Readings
This Week: Ch 5.1-5.4 (Ch 6.1-6.4 in 2nd ed).
(Reminder: Readings are absolutely vital for learning this stuff!)

Labs and Tutorials
This week is Lab #5.

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.
- Explain what “abstraction” is, and why it’s important.
- Explain what “information hiding” (also known as “encapsulation” or “data protection”) is, and why it’s important.

---

# Review: Designing a Die Class

- 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?

---

# Review: 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 litres/100km) averaged since the last reset.

---

# Design Is Creative

- There are many ways to think about a problem.
  - There are usually many good ways to think about a problem.
  - If someone else sets up a problem differently than you would, it takes some adjusting...
  - Even after setting up a problem, there can be many ways to solve it, too.

---

# Mileage Computer

- In my design, the mileage computer object is passive:
  - Mileage Computer

---

# Mileage Computer

- In my design, the mileage computer object is passive:
  - Mileage Computer
  - “Reset yourself!”
In my design, the mileage computer object is passive:

- Distance sensor says we’ve gone 1 meter!

- Fuel sensor says we’ve used 1ml of gas in the last second.
In my design, the mileage computer object is passive:

- Distance sensor says we’ve gone 1 meter!
- Fuel sensor says we’ve used 2ml of gas in the last second.
- What’s my average mileage been?
- You’ve been averaging 100 litres per 100km.
Mileage Computer

- There are many other ways to approach this problem, e.g., an active object that queries the others...

```
Hey how far have we gone?
```

```
You've been averaging 100 litres per 100km.
```

```
Hey, how much fuel have we used?
```

```
```

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 the objects? (instance variables/attributes)
  - What do we DO with with/to the object? (methods)
  - What should our tester try?

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 (reported as litres/100km) averaged since the last reset.

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 (reported as litres/100km) 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 English, it reports mileage as mpg. In Metric, it reports mileage as l/100km. The user can switch modes.

How do we change our design? Implementation? Tests?
Questions?

Managing Complexity
- Computer science creates the most complex artifacts ever created by humans:
  - Windows Vista has 50 million lines of code.
  - Current mainstream Intel processors (Core 2 Duo) have 410 million transistors; quad-core Core i7 chips have 731 million transistors.
- How do we manage to do this?
- Answer: Abstraction

Abstraction
- Abstraction means creating higher-level ways to think about things, so you can ignore lots of lower-level details.

Abstraction in Real Life
- Abstraction means creating higher-level ways to think about things, so you can ignore lots of lower-level details.
- You already do this all the time to manage complexity:
  - How do you get home after school?
  - How do you manage your time in a day?
  - How do you learn biology or chemistry or music or literature or …?

Abstraction in Software
- Abstraction means creating higher-level ways to think about things, so you can ignore lots of lower-level details.
- Creating new classes is a common and powerful way to create new abstractions:
  - How do you print information to the console window?
  - How do you make a Java program take control of the mouse?
  - How do you compute a logarithm?

Encapsulation
- Why is the gas cap on the outside of a car, while the places to add oil, coolant, brake fluid, etc., are under the hood?
- Why are the on/off, channel, and volume switches of a TV on the front, while other controls are hidden away?
- Why is the keyboard on a laptop obvious, while the jumpers to configure the disk drive are hidden inside?
Encapsulation

- A car, a TV, a laptop, these are all abstractions!
- They were intended by their creators to be used in certain ways, and not in other ways:
  - My mom adding water to the engine oil filler.
  - Phone phreaking
  - The designer encapsulates the product to hide the details, to make it harder to misuse.

Encapsulation, Information Hiding, Data Protection

- In computer software, the same idea is also called data protection or information hiding.
- Parnas’s Law: Only what is hidden can be changed without risk:
  - If I change problem 2 of the midterm, does that upset you?
  - If I move the midterm to be RIGHT NOW instead, does that upset you?

Encapsulation, Information Hiding, Data Protection

- In computer software, the same idea is also called data protection or information hiding.
- Parnas’s Law: Only what is hidden can be changed without risk:
  - If Java 7 implements System.out.println slightly differently, does that upset people?
  - If Java 7 changes the name of System.out.println to be System.out.writeln, does that upset people?

Encapsulation, Information Hiding, Data Protection

- By making the instance fields private, we protect them from misuse.
  - They are accessible to others only through the public methods that we provide.
  - If we write these methods carefully, we can help prevent mistakes.

Designing a Class

- You want to provide a useful abstraction. It should let the user think about higher-level things, without worrying about the details.
- You use encapsulation to prevent other programmers (or yourself!) from misusing your class.

Questions?
Commenting Code

- Conventions
  - explain what classes and methods do
  - plus anywhere that you've done something non-obvious
  - usually better to say why than what
    - not useful
      - int wishes = 3; // set wishes to 3
    - useful
      - int wishes = 3; // follow fairy tale convention

---

javadoc Comments

- Specific format for method and class header comments
  - running javadoc program will automatically generate HTML documentation
- Rules
  - /** to start, first sentence used for method summary
  - @param tag for parameter name and explanation
  - @return tag for return value explanation
  - other tags: @author, @version, etc.
  - */ to end
- Running
  - % javadoc Die.java
  - % javadoc *.java

---

javadoc Method Comment Example

```java
/** Sets the die shape, thus the range of values it can roll. 
 * @param numSides the number of sides of the die 
 */
public void setSides(int numSides) {
    sides = numSides;
}

/**
 * Gets the number of sides of the die.
 * @return the number of sides of the die
 */
public int getSides() {
    return sides;
}
```

---

javadoc Class Comment Example

```java
/** Die: simulate rolling a die 
 * @author: CPSC 111, Section 206, Spring 05-06 
 * @version: Jan 31, 2006 
 * @ This is the final Die code. We started on Jan 24, 
 * tested and improved in on Jan 26, and did a final 
 * cleanup pass on Jan 31. 
 */
```