```
                           CS 4:  Lecture 6
                       Monday, February 6, 2006

LOOPS
=====
To do anything really interesting with a computer, you need a way to repeat an
operation over and over.  A _loop_ is a code sequence that is executed
repeatedly.

"while" Loops
-------------
Java has several types of loops.  The simplest is called a "while" loop.

  long i = 1;
  while (i <= 4) {
    System.out.println(i);              \  These lines inside the braces
    i++;                                /  are called the _loop_body_.
  }

The expression "i <= 4" is the _loop_condition_.  It must always be a boolean.

Here’s how the loop executes.
- When Java reaches this "while" loop, it checks whether "i <= 4" is true.
- If i <= 4, Java executes the loop body (the code inside the braces).
- When Java finishes the code inside the braces (i.e. after executing "i++"),
  it checks _again_ whether "i <= 4" is true.
- If it’s still true, Java jumps back up to the beginning of the loop body and
  executes it again.
- If Java reaches the end of the loop body and "i <= 4" is false, Java will
  continue execution with the next line of code _after_ the loop body.

So long as "i <= 4" is true, Java repeatedly executes the code inside the
braces.  Java cannot exit the loop until i > 4.

Take a few minutes to pretend that you’re the Java intepreter.  Walk through
the code line by line, in the order Java executes it, and execute each line
yourself.  To learn how to program, you will need the ability to pretend you’re
Java and execute a program just like Java does.

The code prints the following output.

  1
  2
  3
  4

When Java finally leaves the loop, the variable i contains a 5.

An _iteration_ is a pass through the loop body.  In this example, Java executed
four iterations of the loop body.

If the loop condition is false when Java first reaches the "while" loop, the
loop body doesn’t execute even once.  So the following code prints nothing.

  long i = 8;
  while (i <= 4) {
    System.out.println(i);
    i++;
  }
```

```
"do" Loops
----------
A "do" loop has just one difference from a "while" loop.  If Java reaches
a "do" loop, it _always_ executes the loop body at least once.  Java doesn’t
check the loop condition until the end of the first iteration.

Let’s look at a program that repeatedly reads a string from the keyboard, then
prints it back out.  For now, just look at the "do" loop; I’ll explain the rest
shortly.

  import java.io.*;

  class SimpleIO {
    public static void main (String[] args) throws Exception {
      BufferedReader keybd =
            new BufferedReader(new InputStreamReader(System.in));
      String input;

      do {
        input = keybd.readLine();
        System.out.println("You typed " + input);
      } while (!input.equals("stop"));
    }
  }

The body of the "do" loop does two things.  First, it reads a String from the
keyboard.  The "readLine" method waits for you to type a String at the console;
when you hit the "Enter" key, the program can continue.  Second, the "println"
method prints the String you typed in.

The loop condition, ‘!input.equals("stop")’, is true if you did _not_ type
"stop" at the keyboard.  The "!" operator, or _not_ operator, changes a true to
a false or a false to a true.  If you typed any string other than "stop", Java
will jump back to the beginning of the loop body and wait for you to type in
another String.  If you type "stop", then Java prints "stop", exits the loop,
and ends the program.

It makes sense to use a "do" loop instead of a "while" loop here, because the
variable "input" is not initialized until the loop body executes.

Input Classes
-------------
What about the other stuff in that program?  Java has some objects in the
System class for interacting with a user.

  System.out is a PrintStream object that outputs to the screen.
  System.in is an InputStream object that reads from the keyboard.

Reminder:  this is shorthand for "System.in is a variable that references an
InputStream object."

But System.in doesn’t have methods to read a line directly.  InputStream
objects (like System.in) read raw bytes from some source (like the keyboard),
but don’t format the bytes.  InputStreamReader objects compose the raw bytes
into characters (which are typically two bytes long in Java).  BufferedReader
objects compose the characters into entire lines of text, and have a method
called "readLine" that reads and returns one.  Why are these tasks divided
among three different classes?  So that any one task can be reimplemented
(say, for improved efficiency) without changing the other two.

The first line, "import java.io.*;", is present because the InputStreamReader
and BufferedReader classes are defined in a standard Java library called
java.io .  To use the Java libraries, other than java.lang, you need to
"import" them.
```

You could figure all of this out by looking at the constructors in the online Java libraries API--specifically, in the java.io library. The online API is linked from the class Web page, and will show you tons of methods you can use to do a lot of work for you.

Nested Loops

A loop is _nested_ if it's inside another loop. Here's a nested loop that computes the divisors of each number between 2 and 9 (except the trivial divisors: 1 and the number itself).

```
    int i = 2;
    while (i <= 9) {
      int j = 2;
      while (j < i) {
        if (i % j == 0) {
          System.out.println(i + " is divisible by " + j);
        }
        j++;
      }
      i++;
    }
```

The output is:

```
    4 is divisible by 2
    6 is divisible by 2
    6 is divisible by 3
    8 is divisible by 2
    8 is divisible by 4
    9 is divisible by 3
```

Boolean operators

We've seen the boolean operators in lab: &&, called "and"; ||, called "or"; and !, called "not". These operators compute boolean values from boolean values, according to the following _truth_table_.

a	b	a && b	a || b	!a
false	false	false	false	true
false	true	false	true	
true	false	false	true	false
true	true	true	true	

These operators are handy in "if" and "while" statements.

```
  if (income > 7300 && income <= 29700) {
    taxrate = 0.15;
  }
```

It's important not to take the names "and" and "or" too literally. For example, in English we can say, "If your income is over $7,300 AND under $29,700, your marginal tax rate is 15%." But you can't write the analog in Java:

```
  if (income > 7300 && <= 29700) {           /* Compile-time error! */
```

So don't think of && as a literal, English "and". Instead, think of "&&" (and "||") as a mathematical operation that takes two booleans and returns a boolean.