Java Basics

CSC207

A first program:

Here's a "hello world" in Python:

```
print('hello world')
```

Here's the equivalent in Java:

```
public class Hello {

    public static void main(String[] args) {

        System.out.println("Hello World!");

    }

}
```

Notes

Java is totally object-oriented

- Everything is defined inside classes (or interfaces)
- There's nothing like Python's "main block"
- There are no functions, only methods

Where does the program start?

- the program starts executing at the "main" method of the class that you are running
    - In the **`Hello World`** program, this was the **`Hello`** class.
- Always use:

```
public static void main (String[] args)
```

- More on that shortly…

Formatting

- White space is irrelevant! Use {} to group things instead
- There is a way to automatically format the make the code easier to read.
- Every single-line statement ends with a semi-colon;
- type String in Java needs double quotes like “This”
- [] indicates an array (like a Python list, but much more limited). For example: `int[3]` is an array of integers containing 4 integers.

Printing to screen

- `System.out.println` is Java's version of Python's `print`
- Want details? Look it up in the online "Java API" documentation:
    - http://docs.oracle.com/javase/8/docs/api/
- `System`: a class built in to Java
- `out`: a data member (variable) of class `System`
    - it's type is PrintStream
- PrintStream has lots of methods for printing to screen, file, etc.
    - `print` and `println` are crucial methods

Packages

- packages: bundle up groups of related classes and store within the same directory
- `java.lang` is the only package that's automatically imported. It contains, for example:
    - `Math` class: constants like Pi, methods like abs, cosh, etc.
- `java.util`: crucial classes for compound objects like
    - ArrayList (like Python list)
    - HashMap (like Python dictionary)
- `java.io`: for input/output to files etc.
- notice all the "`javax`" packages:
    - They are extensions to the standard Java language
    - We will see the `Swing` package.

Java Architecture

How the computer runs the code:

- When you run a program in any language, must be translated into the language of the particular chip it will run on.
- Three traditional methods include:
    - compile
    - interpret
    - some combination of both

Compile:

- translate the whole thing before execution
- compiler can perform lots of fancy optimizations because has the big picture
- must create a different executable for every machine
- harder to provide useful feedback during debugging because executing machine code at this point, and programmer wants to think in terms of the source code

Interpret:

- translate and execute, one statement at a time
- must do the translation every time you execute
- also, interpreter can't optimize much so code runs more slowly

A Hybrid Approach: "pseudo-compile" the code

- Before execution, translate into an intermediate form.
    - in Java, it's called bytecode
- At execution time, interpret that intermediate code line by line .
    - in Java, it's called the JVM
- Then we can execute it on any machine that has an interpreter for the intermediate code.

Java code is considered "portable" because:

- Bytecode can run anywhere (but then so could C code, but bytecode is "more compiled", leaving less work for the JVM than a C compile-and-run).
- The meaning of statements in the language is more fully defined.
    - E.g., an int is always 32 bits, vs in C where it is different things on different platforms.
- Java has so much built into the language and its standard libraries that substantial code can be written without any add-ons. Not true of C, for instance!

Running a Java program

Command Line:

1. `javac Something.java`
    - "compiles" into bytecode
    - checks syntax
    - converts to a lower-level language the computer understands
    - produces file Something.class
2. `java Something` -- don't say `Something.class`
    - executes the bytecode

Naming Conventions

- Named file "`Whatever.java`" if it defines class `Whatever`
- If using packages (later), this imposes a directory structure
- Classes must be in the right directory to compile and run.
    - Your IDE or Integrated Development Environment (Eclipse) will do all this for you.
- Eclipse does compile and run in one run button (plus has lots more!)

Further notes about the "hello world" program

Access

- There are different levels of access for classes, variables, and methods.
- We use "access modifiers" to specify a level of access:
    - `public`: accessible by any class
    - `private`: only accessible from within the class it is declared in
    - `protected`: in between (more details later)

Static

- `static` means "there's only one"
    - In other words, the class owns it, instead of every instance having one.
- Can have static variables and static methods
- `main` is a static method (aka class method)
    - It belongs to the class, not to a particular instance

Comments

- // a one line comment
- /* possibly

  *several lines

  *of comment */
- /** a special

  * documentation comment;

  * can be processed by

  * javadoc */

Style Conventions

- We use camelCase, not pothole_case (except for constants)
- class: Its name should be a noun phrase that starts with a capital
- method: Its name should be a verb phrases that starts with lower case Eg, getV, setV, isV, toV
- instance variable: Its name should be a noun phrase that starts with lower case
- local variable or parameter: same as instance variable, but acronyms and abbreviations are more okay
- constant: all uppercase, pothole
- good to follow patterns observed in the Java API, e.g. in, out