Labs and discussions sections start(ed) this week. Get an account (if needed) and register electronically this week.
Go to any sections, labs where you fit.
Class web page and newsgroup set up: read them regularly!
Concurrent enrollment students: bring me your forms.
Readers will be coming from one of the local copy stores (we’ll announce).
For Friday, read Chapters 1–4 of Head First Java.

**Course Organization**

- You read; we illustrate.
- Labs are important: practical dirty details go there.
- Homework is important, but really not graded: use it as you see fit and turn it in!
- Individual projects are really important! Expect to learn a lot.
- Use of tools is part of the course. Programming takes place in a programming environment:
  - Handles program editing, debugging, controlling compilation, archiving versions.
  - We’ll see Eclipse in lab.
  - Or there are coordinated suites of tools (e.g., Emacs + gjdb + make + cvs).
- Tests are challenging: better to stay on top than to cram.
- Tests, 90%; Projects, 90%; HW, 20%
- Stressed? Tell us!
- Now’s your opportunity to decide.

**Programming, not Java**

- Here, we learn programming, not Java (or Unix, or NT, or...)
- Programming principles span many languages
  - Look for connections.
  - Syntax \((x+y) vs. (+ x y)\) is superficial.
  - E.g., Java and Scheme have a lot in common.
- Whether you use GUIs, text interfaces, embedded systems, important ideas are the same.

**Really simple example**

```java
public class Greet {
    /** Print a greeting message on standard output. */
    public static void main (String[] args) {
        System.out.print ("Hello, ");
        if (args.length > 0)
            System.out.println (args[0]);
        else
            System.out.println ("");
    }
}
```

% javac -g Greet.java       # Creates Greet.class
% java Greet world          # Interpreter calls Greet.main
Hello, world                # Output
% java Greet me warmly      # Another run
Hello, me                   # args[0] = "me"
Lessons from Simple Example

- All definitions are inside some class.
- Syntax $A.B$ means "the $B$ that is defined (or contained) inside $A,"$
  - E.g., System.out.println, Greet.main
- Ordinary function is static method, like Greet.main.
- Methods declare what kinds (types) of arguments they take, and what kind of value they return (void means "no value").
- Method calls use familiar prefix syntax.
- Command-line arguments become an array of strings.
- Array is indexed sequence: args[0], args[1], ..., args[args.length-1]
- Conditional statement: if (condition) ...else ....
- Access control: public and others control what parts of the program may use a definition.

Prime Numbers

Problem: want java PrintPrimes0 L U to print prime numbers between $L$ and $U$.
You type: java primes 101
It types: 2 3 5 7 11 13 17 19 23 29
            31 37 41 43 47 53 59 61 67 71
            73 79 83 89 97 101

Definition: A prime number is an integer greater than 1 that has no divisors smaller than itself other than 1.

Useful Facts:
- If $k \leq \sqrt{N}$, then $N/k \geq \sqrt{N}$, for $N, k > 0$.
- $k$ divides $N$ iff $N/k$ divides $N$.

So: Try all potential divisors up to and including the square root.

Plan
class primes {
    /** Print all primes up to ARGS[0] (interpreted as an * integer), 10 to a line. */
    public static void main (String[] args) {
        printPrimes (Integer.parseInt (args[0]));
    }

    /** Print all primes up to and including LIMIT, 10 to * a line. */
    private static void printPrimes (int limit) {
        /* For every integer, x, between 2 and LIMIT, print it if * isPrime (x), 10 to a line. */
    }

    /** True iff X is prime */
    private static boolean isPrime (int x) {
        return /*( X is prime )*/;
    }
}

Testing for Primes

private static boolean isPrime (int x) {  
    if (x <= 1)  
        return false;  
    else  
        return ! isDivisible (x, 2); // "!" means "not"
}

/** True iff X is divisible by any positive number >=K and < X, * given K > 1. */
private static boolean isDivisible (int x, int k) {  
    if (k >= x) // a "guard"  
        return false;
    else if (x % k == 0) // "/%" means "remainder"  
        return true;
    else // if (k < x && x % k != 0)  
        return isDivisible (x, k+1);
}