Lecture 11

Some slides borrowed from Kurt Eiselt and Beth Simon.

Readings
This Week:  Ch 5.1-5.4 (Ch 6.1-6.4 in 2nd ed).
Next Week:  Ch 6 (Ch 7 in old 2nd ed).

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

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.
- 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...
- 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.
- Use javadoc to generate nice-looking documentation webpages.
- Write complicated, nested if statements.
- Write programs with simple loops.

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.

If you want, try doing “updated” specifications in my slides.

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 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 that can compute with arbitrarily big numbers?
  - How do you compute a logarithm?

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 …?
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 on the final, does that upset you?
  - If I move the final 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 `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

/**
 * 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

/**
 * 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.
 */

Questions?
Java's if statement

- Syntax:
  
  ```java
  if ( condition ) statement;
  ```

- Examples:
  
  ```java
  if (tax < 0) tax = 0;
  ```

  ```java
  if (age < 19)
    System.out.println("Sorry, you may not buy alcohol.");
  ```

Intuitively, a lot like English:

If you’re hungry, have a sandwich.

```java
if (tax < 0) tax = 0;
```
Nested If Syntax

- Statements within if-else statements can themselves be if-else statements.
- Use curly braces to be clear.

```java
public class NestTest {
    public static void main(String[] args) {
        int x = 1; int y = 3; int z = 2;
        if (x == y) {
            if (y == z) {
                System.out.println("all three values the same");
            } else {
                System.out.println("y is not equal to z");
            }
        } else {
            System.out.println("x is not equal to y");
        }
    }
}
```

Body Mass Index (BMI) Calculator

- The Body Mass Index (BMI) is a standard medical test to give a quick and approximate indicator of healthy body weight.
- Your BMI is defined as your “weight” in kilograms divided by the square of your height in meters.
  - BMI < 19: Underweight, higher risk of disease
  - 19 <= BMI <= 25: Normal
  - 25 < BMI <= 30: Overweight, higher risk
  - 30 < BMI: Obese, much higher risk of disease

Nested if Syntax

- A common solution (others are possible):
  ```java
  if (bmi < 19)
  { // underweight statements
  } else if (bmi <= 25)
  { // normal statements
  } else if (bmi <= 30)
  { // overweight statements
  } else
  { // obese statements
  }
  ```

Easier to read indented like this:
```java
if (bmi < 19)
{ // underweight statements
} else if (bmi <= 25)
{ // normal statements
} else if (bmi <= 30)
{ // overweight statements
} else
{ // obese statements
}
```

Dangling else

- What does this code do?
  // print warnings only if BMI not normal
  if (bmi >= 19)
      if (bmi > 25) System.out.println("Over");
  else System.out.println("Under");

What does this code do?
// print warnings only if BMI not normal
if (bmi >= 19)
    if (bmi > 25) System.out.println("Over");
    else System.out.println("Under");
else goes with nearest if
- (but doesn’t look inside curly braces)
- common mistake
- whitespace ignored by compiler

Beware the Dangling else
Use Curly Braces to Control Nesting

- What does this code do?

```java
// print warnings only if BMI not normal
if (bmi >= 19) {
    if (bmi > 25) System.out.println("Over");
} else System.out.println("Under");
```
- `else` goes with nearest `if`
- but doesn't look inside curly braces

---

Java (and many) programming languages have 4 basic "control flow" options

- **Sequential**
  - Each statement is executed when the statement before it (above it) is completed. ("top to bottom")
- **Conditional (if statements switch statements)**
  ```java
  if (somethingPossiblyTrue) {
      A set of Java statements to do ONLY if true
  }
  ```
- **Method call**
- **Iteration**
  - Do this set of statements until* do while
  - Do this set of statements while while
  - Do this set of statements X times for

*used a lot in English, much less in programming

---

What's Iteration?

- **Iteration** is just a fancy word for repeating something over and over again.
- In CS, we also call this **looping** because a loop goes around and around over and over again.
- This is an incredibly cool concept, because it means you can make the computer do way more work than you type!

---

Iteration in Daily Life

- Beth’s Example: Washing Dishes – While there are more dishes to wash, wash another item.
- Kurt’s Example: Climbing Stairs – While there’s farther up to go, climb one more step.
- Alan’s Example: Eating Junk Food – While you’re still hungry, have some more fries.
Climbing Stairs

- Am I at the top of the stairs?
- No.
- Climb up one step.
- Am I at the top of the stairs?
- No.
- Climb up one step.
- Am I at the top of the stairs?
- No.
- Climb up one step.
- Am I at the top of the stairs?
- No.
- Climb up one step.
- Am I at the top of the stairs?
- No.
- Climb up one step.
- Am I at the top of the stairs?
- No.
- Climb up one step.
- Am I at the top of the stairs?
- No.
- Climb up one step.
- Am I at the top of the stairs?
- No.
### while Statement

- **while** *(boolean expression)*
  - **body**

- **Simplest form of loop in Java**
- **Body of loop can be**
  - single statement
  - whole block of many statements in curly braces
- **Meaning is kind of like English:**
  
  "While you’re still hungry, have some more fries."
  
  while (still hungry) {
  
  eat fries;
  
  }

### Using while Statements

```java
public class WhileDemo {
    public static void main (String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter <= limit) {
            System.out.println("The square of " + counter + ", is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

**while statement**

### Using while Statements

```java
public class WhileDemo {
    public static void main (String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter <= limit) {
            System.out.println("The square of " + counter + ", is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

**while statement body**

### Using while Statements

```java
public class WhileDemo {
    public static void main (String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter <= limit) {
            System.out.println("The square of " + counter + ", is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

**statement after while**

- control flow resumes here when boolean is false

### Control flow

- Is boolean expression true? If not, exit loop.
- Execute body of loop.
- Check again, is boolean expression still true? If not, exit loop.
- Execute body of loop.
- … and so on …
- Repetition continues until expression false.
- Then processing continues with next statement after loop
Using while Statements

```java
public class WhileDemo {
    public static void main (String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter <= limit) {
            System.out.println("The square of " + counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

trace what happens when execute

```
limit 3 counter 1
```

```
Is counter <= limit? yes
```

```
"The square of 1 is 1" printed on monitor
```

```
limit 3 counter 2
```

```
Is counter <= limit? yes
```

public class WhileDemo
{
    public static void main (String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter <= limit)
        {
            System.out.println("The square of "+ counter + " is "+ (counter * counter));
            counter = counter + 1;
            System.out.println("End of demonstration");
        }
    }
}

public class WhileDemo
{
    public static void main (String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter <= limit)
        {
            System.out.println("The square of "+ counter + " is "+ (counter * counter));
            counter = counter + 1;
            System.out.println("End of demonstration");
        }
    }
}

public class WhileDemo
{
    public static void main (String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter <= limit)
        {
            System.out.println("The square of "+ counter + " is "+ (counter * counter));
            counter = counter + 1;
            System.out.println("End of demonstration");
        }
    }
}

public class WhileDemo
{
    public static void main (String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter <= limit)
        {
            System.out.println("The square of "+ counter + " is "+ (counter * counter));
            counter = counter + 1;
            System.out.println("End of demonstration");
        }
    }
}
Using while Statements

```java
class WhileDemo {
    public static void main(String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter <= limit) {
            System.out.println("The square of " + counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

Example:
- `limit = 3`
- `counter = 4`
- Is `counter <= limit`? NO!
- "End of demonstration" printed on monitor

More Work Than You Typed!

```java
class WhileDemo {
    public static void main(String[] args) {
        int limit = 100;
        int counter = 1;
        while (counter <= limit) {
            System.out.println("The square of " + counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

Example:
- The same length program does more work!

What if boolean expression is already false?

```java
class WhileDemo {
    public static void main(String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter >= limit) {
            System.out.println("The square of " + counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

Example:
- change termination condition

What if boolean expression is always true?

```java
class WhileDemo {
    public static void main(String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter > limit) {
            System.out.println("The square of " + counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

Example:
- change termination condition
- body of loop never executed
- always true
Infinite Loops

```java
public class WhileDemo {
    public static void main (String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter >= counter) {
            System.out.println("The square of "+ counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

- if termination condition always true, loop never ends
- infinite loop goes forever

Questions?

```java
public class WhileDemo {
    public static void main (String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter <= limit) {
            System.out.println("The square of "+ counter + " is " + (counter * counter));
            counter = counter - 1;
        }
        System.out.println("End of demonstration");
    }
}
```

- good termination condition
- but process never gets closer to condition

Infinite Loops

```java
public class WhileDemo {
    public static void main (String[] args) {
        int limit = 9;
        int counter = 0;
        while (counter != limit) {
            System.out.println("The square of "+ counter + " is " + (counter * counter));
            counter = counter + 2;
        }
        System.out.println("End of demonstration");
    }
}
```

- process gets closer to termination condition
- but never satisfies condition, keeps going past it

Midterm Info

- Out of 100: raw avg=57.1, low=11, high=100
  - Will be scaled (info later)!
- Tough exam, but not unreasonable:
  - Everyone should be able to do it (with time).
  - Talk to friends, learning centre, WebCT, TAs, prof.
- Solutions will be posted.
- Protocol for re-mark requests:
  - read solutions first, carefully
  - no re-mark requests accepted until Monday
  - re-mark requests must be in writing (paper attached to exam) and submitted to instructor
  - entire exam re-marked

Do you want to “do better” in this course?

- Suppose you aren’t satisfied with your learning or your marks in this course to date.

What are you going to do differently from now on to improve your learning and/or your grade?
What the research says

- In a study of Biology students in a genetics course, students who had specific plans for improving after the first midterm did better in the class than those with "general" plans.
- **NOT GOOD:**
  - "I'm going to turn things around."
  - "I'm going to dig myself out of this bad mark."
- **BETTER:**
  - "I'm going to revise my notes for at least 10 minutes after each class and before the next class."
  - "I'm going to read the textbook pages before class."
  - "I'm going to ask questions of the professor or TA."


Problems with General Goals

- 1) You still have to figure out what you should actually be doing to accomplish that goal.
  - What will you do each day or week to "turn it around"?
- 2) It's hard to get feedback to let you know if you are making any progress on that.
  - What kind of thing that you control will "tell you" if you are successfully "turning it around"?
  - The final exam? That's a bit late...

Benefits of Specific Goals

- 1) The goal itself tells you exactly what you should be doing to accomplish it.
- 2) You can get feedback on your own to determine if you are meeting your goal.
  - The goal should be measurable or countable.
  - You can make a log (on paper, on the wall, in email, whatever) that shows the work that you did to accomplish the goal.

But you knew that...

- Which person do you think is going to improve their fitness the most?
  - **JoeGeneral:**
    - "Gee, I'm really getting out of shape, I need to be more active."
  - **SusanSpecific:**
    - "Gee, I'm really getting out of shape, I'm going to hit the gym for at least 30 minutes, MWF. I'm going to do cardio MF and lift weights on W."

Specific, measurable goals

- Are more likely to produce actual results.
  - Sports teams have highly specialized, specific training programs.
- Describe something you can do
  - you think addresses a deficiency in your learning or understanding
- Identify how you will know if you have done it
  - What the "evidence" will be
- When you are going to do it
- Keep a record (maybe email a friend)

Possible Goals for Beginning Computing Students

- I'm going to solve on paper (and then type in) the problems at the end of the section/chapter.
- I'm going to type up and play around with changes in the code provided in class.
- I'm going to explain out loud what some code does (homework code, code from class, etc.) - possibly to someone else.
- After each class, I am going to discuss for 20 minutes with another student what I thought was important from class and what I am confused about.
Possible Goals Suitable for Science Classes

- I'm going to read the book before class and write down two questions I have about the reading.
- I'm going to look at all the figures and diagrams in the text and write down at least two things that are important about each one.
- After each class, I am going to discuss for 20 minutes with another student what I thought was important from class and what I am confused about.
- I am going to review and paraphrase my notes after each class.

Midterm #1

- Raw Score Stats:
  - Average: 57.1, Low: 11, High: 100 (Out of 100)
- Scaling Formula:
  \[(\text{raw}/100)^{0.7} \times 100\]

<table>
<thead>
<tr>
<th>Raw Score</th>
<th>Scaled Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>27</td>
</tr>
<tr>
<td>30</td>
<td>43</td>
</tr>
<tr>
<td>45</td>
<td>57</td>
</tr>
<tr>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>75</td>
<td>82</td>
</tr>
<tr>
<td>90</td>
<td>93</td>
</tr>
</tbody>
</table>