Java and Command Line Compiling

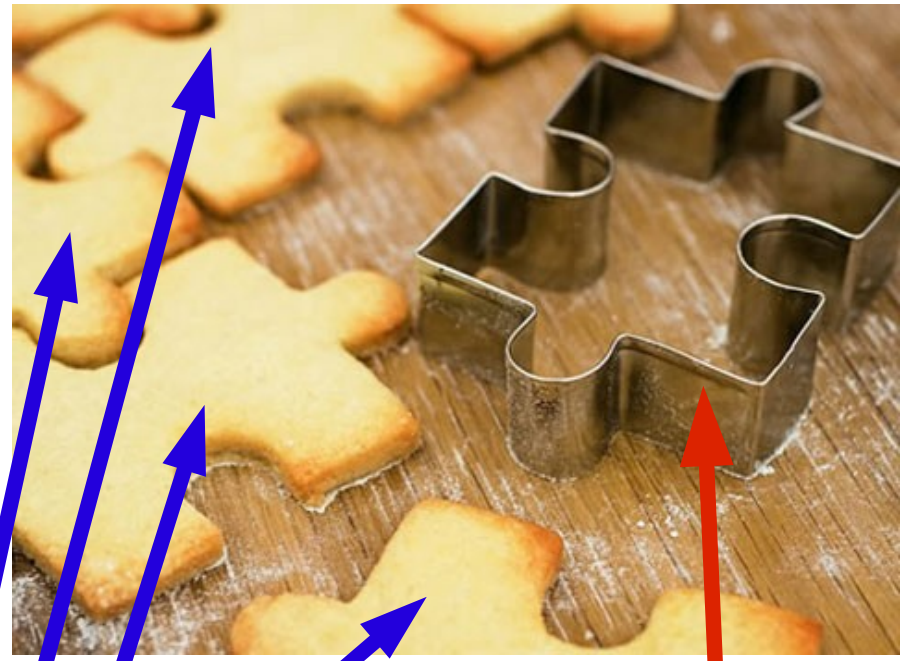
Nasser Giacaman
SoftEng 206: Software Engineering Design

Very short Java intro

- We won't be diving deeply into Java here, mainly just a single lecture for the sake of your first assignment ;-)
- Why Java?
  - "Write once, run anywhere"
  - Real object-oriented language
  - Free under the GPL
  - See TIOBE Programming Index
  - Android
  - Most importantly... cos I said so (well, at least for assignment 1)

Main Java concepts

- Class
- Object (or instance)
- Method
  - Class method (`static`)
  - Instance method
- Variable
  - Class variable (`static`)
  - Instance variable



Objects or instances

Class
(the definition)

Hello World

```
public class HelloWorld {
      public static void main(String[] args) {
            System.out.println("Hello 206");
      }
}
```

- No cookies were made during the execution of this program
- To a certain extent, you can ignore all the benefits that Java brings (i.e. OO concepts) and write everything inside the main method
  - Since your 1st assignment is smallish, you can do that. Just make sure you don't do this anywhere else (e.g. your next assignments, projects, etc in other courses or endeavors in life)

Classes versus instances

```
public class Animal {
        private String name;
        private int numLegs;
        public Person(String n, int l) {
                name = n;
                numLegs = l;
        }
        public void talk() {
                String says = “Hello from ”+name;
                System.out.println(says);
        }
}
```

```
Animal garfield = new Animal(“Garfield”,4);
Animal skippy = new Animal(“Skippy”,2);
skippy.talk();
```

Remote access: SSH vs SFTP

- `> ssh ngia003@shell.ece.auckland.ac.nz`
  - This allows you to connect to the shell of the specified machine as the specified user. You only have access to the files on the remote machine, and cannot upload/download to the local machine
- `> sftp ngia003@shell.ece.auckland.ac.nz`
  - SSH File Transfer Protocol
  - `> pwd` `> lpwd`
  - `> cd` `> lcd`
  - `> put` `> get`
  - `> help`
- A combination of `ssh` and `sftp` will allow you to move files between the remote and local machine, and also edit then on the remote machine... for example your homepage, running experiments, etc

Compiling C

- Now that we know a few things about Linux, let's have a closer look at compiling C programs
- `> ls`

  `example.c`
- `> gcc example.c`
- `> ls -l`

  `-rwxr-xr-x  a.out`

  `-rw-r--r--  example.c`
- `./a.out`

C program with arguments

- `> ./a.out 3 5`

  `The sum is 8`
- `> ./a.out 1`

  `Usage: you must provide 2 numbers`

- `int argc, char *argv[]`
- `int num1 = atoi(argv[1])`

Let's make this program “real”

- `> gcc -o calc example.c`
- `> ls -l`

```
-rwxr-xr-x  calc
-rw-r--r--  example.c
```

- `./calc 3 5` (rather than meaningless a.out)

- `> mv calc ~/bin`
- `> calc 3 5`
  - From anywhere on your system, since its on your `PATH`

Java

- So, is Java an interpreter or compiler?? → BOTH!

Java source
Code (*.java)

Java Compiler
`javac`

Java bytecode
(*.class)

JVM
`java`

- Interpreting, or
- JIT compilation, or
- AOT compilation

JVM
`java`

Windows machine

Linux machine

Compiling (and trying to run) Java programs

- `> ls`

  `MyProgram.java`
- `> javac MyProgram.java`
- `> ls -l`

  `-rw-r--r--  MyProgram.class`

  `-rw-r--r--  MyProgram.java`
- `./MyProgram.class`

  `./MyProgram.class: Permission denied` (permission is 644)

- `chmod +x MyProgram.class ; ls -l`

  `-rwxr-xr-x  MyProgram.class`
- `./MyProgram.class`

  `./MyProgram.class: Permission denied`

Compiling and running Java programs

- `> javac MyProgram.java` (Java compiler, part of SDK)
- `> java MyProgram` (Java virtual machine)

- PATH versus CLASSPATH
  - Both are a list of paths (relative/absolute)
- PATH tells Linux where to find executables (e.g. `ls`, `java`, `javac`, `pwd`, etc)
- CLASSPATH tells `javac` and `java` where to find other *.class files (some of which might be contained within .jar files)
  - If this environment variable is not set, then you must provide the classpath as an argument

- `> java -cp AnotherLib.jar:. MyProgram`

Java classpath

- ```
  import hello.Cool;
  class MyProgram {
        Cool c = new Cool();
  }
  ```

- `> jar -cf hello.jar hello/`
  - `> rm -fr hello/`

- `> javac -cp hello.jar MyProgram.java`
- `> java -cp hello.jar:. MyProgram`

- Or `> export CLASSPATH=hello.jar:.` (now no need for `-cp`)
  ```

Java packages

- ```
  package se206;
  class MyProgram {
  }
  ```

- `> mkdir se206`
  - `> mv MyProgram.java se206`
- `> javac se206/MyProgram.java` (se206 is “just a folder”)
- `> java se206.MyProgram` (se206 is part of the class name)
  ```

Running your “real” Java program

- So, you've got a Java application, it uses all these external libraries, inside a package, etc etc.. It would be annoying if you (and users of your program) always have to set the classpath, be in the correct directory, etc, etc...
- Some possible ways to simplify it:
  - Place everything in a single .jar file, specify the jar's manifest file
    - `> java –jar hello.jar`
    - Advantage: runs on Linux/Mac/Windows
    - Disadvantage: Not as “pretty” as having a real executable
  - Create an executable script file, put it somewhere in your PATH
    - `> calc`
    - Disadvantage: BASH for Linux, .bat for Windows, …

Making an executable jar

- Create a manifest file (e.g. mani.txt), just a text file with the following line:
  - `Main-Class: se206.MyProgram`
- And now create the jar file:
  - `> jar -cfm hello.jar mani.txt hello/ se206/`
- Notice how `f` and `m` must be specified in the correct order (hello.jar and mani.txt respectively)

- To take it further, create a text file, for example `calc` inside ~/bin, and use the absolute path to where the jar file is, e.g:
  - `java -jar ~/bin/hello.jar` (inside calc)
  - `> chmod +x ~/bin/calc`
  - `> calc` (our first BASH script file!)