# Lecture 1 — Introduction
## CITS2005 Object Oriented Programming

Department of Computer Science and Software Engineering
University of Western Australia

# Basic Information

- Unit coordinators: **Dr Max Ward** and **Dr Andrew Gozzard**
- Dr Andrew Gozzard is your main point of contact, I am second
- Let's start with the unit outline for details about course content, lectures, labs, and assessments

## Where to go

- LMS contains all required links and resources
- See `https://lms.uwa.edu.au/`
- Content will be released roughly a week in advance
- The textbook is available via LMS
- Announcements will be made via LMS. Please check regularly
- Unit email: `cits2005-csse@uwa.edu.au`
- Please use `help2005` for help first, and the unit email second
- Never use our personal emails!

# Labs

- Labs start week 2
- Due to a timetabling error, we were not allocated enough labs
- Please feel free to come to any lab and to multiple labs
- https://timetable.applications.uwa.edu.au/
- Due to yet another timetabling error, Wednesday labs are in 2.05, not 2.03!

# Labour Day

- Monday March 4th (next Monday) is a public holiday
- Thursday lecture is unaffected
- Replacement lecture:
  - 11AM
  - Friday 8th of March
  - GGGL: [ G21] Webb Lecture Theatre

## Unit Philosophy

- A philosophy of this unit is that learning a programming language is like learning a nautral language. You must immerse yourself to become fluent. We will be seeing a lot of code examples in lectures
- We will also be running lots of code live in the lectures
- Please take this to heart when it comes to the labs and revision. The only way to write good code is to write lots of bad code first

# Contents

- See Chapter 1 of the textbook
- History of Java
- How to run Java
- Understanding "hello world" in Java

# Java

- We will be learning Object Oriented Programming (OOP) with the *Java* programming language
- Java 1.0 was released in 1996 by Sun Microsystems
- Java is known for using a Virtual Machine (VM) allowing "write once, run anywhere"
- The use of a VM gives Java security and portability advantages
- Became popular through the web (e.g., Java applets)
- Now widely used everywhere
  - https://pypl.github.io/PYPL.html
  - https://www.tiobe.com/tiobe-index/

# Java

- These days Java is used in many places
- Web servers
- Android
- Desktop applications
- Games (e.g., Minecraft)

# Java

- Java is often called a C-style language
- It borrows syntax from C and C++ (e.g., curly braces { })
- However, C and C++ are compiled languages
- Java blurs the line by compiling to bytecode that runs on a VM
- Microsoft's C# language is heavily influenced by Java and is intended to compete with it
- JavaScript actually has nothing to do with Java! Pure marketing

# Java

- The Java VM originally *interpreted* bytecode
- This gave it a reputation for being slow
- More recent versions of Java have a sophisticated *just-in-time* compiler
- This allows the VM to generate fast code while the program is running
- https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python3-java.html

# Getting Started with Java

- We use Java version 11
- You will need to install the JDK (v11 or greater)
- Instructions in the first lab
- You can use whatever editor you prefer: VS Code, Vim, Emacs, Sublime Text, etc.
- If you want an IDE: Eclipse, Netbeans, Intellij Community

# Hello World

```java
public class HelloCITS {
    public static void main(String[] args) {
        System.out.println("Hello, CITS2005!");
    }
}
```

- Prints "Hello, CITS2005!" to the terminal
- Let's do a live demo!

# Running Hello World

```
$ javac HelloCITS.java
$ java HelloCITS
Hello CITS2005!
```

- First, compile the code: `javac HelloCITS.java`
- "javac" is the Java Compiler
- This creates a compiled "bytecode" file `HelloCITS.class`
- Second, run the compiled code: `java HelloCITS`
- More about using the terminal with Java in *Lab 1*

# Hello World Breakdown

```
public class HelloCITS {
    public static void main(String[] args) {
        System.out.println("Hello, CITS2005!");
    }
}
```

- Let's break down the code starting with the first line

# Hello World Breakdown

```
public class HelloCITS {
    public static void main(String[] args) {
        System.out.println("Hello, CITS2005!");
    }
}
```

- `public class HelloCITS` defines a "class" called HelloCITS
- Classes are the main way Java supports OOP
- Code in Java is (almost) always found inside a class
- A class is a type of object, and an object is just a thing
- A book is a *class* and the unit textbook "Java: A Beginner's Guide" is a definite *object*
- More on this in later lectures

## Hello World Breakdown

```java
public class HelloCITS {
    public static void main(String[] args) {
        System.out.println("Hello, CITS2005!");
    }
}
```

- There are many *keywords* in the example
- `public`, `class`, `static`, `void`
- These let us tell Java what we are doing (e.g., making a public class)

# Hello World Breakdown

```
public class HelloCITS {
    public static void main(String[] args) {
        System.out.println("Hello, CITS2005!");
    }
}
```

- After `public class HelloCITS` we see some curly braces $\{\ldots\}$
- These describe the *body* of the class
- Curly braces are used to group code together
- In this case, all the contents of a *class* are grouped together between the $\{\ldots\}$
- In our example, the class contains only a single *method*...

# Hello World Breakdown

```java
public class HelloCITS {
    public static void main(String[] args) {
        System.out.println("Hello, CITS2005!");
    }
}
```

- `public static void main(String[] args)` is a *method*
- The name of the method is `main`
- For now, ignore the `public static void` part and the `String[] args` part
- A method comprises a block of code that can be executed at any time
- The `{...}` surround that code block

# Hello World Breakdown

```java
public class HelloCITS {
    public static void main(String[] args) {
        System.out.println("Hello, CITS2005!");
    }
}
```

- `System.out.println("Hello, CITS2005!");`
- This is a "real" line of code that actually does something
- Prints the message to the terminal
- It is called a *statement*
- It contains two things
  1. A *method* call (`System.out.println(...)`)
  2. An *argument* to that method, the string `"Hello, CITS2005!"`

# Hello World Breakdown

```java
public class HelloCITS {
    public static void main(String[] args) {
        System.out.println("Hello, CITS2005!");
    }
}
```

- When you execute a class using `java HelloCITS`, it looks for a `main` method
- It then executes all the code inside the `main` method
- If it cannot find one that looks like the one above, you will see an error

```
Error: Main method not found in class HelloCITS, please define the main method as:
    public static void main(String[] args)
```

# Hello World Extended

- Take another look at the main method `public static void main(String[] args)`
- This part of a method is called the *header*
- The part between the curly braces {...} is called the *body*
- `String[] args` is a *parameter* of the method
- Next, we see how `String[] args` works

# Hello World Extended

```
public class HelloCITS2 {
    public static void main(String[] args) {
        System.out.println("Hello, " + args[0]);
    }
}
```

- The difference is `"Hello, " + args[0]`
- Compile (`javac`) and run (`java`) the code
- It gives an error!
- `Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0`

```
public class HelloCITS2 {
    public static void main(String[] args) {
        System.out.println("Hello, " + args[0]);
    }
}
```

- The problem is that the code is trying to access args
- We did not give any *arguments*
- Instead, run java HeloCITS2 test
- This prints Hello, test
- "test" is args[0]!

# String Concatenation

```java
public class HelloCITS2 {
    public static void main(String[] args) {
        System.out.println("Hello, " + args[0]);
    }
}
```

- Consider `"Hello, " + args[0]`
- The + operator allows us to *concatenate* two strings
- This creates a new string that is the two strings glued together (e.g., `"Hello, test"`)

```java
public class HelloCITS2 {
    public static void main(String[] args) {
        System.out.println("Hello, " + args[0]);
    }
}
```

- `String[] args` contain the arguments from the terminal
- `java HelloCITS2 a b c`
- `args[0] = "a"`, `args[1] = "b"`, `args[2] = "c"`
- `String[] args` is an *array* of *Strings*
- `String[]` is the *type* of args

# Loops

```java
public class HelloCITS3 {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println("args[" + i + "] is " + args[i]);
        }
    }
}
```

- Lots of new things here
- Lets run the code
- java HelloCITS3 a b c produces,

  args[0] is a
  args[1] is b
  args[2] is c

# Loops

```java
public class HelloCITS3 {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println("args[" + i + "] is " + args[i]);
        }
    }
}
```

- `for (int i = 0; i < args.length; i++)` is a *for loop*
- It iterates through the *indexes* (0, 1, . . . ) of args
- We use `args.length` to know when to stop iterating
- More on loops later

# Loops

```java
public class HelloCITS3 {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println("args[" + i + "] is " + args[i]);
        }
    }
}
```

- `System.out.println("args[" + i + "] is " + args[i]);`
- Uses string concatenation to create a message that looks like "args[1] = b"
- Notice how it worked with numbers (like 0, 1, and 2) as well as with strings like ("a", "b", and "c")
- Notice that we could access `args[i]`