Programming Design and Implementation

Lecture 1: Introduction

Updated: 21st February, 2020

Mark Upston
Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

Outline

What is a Computer?

- Machine that accepts data, processes it and produces output
- Computer consists of:
  - CPU
  - Input/output devices
  - Dynamic memory
  - Secondary storage
- So it can run software:
  - Systems software:
    - Operating systems, network tools, software development tools (e.g. compilers)
  - Application software:
    - Word processors, spreadsheets, databases, modelling and simulation.

Software

- A program is a set of instructions to a computer.
- These instructions are written in a programming language.
- Each computer understands one language, its machine language, which is not (usually) human readable.
- A program is written in a high level language (for humans) and then translated into machine code (for the computer).
- The purpose of you studying this unit is to design and write well structured, robust, maintainable software - not just to write a program that works some of the time.

Programming Languages

▶ 1940's	Machine language
▶ Early 50's	Assembly language
▶ Late 50's	High level languages & compilers introduced
▶ Fortran	
▶ COBOL	
▶ 70's and 80's	Emergence of better structured languages
▶ Pascal	
▶ Ada	
▶ C	
▶ 90's	Object Oriented Languages
▶ C++ *(or was it?)*	
▶ Java	
▶ Perl	
▶ 2000's	
▶ C#	
▶ Python	
▶ Ruby	

To be OO or not to be OO?

- There are 2 basic paradigms for designing imperative algorithms:
  - Procedural:
    - Focus on the steps required to perform the task.
    - The design of the steps lead to the types of data structures that will be required.
  - Object Oriented:
    - Focus is on the entities required. (i.e., What do we need to represent in the algorithm?)
    - What functionality each thing will require.
    - How these things will communicate with each other.
    - Each entity will be represented as an object.
    - The design of each object leads to the steps required.

Data Representation in Computer Memory

- Computer memory is made up of components that can be in one of two states (on or off)
- Other binary methods of information include:
  - On and Off (Flashing light)
  - Dot and Dash (Morse)
  - North and South (Magnet)
- Can be used in two ways:
  - Represented actual values in base 2
  - Hold an arbitrary series of states coded with particular meanings

Terminology

- Each **b**inary dig**it** is called a **bit**
- A group of eight (8) bits is a **byte**
- Memory is broken up into storage locations of a particular **wordsize**
- Wordsize is machine dependent and will be one or more bytes long
  - Now 64 bits (8 bytes)
- Each memory location is located through its **memory address**
- All data and programs are stored in memory using various interpretations of these groups of 1's and 0's

Data Types

- Manner of interpretation of the 1's and 0's varies for different data types stored:
- The way in which the 1's and 0's are interpreted depends on the data type being represented.
  - Addresses
  - Instructions
  - Integer values
  - Real values
  - Characters
    - Single characters
    - Character Strings
  - Boolean

Software Development

- Problem Definition

  Think/consult/revise

- High Level design (typically UML)

  Specify

- Algorithm Design

  Test

  Convert

- High level programming language implementation

  Translation

- Conversion to machine instructions

  Test

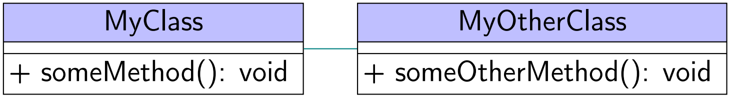
- Execute and test that it actually does the required job

Problem Definition

- Must define and understand the problem requiring a solution
- Start with a "top Level" general English description of the problem
- State what data is required to solve the problem
  - The input data
- State what results will be calculated
  - The output data
- What actions need to be taken to generate the results
- Crucial part of solution is to know what the problem is, but this is often ignored by poor software developers
- Need to consider security, reliability and performance requirements

High Level design (UML)

- UML is a way of showing software design.
- It is not covered in depth in this unit, but it is useful to know how it is structured and percieved.
- It outlines communication and structure between different parts of the program
- This wont make sense yet, but if you see a diagram like this, it may help with understanding the structure of a system.



Algorithm

- An algorithm is a set of detailed, unambiguous, ordered steps specifying a solution to a problem
  - Steps must be stated precisely, without ambiguity
  - Enter at the start & exit at the bottom
  - English description independent of any programming language
  - Non trivial problem will need several stages of refinement
  - Various methodologies available
  - Must be desk-checked for correctness

Algorithm - Pseudo Code

- Algorithms are expressed in Pseudo Code:
  - English like phrases which describe the algorithm steps
  - The pseudo code is evolved from a rough description to something which almost looks like a programming language
  - Pseudo code development is about refinement
    - Developing an algorithm is a journey where the problem
  - Algorithm design is an art that takes a lot of practice

Pseudo Code - Simple Example

- Problem
  - Write a program to calculate the sum of 2 numbers input from the keyboard (user) and output to the screen (user)
- Algorithm:

```
MAIN:
  INPUT numOne
  INPUT numTwo
  sum = numOne + numTwo
  OUTPUT sum
```

Conversion

- Convert algorithm description into implementation of HLL **H**igher **L**evel **L**anguage. e.g., Java.
  - Known as coding
  - Programmer needs to know the semantics and the syntax of the language
  - The files of HLL statements are called the **source files**

Translation

- Process by which the "Language" translates source code into machine code
- Errors found are called *syntax* or *compile-time* errors
- Two variants
  - Languages that are compiled
    - Translation is done before execution
    - C, C++
  - Languages that are interpreted
    - Translation is done during execution
    - Ruby, Python
  - Discussed later

Java Implementation

```
/*********************************************
 * Author: Mark Upston                        *
 * Date: 25/02/2020                           *
 * Purpose: Calculate the sum of 2 Integers   *
 *********************************************/
import java.util.*;
public class MyFirstJavaApplication
{
    public static void main(String [] args)
    {
        int numOne, numTwo, sum;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter 1st number");
        numOne = sc.nextInt();
        System.out.println("Enter 2nd number");
        numTwo = sc.nextInt();
        sum = numOne + numTwo;
        System.out.println("Sum is " + sum);
    }
}
```

Program Execution

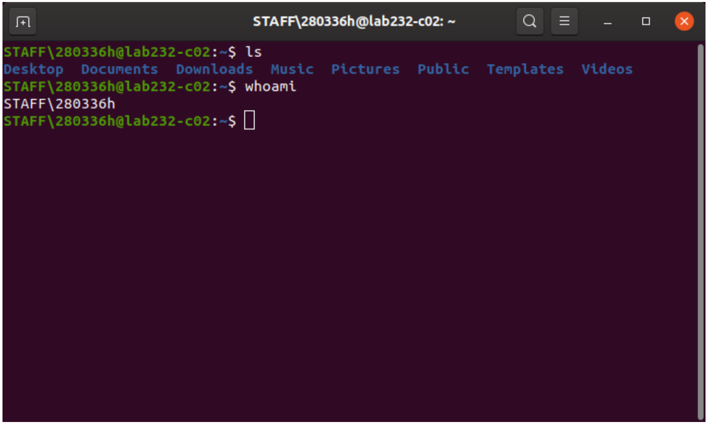
- If no compile errors, the machine language file can be executed
- Errors during program's execution are referred to as `"Runtime"` or `"Execution"` errors
- Errors in a computer program are called bugs and the process of eliminating them as debugging.
- They are not really bugs they are **mistakes**.
  - These mistakes are found and eliminated by thorough testing
- When a program is ready, it is "put into production" and will be amended and expanded over time.

Introduction to UNIX

- The operating system we will be using is called Linux
  - Linux is a variant of UNIX
- UNIX is:
  - Totally different to what you might be used to
  - An operating system which provides far greater scope in what can be accomplished but at a cost of a more difficult to learn user interface
- As future computing professionals it is unacceptable for you not be familiar with a UNIX environment

Command Line Interfaces

- A command line interface is composed of:
  - A prompt which signals the user when they can type commands
  - A command line on which the typed commands appear



```
STAFF\280336h@lab232-c02:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
STAFF\280336h@lab232-c02:~$ whoami
STAFF\280336h
STAFF\280336h@lab232-c02:~$ 
```

CLI vs GUI

- **G**raphical **U**ser **I**nterfaces are:
  - Easy to learn how to use poorly
  - Slow
  - Inefficient when dealing with repeated sets of the same functionallity
- **C**ommand **L**ine **I**nterfaces are:
  - Require the user to understand a number of concepts
  - Require the user to be familiar with a command language
  - Very fast
  - Highly efficient, when dealing with repeated sets of the same functionallity
    - e.g., Multiple files that have a similar name or extension
- In PDI we will **only** be dealing with the Command Line Interface

UNIX Shells

- Under UNIX the Command Line Interface is known as a Shell
- Shells allow us to perform commands on the Operating System as a user
  - Think of it as the communication medium between you and the hardware
- The Shell we will be dealing with in PDI will be **bash**
  - **B**ourne **A**gain **SH**ell
  - This will be covered in depth in Unix and C Programming (`COMP1000`)

Paths and Directory Structure

- Within UNIX everything is a file, and hence can be accessed with a direct path
- Generally speaking we will only be accessing our files in our Home (∼) directory
  - This means we may use the shortcut 'tilde' to access home from anywhere

```
mark@314lab:∼$ cd ∼/Documents/PDI
```

- However we may also give a direct path, specified from the root directory (/)

```
mark@314lab:∼$ cd /home/mark/Documents/PDI
```

- Any time you specify a path you **must** make sure it is accurate, otherwise the OS does not know where you are talking about

Common Bash (UNIX) Commands

- **cd**: Change Directory
  - Moves to a different directory, specified after the command

```
mark@314lab:~$ cd Documents
```

- **cp**: Copy
  - Copies a file or folder from source to destination

```
mark@314lab:~$ cp Source.txt Destination.txt
```

- **mv**: Move
  - Moves a file or folder from source to destination
    - Note: The original file will not remain

```
mark@314lab:~$ mv Source.txt Destination.txt
```

Common Bash (UNIX) Commands (2)

- **`ls`**: List
  - Lists all (non hidden) files and folders in the current directory
    - Note: You may specify "**`-la`**" to display hidden files aswell
    - *This is L (el)* ***not*** *1 (one)*

```
mark@314lab:~$ ls
```

- **`mkdir`**: Make Directory
  - Makes a new folder, specified after the command

```
mark@314lab:~$ mkdir PDI_Worksheets
```

- **`rm`**: Remove
  - Removes a file, specified after the command
    - Note: To remove folders, you need to specify "**`-rf`**"
    - Be careful! This will remove with force and can not recover it

```
mark@314lab:~$ rm MyFile.txt
```

Introduction to Java

- Java is an Object Oriented (OO) language with its roots in C & C++
  - C is an imperative language, strongly associated with UNIX operating system, designed in the early 70s
  - C's flexibility, which lead to its popularity, also made it dangerous
  - C++ is basically C with OO features added which makes it a "large" language
- In 1990-91 James Gosling designed Java as a "small" OO language
  - Initially aimed at information appliances, but in 1993 adapted for animation & interaction on WWW
  - Introduction of 1995 Netscape allowing Java applets led to its current popularity

Compiling

- Compilation:
  - Process whereby the file of source code is translated into machine code
  - The source code is checked to ensure it conforms with grammar (syntax and semantics) rules
  - If no syntax errors found at compile time then machine code file is created and can be executed
  - If even just one error then machine code file does not exist and there is no machine code to run

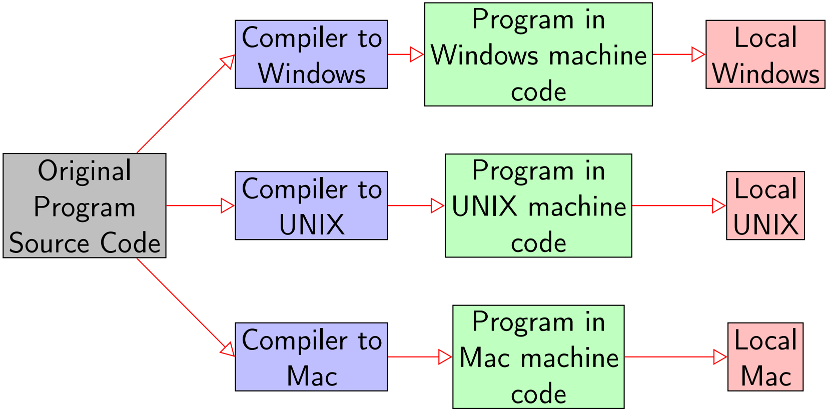
Interpreting

- Process whereby the file of source code is translated a line at a time into machine code instructions that can be executed by the machine.
  - If syntax errors exist then the program will be partially executed
  - As soon as syntax errors are encountered, execution halts
- Python and Ruby are both interpreted HLL
- Interpreting code is slower than executing compiled code, because syntax checking & translation has already been done with compiled code
- If syntax errors can be eliminated before the source code is interpreted then the syntax error problem is avoided

Java Platform Independance

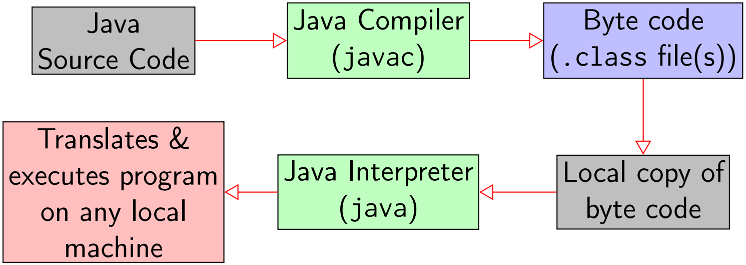
- Platform independence is achieved by running byte code on a **J**ava **V**irtual **M**achine (JVM)
  - Byte code is machine code for the JVM. A Java compiler compiles from source code to byte code
- The JVM is itself a program whose job it is to interpret the Java byte code
- Each machine actually needs code in its own particular native machine language: thus byte code needs to be interpreted to the local machine code to be able to execute locally
- The overheads of interpreting byte code are lower than traditional interpreters because the conversion is from machine code (JVM) to machine code (native machine code)
  - Syntax checking has already been done at compile time

Traditional Methods: Compile -> Execute


Original Program Source Code
Compiler to Windows
Program in Windows machine code
Local Windows
Compiler to UNIX
Program in UNIX machine code
Local UNIX
Compiler to Mac
Program in Mac machine code
Local Mac

Java Virtual Machine

- As long as local machine has the byte code interpreter it can download any Java program in byte code & execute it and yet:
  - Source code is secure
  - Only one version of the byte code needs to exist



How to create a Java program

- Design and write the algorithm first!
- A text file with a file extension of `.java` must be created
  - In this file, you store the human readable Java program
- The Java compiler (known as `javac`) is then used to compile the source code into byte code
  - For each class (see later) defined in source code the compiler will create a corresponding file of byte code
  - The name of each byte code file will be the name of the class (as defined in the source code) with an extension of `.class`.
- The Java interpreter (known as `java`) is then used to translate the byte code and execute the resulting native machine code

Applications and Applets

- A Java application is a program which is executed by directly invoking the Java interpreter
  - Must contain a `main()` method
- Java servlets and applets are Java programs which are executed from within a web browser
- PDI will deal with only Java Applications
- Graphical user interface issues will be dealt with in great detail in Computer Graphics (`COMP2005`) and Human Computer Interfaces (`ICTE3002`)

Pseudocode

```
MAIN:
  message = ”Welcome to Java”
  OUTPUT message
```

A Simple Java Application

```
import java.util.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        String message;

        message = new String("Welcome to Java!");
        System.out.println(message);
    }
}
```

Creating, Compiling and Running

- The Java code on the previous slide is entered into a text file using a text editor (under Linux we <u>**must**</u> use `vim`).
- The name of the `.java` file <u>**must**</u> be exactly the same as the name of the class (i.e., `MyFirstProgram.java`)
- The `.java` file is then compiled into byte code
- The command would be:

```
[user@pc]$ javac MyFirstProgram.java
```

  - If the program contained errors then they will be displayed and the byte code is <u>**not**</u> generated
  - Otherwise the byre code is produced
  - The byte code is stored in a file called `MyFirstProgram.class`
- The program can then be executed using the command:

```
[user@pc]$ java MyFirstProgram
```

The Import Statement

- Java comes with an extensive library of classes which can make the task of implementing algorithms much easier
- It is very common for organisations to develop their own class libraries
- In PDI we do not have any class libraries, however you do need to understand the import statement
- The import statement tells the Java compiler that libraries are to be found by looking for the directory path specified

```
import java.util.*;
```

  - Means all of the class library files in the `util` directory

The CLASSPATH Environment Variable

- Contains all the directories that should be searched when compiling.
- The Java class libraries are stored in a default area when Java is installed. Under Linux it will be somewhere like:

```
/usr/local/java/jdk1.6.1/lib
```

- The CLASSPATH environment variable does not have to be set if the only place your Java program needs to look is the standard one
- If you have libraries, you must specify where they are
- Previously the CLASSPATH variable was set as below:

```
$CLASSPATH:/usr/java/latest/lib:/usr/units/st151/classes:.
```

  - **Note:** that ST151 is the old name for PDI and this CLASSPATH setup is no longer required in this unit

Setting the CLASSPATH Variable UNIX

- Each path is delimited by a colon (:). It tells java where to look for files outside of the local scope
- Under UNIX to set the CLASSPATH variable we use the export command:

```
export CLASSPATH="$CLASSPATH:<Java_Path>:/usr/units/st151/classes:."
```

  - **Note:** Descriptors denoted in "<Something>" mean implement yourself
  - If we do this interactively we will have to type this command every time we login
  - A better way to do this is to add a special file to our home directory called the .bashrc

Setting the CLASSPATH Variable Windows

- Under Microsoft O/S (Windows) to set the CLASSPATH variable we use the GUI:

```
set CLASSPATH=C:\jdk1.6\lib;C:\myjava\st151;
```

  - Where myjava is the folder where you have placed other non standard libraries
  - By going to the control panel on Windows you can set the CLASSPATH variable so that it will always be in effect. Please see the appropriate howto on Blackboard for instructions on how to do that

Java Classes and Objects

- A Java application consists of a series of classes
  - Everything in Java is encapsulated within a class
- Within a class will be a number of:
  - Methods
    - A method is a set of Java instructions (more later)
    - Also referred to as a "Submodule" in this unit
  - Variables
    - A variable is a piece of memory used to store data (more later)
- An Object is an instantiation of a class:
  - A variable is declared to be of a particular class
  - It is initialised by allocating memory and creating an instance of the class in memory
  - The memory location of the object is then placed in the object variable

The main() Method

- A Java program starts by executing the main() method
  - The main() method provides the starting point for the program
  - Within the main() method:
    - Other methods will be invoked
    - Objects will be created
    - Methods within those objects invoked

```
  public static       void      main   (String [] args) { .... }
|-Type of method-| |-Return-| |-Name-| |- Parameters -| |-Code-|
```

- For the moment do not worry about the "Parameters" in main(). This will be covered in DSA

Summary

- We have covered a lot today
  - It is extremely important to consolidate this lecture
  - Watch the iLecture a couple of times
- Attend the practical sessions
  - Start in week 1
  - Compulsory (Sign-offs)
  - Linux labs are in building 314, rooms 218, 219, 220, 221 & 232
  - Use labs outside of your prac time if there is no class or there are free computers (check with the tutor first)
  - Building and labs are 24/7 access
- Install Java so you can work at home
  - You can also remote login to machines using SSH (**S**ecure **Sh**ell) via VPN

Next Week

- The next Lecture will address the following:
  - Variables
  - Data Types
  - Arrays