  - We can query the computer for the gas mileage (reported as l/100km) averaged since the last reset.
Updated Specs: Mileage Computer
- Design a class for a trip computer in a car that computes the gas mileage
- Every 1m, a distance sensor sends a signal to the computer.
- Every 1 second, a fuel sensor sends a signal to the computer indicating how many ml of gas were used during the last second.
- The measurements can be reset
- We can query the computer for the gas mileage averaged since the last reset.
- The computer can be configured in two modes: English or Metric. In Metric, it reports mileage as l/100km. In English, it reports mileage as mpg.

How do we change our design? Implementation? Tests?

Questions?

Updated Specs 2: Mileage Computer
- Design a class for a trip computer in a car that computes the gas mileage
- Every 1m, a distance sensor sends a signal to the computer.
- Every 1 second, a fuel sensor sends a signal to the computer indicating how many ml of gas were used during the last second.
- The measurements can be reset
- We can query the computer for the gas mileage averaged since the last reset.
- We can query the computer for the “current” gas mileage that was achieved during the previous 1 second period.
- The computer can be configured in two modes: English or Metric. In Metric, it reports mileage as l/100km. In English, it reports mileage as mpg. The user can switch modes.

How do we change our design? Implementation? Tests?