Course Objectives

- Familiarise with both the Imperative and the Object-Oriented aspects of Java
- Build structured, reusable, manageable platform independent code.
Course Structure

- Language basics
- Core Object Orientated Principles
- Advanced topics
Introduction

- 3hrs of lectures per week
  - 1hr tutorial/practice every week
- **IMP:** Practice MUST be taken seriously
- Course slides and resources will be made available at
  http://staff.um.edu.mt/cabe2/lectures/csa1019.html
- 1 assignment will be given (25% of final mark)
- Slides are detailed but you should make your own notes
  - worked examples
  - techniques
  - design
Requirements

- **Recommended course textbooks**

- **Java Compiler and Documentation**
  - Available at http://java.sun.com as *Java 2 SDK 1.6 (J2SE)*
  - Documentation from: http://java.sun.com/javase/6/docs/

- **IDE**
  - *BlueJ*: easy-to-use
  - *Eclipse*: robust, used in industry/research
  - *NetBeans*: recommended by Sun
**Java Language Specification**

- A general-purpose, concurrent, class-based, object-oriented language.
- Designed to be simple enough that many programmers can achieve fluency in the language.
- The language is related to C and C++ but is organized rather differently, with a number of aspects of C and C++ omitted and a few ideas from other languages included.
- Language specification is the technical definition of the language, i.e.  
  - syntax and semantics  
  - read more: http://java.sun.com/docs/books/jls/
Java API & JDK

- **API (Application Program Interface)** contains predefined classes and interfaces
  - three editions:
    - Java 2 Standard Edition (J2SE)
    - Java 2 Enterprise Edition (J2EE)
    - Java 2 Micro Edition (J2ME)

- **JDK (Java Development Kit)**
  - we will use J2SE 6.0 (formerly known as JDK 1.6)
  - set of programs for developing and testing Java programs.
Java compilers translate language code into an intermediary bytecode (specifically Java bytecode), using the `javac` command.

- bytecode consists of machine instructions specific to an abstract or virtual machine.
- code is then run on JVM (Java Virtual Machine), using the `java` command, that interprets and executes the generic Java bytecode.
Java Compiled & Interpreted

- Java uses a combination of compiling and interpreting, and this presents a fundamental advantage:
  - allows programs to be run on any machine, provided that the JVM is present on that machine, “Write once, run anywhere”.

```java
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

HelloWorldApp.java
Java Virtual Machine

- The JVM is considered to be “virtual” because it is an “abstract computer” that runs compiled java programs.
- It is generally implemented in software on top of a "real" hardware platform and operating system.
- All Java programs are compiled for the JVM. Therefore, the JVM must be implemented on a particular platform before compiled Java programs will run on that platform.

<table>
<thead>
<tr>
<th>Compiled Java Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>Hardware Platform and Operating System</td>
</tr>
</tbody>
</table>
JRE: Runtime Environment

- subset of JDK, can be downloaded separately
- contains modules required to run applications anywhere
- no development utilities
Java File Structure

- **.java** – source file created by the developer
- **.class** – bytecode file generated by the compiler
- **.jar** – compressed package file
  - easily portable
  - executable through `java -jar` command

- There is no tool in JDK that allows for the creation of `.exe` files, however it's possible to use:
  - **Webstart**: allows users to launch and manage the application from their browser, e-mail or desktop
  - third party software
    - http://www.bysoft.se/sureshot/exej/
Good Programmers

- Programming is synonymous to a strategy game
  - the best players know and understand the domain
  - they experiment with different tactics
  - they practice the game

- How to become a good programmer?
  - learn the language rules well
  - before indulging into coding understand the problem domain well
  - experiment by writing different code that does the same/similar things
  - learn effective testing and debugging
  - document your code
  - practice…more practice…even more practice…
Essentially

- What do you need to start programming in Java
  - JDK (J2SE edition) version 6.0
  - J2SE documentation
  - Java IDE
  - Recommended text book
Simple Java Program

```java
//outputs Welcome to Java
public class Welcome{
    public static void main(String[] args){
        System.out.println("Welcome to Java");
    }
}
```

- Save the file as `Welcome.java`
- Compile and execute
Sample Output

- open `cmd` window
- `change directory (cd)` to source file directory
- compile using `javac` command
- execute using `java` command

Note: you need to make sure that the `bin` directory of the JDK is included in the system’s classpath.
Exercises

- Compile and execute your own *Welcome.java*
- Create a new program that prints out your name
- Create a new program that prints out multiple strings