# Workshop Week 2

## Table of Contents

## Discussion

### Transitioning to Java

1. What is OOP? How is it different to procedural programming?

- OOP uses classes as fundamental units of abstraction
  - data belongs to classes, manipulated by methods
  - different way to structure programs
- procedural uses functions as fundamental units of abstraction
- also functional and logic programming
- Object oriented programming use abstract data types, which contain data and functions that can act upon this data.
- procedural programming consists of sequences of function calls acting on data
- built-in abstraction helps you build hierarchies, building on work already completed and ensure data is handled in the way that it is meant to be

2. What are the primitive data types (basic arithmetic types) available in C?
   - primitives: basic building blocks; other data types are derivative
   - various flavours of **`int`**, **`long`**, **`short`** (signed/unsigned, different lengths)
   - **`float`**, **`double`**, **`char`**
3. What are the primitive data types available in Java?
   - **`boolean`**, **`char`**, **`float`**, **`double`**, **`int`**, **`long`**, **`short`**
   - **`char`** in Java is unicode ⇒ 2 bytes
4. What is the entry point to a Java program? What parameter does it take?
   - main method: `String[] args` array of arguments

5. In 1995, Sun Microsystems created a slogan “Write once, run everywhere” to entice the adoption of Java. What does this slogan mean?

    - Java is portable, as it is first compiled to Java bytecode, and then interpreted to machine code on a specific machine. This means that each unique hardware will need to have its own interpreter built, however this is much less involved than creating a full compiler
    - Java VM written for each architecture and then you can run the same bytecode on any architecture
    - java code -> | `javac` | -> bytecode -> JVM -> program
    - JVM does just-in-time compilation, which converts bytecode to native machine code, then runs it
    - Python uses has to interpret directly [TODO: research how this works]
    - C# works in a similar way with .net virtual machine. Was developed as competitor to Java
    -

- C function pointers in structs: basically methods
- led to C++

**Software Projects**

1. Experiences when developing software in past projects

    - spaghetti
    - replicated efforts because system was not broken down well into reusable code
    - poor documentation: intrinsic + extrinsic resulting in significant efforts to understand codebase when maintenance was required
    - failed to implement existing solutions through lack of knowledge (e.g. databases)
    - difficulty maintaining code and dependencies

2. Thoughts on software design and software development

    - why is design important?
        - as software gets larger there are more “moving parts” that need to interact
        - high-level design allows you to divide this into tasks that can be tackled by teams simultaneously, with well managed interfaces
        - this makes code easier to test, more extensible, more maintainable
    - what is difficult about software design?
        - understanding requirements, and how requirements will change over time
        - implementing an architecture that is maintainable and extensible

  - discipline to plan rather than tackle the problem directly, and following your procedures as you progress (e.g. keeping documentation up to date)
  - project management: team, deadlines, …
- what can you do differently right now to help practice designing software?
  - learn UML and markdown
  - improve documentation
  - record design process in documentation!
  - look for existing solutions