

# CS61B Lecture #1

- Lab reader and Java reference manual available at Vick Copy, 1879 Euclid.
- Labs start immediately. *Get an account (if needed) and register electronically this week*
- Account forms available in labs.
- Class web page and newsgroup set up: read them regularly!
- Classes crowded: we're working on it. Go to any sections, labs where you fit.
- Appeals: boxes on third floor of Soda; Enrollment problems: Michael-David Sasson, 379 Soda; *NOT US!*
- Reading for today and Wednesday: *Programming Into Java* (reader) 1.1-1.9.

# 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.
- 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

```
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* defined inside *A*,"
  - E.g., `System.out.println`, `Greet.main`
- Ordinary function is *static method*, like `Greet.main`.
- Methods declare what kind 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.

# 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) {
        // do the work
    }
}
```

## Printing Primes (first attempt)

```
/** Print all primes up to and including LIMIT */
private static void printPrimes (int limit) {
    printPrimes (2, limit);
    System.out.println ();
}

/** Print all primes from L to U, inclusive, on one line */
private static void printPrimes (int L, int U) {
    if (L <= U) {
        if (isPrime (L))
            System.out.print (L + " ");
        printPrimes (L+1, U);
    }
}
```