CS 100

- Intro to programming ... mostly Java
  some Matlab

- knowing one language well → easy to pick up another one

  Java ... powerful, portable
  Matlab ... nice for simulations and graphing

- Java basics (common to most languages)
  - I/O ... messy, but points to underlying ethos
  - if/then, loops, arrays, etc.
  - 'methods' (aka functions, subroutines) → delegation
  - classes/objects ... organisation + modularity + reuse
  - GUIs ... interface design
  - threads ... juggling processes
  - recursion, efficiency, sorting
  - LOTS of programming!!!!

Java → uses **classes** and **objects** = hyperorganised!

A **class** Car is like a manufacturer who can only **construct** individual new cars.

A **class** Bucket will only **construct** individual new buckets.

Cars + buckets have natural things which belong to every car or bucket, although of course cars have different colours, etc...

To build a red car called Ferrari, we might write

```
Car Ferrari = new Car(red);
```

- I'm not promising that this will work!!!

and to build a yellow car called ccbb, we might write

```
Car ccbb = new Car(yellow);
```

Of course,

```
Bucket rollsroyce = new Bucket(huge);
```

will only give you a peculiarly named huge bucket.

If you want to access stuff in your car, then the dot **.** is the "genitive case", so

```
Ferrari.colour
```

would be red, and

```
ccbb.colour
```

would be yellow. This is also like a **path**; looking **into** the Ferrari or the ccbb to find the individual colours.

More on this later...

Enough of such generalities! How do we write a simple program in Java?

First we need to be able to get stuff into and out from the computer!

```
System.out.println("Once upon a time...");
```

looks into the **System** where it finds an **out**, and looks into **System**'s **out** where it finds a **method** (or **function** or **routine**) which can print a **String** of **characters** onto a fresh line on the standard output screen.

```
System.out.print("Golly gosh");
```

does exactly the same, except the **method print** doesn't finish with a "new line."

To read in a **String** of **characters** from the standard input keyboard:

```
InputStreamReader nab = new InputStreamReader(System.in);

BufferedReader grab = new BufferedReader(nab);
```

constructs a BufferedReader called **grab** so that

```
grab.readLine();
```

reads a whole line of input.

Java is a language of “let’s pretend!”, so **grab** is a **virtual** keyboard which has the ability (among other skills) of **readLine()**—all the other stuff is there to establish a connection between “make believe” and “reality.”

We can do the same thing with files:

```
FileReader secret = new FileReader(“spy.oops”);

BufferedReader james = new BufferedReader(secret);
```

constructs a BufferedReader called **james** so that

```
james.readLine();
```

reads a whole line of input from **spy.oops**. As a matter of common courtesy, you should

```
secret.close();
```

close the ‘file’ when you’ve finished with it!

(If you need to specify a path for your file, you can have

```
= new FileReader(“c:/money/penny/spy”);
```

or whatever is appropriate for your system.)

Actually, you should also nab.close() any InputStreamReaders you’ve opened when finished.

Similarly,

```
FileOutputStream plop = new FileOutputStream(‘meow.t”);

PrintWriter scribble = new PrintWriter(plop);
```

allows

```
scribble.println(“What big teeth you have!”);
```

to write the file **meow.t**, which again should be closed by

```
plop.close();
```

when finished with.

Of course, we could have done the same thing when writing to the screen:

```
PrintWriter tube = new PrintWriter(System.out,true);
tube.println(“How time flies!”);
```

Here, **tube** is the name of the make-believe computer screen.

Now that we can get stuff into and out of the computer, let’s actually write a program.