CS620c Structured Programming Lesson 1

Joe Duffin

Email: Joseph.Duffin@nuim.ie

Acknowlegements

- The following people have contributed to the development of the lecturing and assessment material for this course.

  Mr. Robert Voigt

  Dr. Susan Bergin

  Dr. Aidan Mooney

  Dr. Diarmuid O'Donoghue

  Dr. Charles Markham

CS620c Module Topics Overview

Programming fundamentals: variables, types, expressions and assignment; simple I/O (Input/Output);

Conditional and iterative control structures (if statements , switch statements and while loops, for loops, do while loops);

Designing and writing java methods, return types, actual and formal parameter lists, the run time stack and the stack frame.

Problem solving: understanding and developing algorithms (recipes);

Implementing algorithms as simple programs.

Strings and string processing;
Use of class APIs for creating objects and calling methods;

Understanding **data abstraction and encapsulation**

OO concepts: Understanding **classes**, **objects**, **inheritance**, **polymorphism**, **method overloading**, **method overriding**.

Learning Outcomes

On successful completion of this module, students should be able to:

1 **Create algorithms** (recipes) based on the description of a problem (assignment)

2 **Understand, evaluate** their own algorithms (recipes);

3 **Write methods** (mini programs with a program) in java program code;

4 **Debug runtime errors** (find the cause of errors) ;

5 **Write object oriented** programs (create their own data types).

Day one Timetable

Time	Topic	Person
9:30 to 10:00	Setting up passwords	
10:00 to 11:15	Lecture	Joe
11:15 to 11: 30	Break	
11:30 to 1:00	Practical Session	Your assigned demonstrator
1:00 to 2:00	Break	
2:00 to 3:20	Lecture	Joe
3:30 to 5:00*	Practical Session	Your assigned demonstrator

* Friday of week 1 and week 2 you will have a written test. Friday of week 3 your final CA exam will take place.

Teaching and Learning Methods

August 28th to September 15th (Monday to Friday)

Learning methods each day

- ❑ Lecture preparation and review time for students 9:00 am to 9:30 am.
- ❑ Lecture at 9:30am to 11:15am
- ❑ 11:15 to 11:30 break
- ❑ Lab Practical 11:30am to 1:00pm
- ❑ Lunch break 1:00pm to 2:00pm
- ❑ Lecture 2:00pm to 3:25pm
- ❑ Lab Practical 3:30pm to 5:00pm

Assessment

This CS620c structured programming module will be assessed by continuous assessment. There will be one final exam **on Friday 15th worth 50%** of your total marks for the module. The other **50% is comprised of (a)** cumulative grades for all the laboratory programming assignments **(b)** two written exams on the Friday of the first and second week (~1hr 30 minutes each). You may also receive a multiple choice quiz as part of your assessment.

Optional: Extra laboratory support from 5:15 pm until 6:30 pm.
Labs will be open until 8:00pm each evening.

Resources

- **Moodle** https://moodle.cs.nuim.ie

– Contact information / Forum
– Lecture slides / Lab solutions (some solutions but not all)
– Useful Links and Resources
- PSC (Programming Support Page) of the programming support centre in the Computer Science Department.
- Your demonstrator group library of Java Books

- **Network Drives**

– X: To store your files on (your storage account up until three weeks after the module has ended.)

- **Save your work (often!) and keep track of where it is saved**

– Save work onto USB drives (as well as X:) if you can.
– Email files to yourself for safekeeping and if you have a Gmail account you can also store your work on your **google drive**.

Pass Grade

People taking this module are registered or intend to register on a number different courses. The general pass mark for this course is **40%.** Your should check with your relevant course coordinator for the required % mark necessary for you to progress past CS620c.

Different courses taking the CS620c module

- ❑ HDip in Information Technology
- ❑ HDip Software Development
- ❑ HDip in Data Analytics
- ❑ MSc in Data Analytics
- ❑ MA in Digital Humanities
- ❑ MSc in GeoComp
- ❑ MSc in GIS RS
- ❑ MSc in SoftEng
- ❑ Structured PhD students.

Note: In addition to the **CS620c** module code, this module is labelled with the code **CS820** (for structured PhD students) and **CS632** (For Data Analytics students). This module is worth different credit values depending on your course of study (speak to your course coordinator) We will use the Moodle account hosted by the computer science department for three weeks. When the module is finished, I will populate your official University Moodle page with the module material and I will provide you with your provisional grade score approximately three weeks after the module has finished.

Warning

Some students taking this module as part of a number of different courses, have extensive programming experience from other courses or previous study. It would be very unfair for any beginner programmer to compare him/her self against these experienced students. Please be mindful of this when you are looking around at the progress being made by your neighbours in the class room**.**

This module assumes **NO** previous experience of programming and it is taught as such.

The presence on this module of students with programming experience will NOT affect either the pace of delivery or the level of the content of this module as it is targeted at absolute beginners.

What to expect?

You often hear the line “I can only program in Java”.

But rarely hear “I can only drive a Ford Focus”.

At the end of this module you will be able to say “I understand the basics of programming” and the learning curve associated with learning other programming languages will be dramatically reduced.

Once you know Java, languages such a C# and C++ are so similar that the transition is relatively straightforward.

(Don’t believe me, ask your demonstrators)

Software Engineering, Development?

Programming and problem solving is an art in that it requires a good deal of imagination, ingenuity, and creativity. But it is also a science in that certain techniques and methodologies are commonly used. The term software engineering has come to be applied to the study and use of these techniques. The life cycle of software, that is programs, consists of 5 basic steps.

1.Problem analysis and specification (your programming tasks/problems)

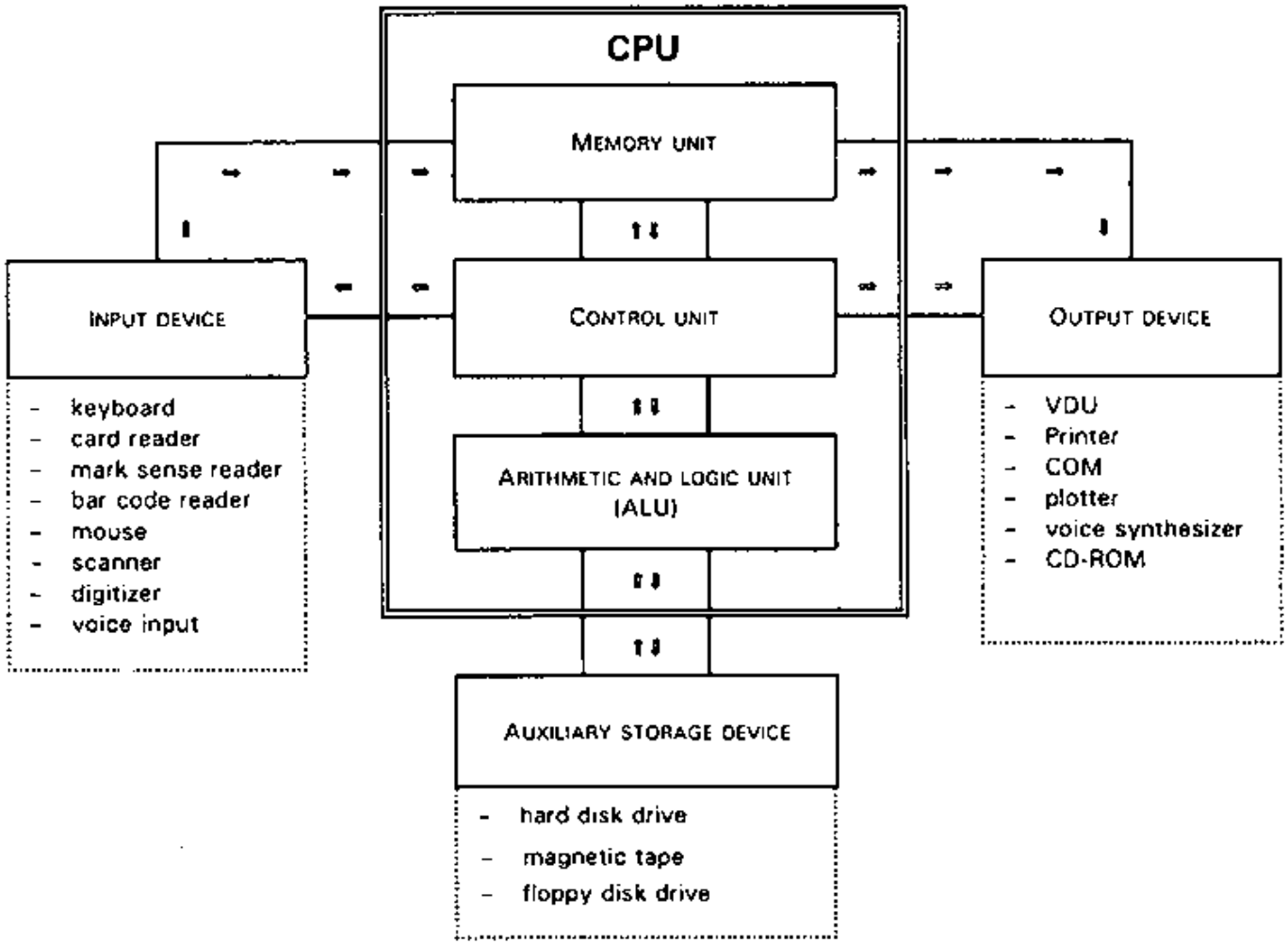
2.Algorithm development. (set in out a recipe to solve the tasks/problems)

3.Program coding. (converting your algorithm (recipe) into java code)

4.Program execution and testing. (compile and run your program (transformation))

5.Program maintenance. (make changes to your program to correct errors or make improvements)

Major components of a computer.



Computer Components (Units).

- **Input device.** This the “receiving” section of your computer. It obtains information (Data and computer programs) from input devices and makes this information available to the other units for processing.

- **Output device:** This is the “shipping section of the computer. It takes information that has been processed by the computer and places it on various output devices to make the information available for use outside the computer. (e.g. computer screen).

- **Memory unit** (memory or primary memory) : This is a rapid access, a relatively low capacity “ware house of the computer. It retains information that has been entered through the input unit so information is made available when needed.

- **Arithmetic and logic unit (ALU):** This is the “manufacturing” section of the computer. It is responsible for performing calculations such as addition, subtraction, multiplication and division. It contains the decision mechanism that allow the computer to compare two items from memory to determine if they are equal.

Computer Components (Units).

- **Central processing unit (CPU):** This is the "administrative" section of the computer. It is the computer's coordinator and is responsible for supervising the operation of the other sections. The CPU tells the input unit when information is should be read into the memory unit, tells the ALU when information from the memory should be used in calculations and tells the output unit when to send information from the memory unit to certain output devices.

- **Secondary storage unit (auxiliary storage device):** This is the long-term, high capacity "warehousing" section of the computer. Programs or data not actively being used by the other units normally are placed on secondary storage devices (such as disks) until they are needed again. Information in secondary storage devices takes much longer to access than information in primary memory.

Operating Systems

- The next ‘level’ up from physical parts

- Sits between our programs and the hardware of the computer

– Determines how our software ‘speaks’ to the Hardware

- Monitors and controls system activities

- **Manages memory** and other resources

– Scheduling

Software

The applications and programs that actually do the things we want to do

- Written in a variety of languages

– C/C++, Java, FORTRAN, BASIC, Python,
Perl, Ruby, PHP, …
– More on these later!

- Managed by the OS, running on the hardware, interacting with different parts of the machine

Tools of the Trade

SDK: Software Development Kit (and a simple text editor to type your code)

An SDK is a basic set of tools and software which allows you to write and create applications using a specific programming language.

http://www.oracle.com/technetwork/java/javase/downloads/index.html Java Platform (JDK) 7

IDE: Integrated Development Environment (BlueJ, JCreator, Eclipse)

An IDE provides a comprehensive set of tools for programmers, these are contained inside one application and make software development much more straightforward.

The **Editor**: Allows you to enter and change you program.

The **Compiler** (or interpreter): Converts your program (source-code, what we understand) to machine code (what the computer understands).

The **Debugger**: Allows you to tracks errors and watches internal operation of code

Possible choices of IDEs

BlueJ – The interactive Java environment

http://www.bluej.org/

Good system for learning Java will be used for CS620. Very good debugger.

JCreator®
SPEED, RELIABILITY, USER FRIENDLINESS

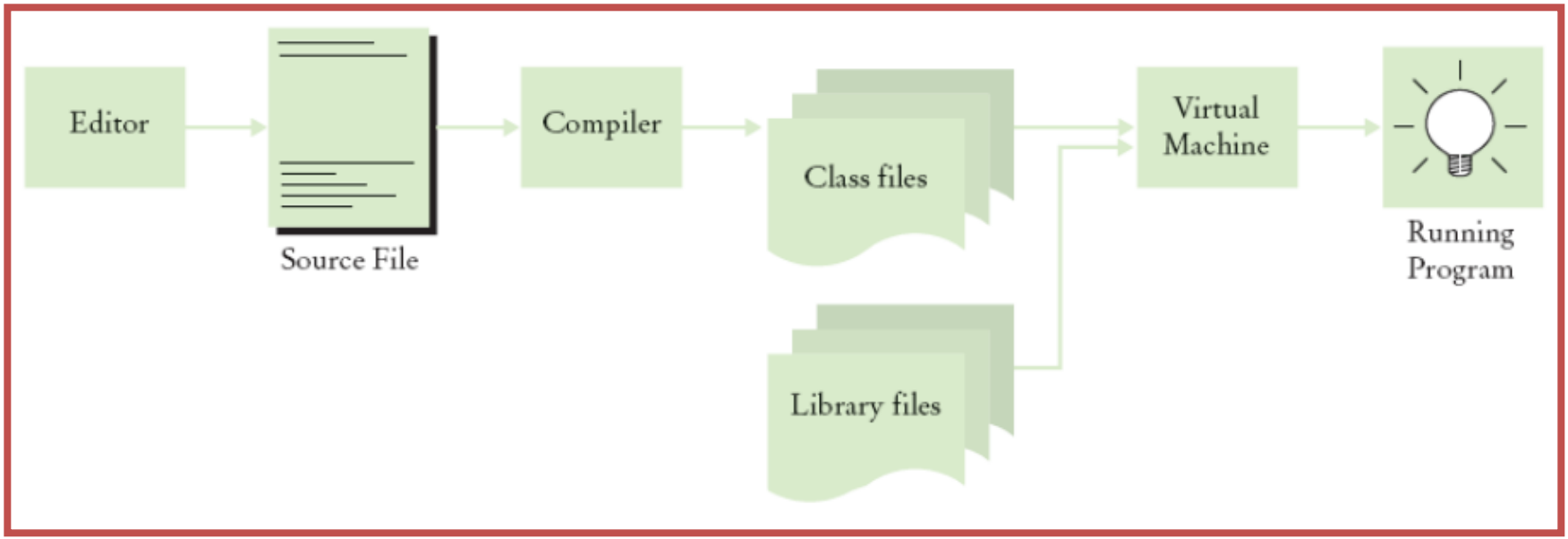
http://www.jcreator.com/

The free version is a lightweight system that provides a gentle start to Java but has no auto-complete or debugger.

http://www.eclipse.org/

Very comprehensive tool used by many professionals. It has auto-complete and debugger. Bigger learning overhead at start.

Programming Process



Examples of simple Editors → Notepad, **Notepad ++** (don't use Wordpad)

JDK Compiler → **javac** → the command **javac myprogram.java** results in the file **myprogram.class** being present in your directory.

JDK JVM interpreter → **java** → the command **java myprogram** runs or executes the file myprogram.class

The Programming Process

- **Write**, **Compile**, **Test**, **Fix or Update code**, **Repeat**

– 1. **Write** source code

– 2. **Compile** source code into bytecode / binary

– 3. **Test** the program

– 4. Go back to step 1 and make changes **if you find errors**.

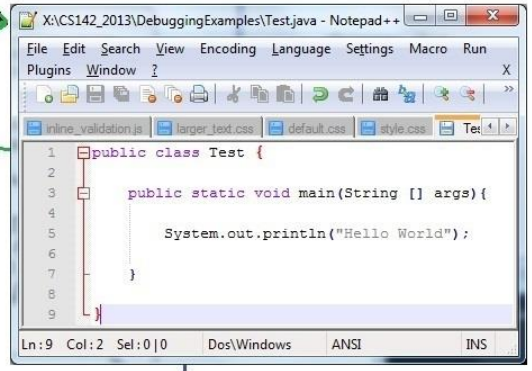
– 5. **Finish**

– 6. **Deliver** your code for assessment.

Write your Program (using NotePad++), **Compile** your program (using the **javac** command), and **Run** your program (using the **java** commnand)

Note: The Java program opened here in NotePad ++ is named **Test.java** This name also appears in the first line of your Java Program on the line

```
public class Test
```



Note: You edit or write your Java program code using NotePad++. You then go to the File button in NotePad++ and save the file as a **java source file** using the name:

Test.java

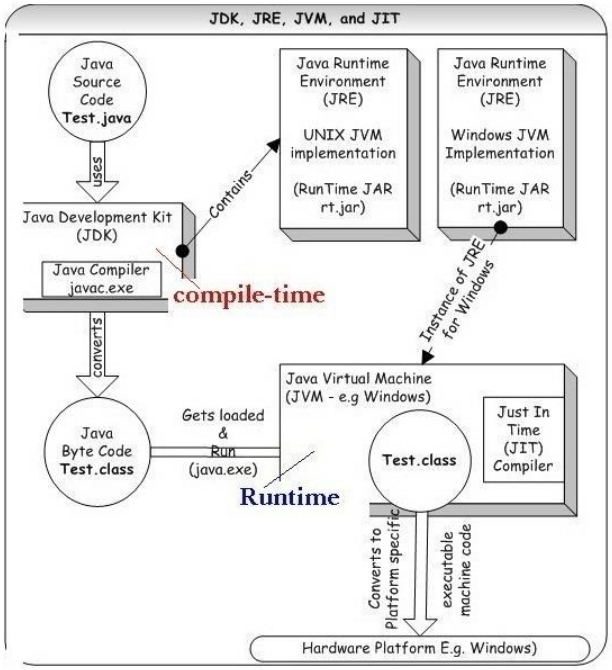
NB: This is a hard and fast rule: the name you save your code as and the name after the word class in the first line of your program, must be identical

Use the following Windows DOS commands:

`javac Test.java` **(compiles your code)**

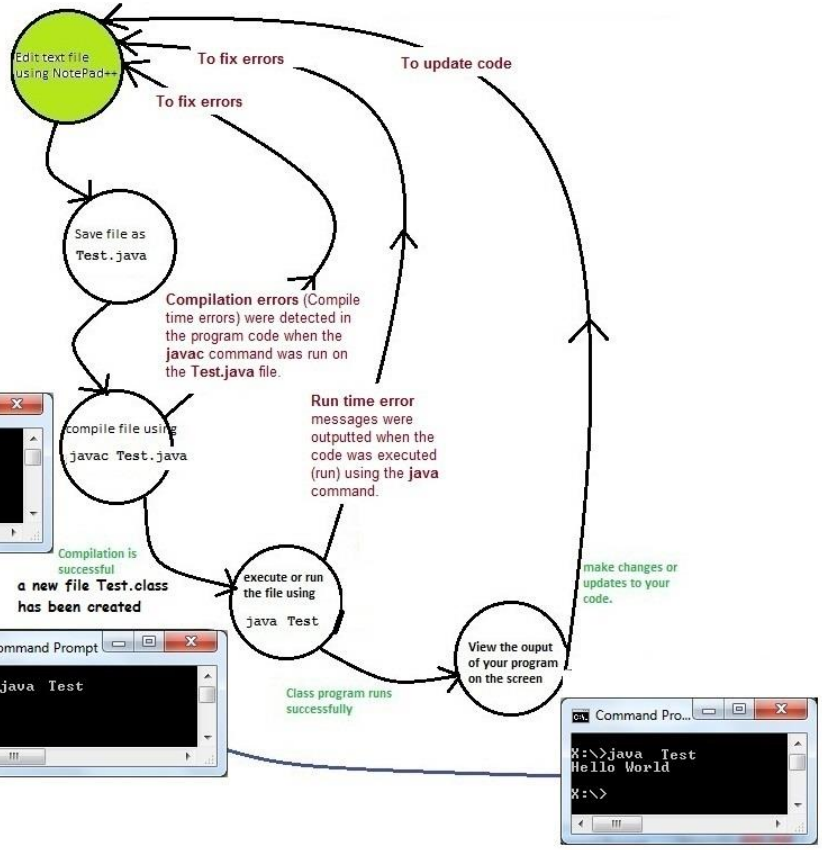
`java Test` **(runs your code)**

State diagram:
Illustrating the use of **NotePad++** to edit your program, **javac** to compile your program and **java** to run your program.



Another view of the software involved (JDK) in compiling and running any Java program that you will write on this module.

When writing, compiling and running your Java programs, leave open at the same time, both the NotePad++ editor and the DOS shell window. This will allow you to switch between these programs.



Type the following dir command to see that a new file named Test.class has been created.

X:\>dir

General Programming Process

- **How & Where do we write code?**

– In Java, you write code into a **simple text file**, which we call our 'source code file' (This is the text or java code that you will type using **NotePad++**)

– For a program called '*Hello' we'd put our code*
into a file called '*Hello.java'*

- **Turning source code into a working program**

– To turn that code into a working program it has to be ***compiled, which means translating*** the "human-understandable code" (the `.java` file) into something that your computer can understand and run.

Minimum Legal Java Program

```
// This is the most basic Java program possible.
// it will compile without errors and it will run but do nothing
// Learn this off by heart.

public class ProgramZero
{
        public static void main(String[] args)
        {

        }

}
```

A simple Java Program

- "Hello World!" – Everybody's first program
- Parts:

– Comment
– Class Definition
– Method
– Statement *(// Comment)*

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces - Comments

- **Comments (inactive annotations to your code)**

– Ignored by the compiler!
– Added for the benefit of people reading the code (including the author and your employer!)

– For documentation of the code (using javadoc, google “What is javadoc”)

– For keeping notes of what different parts of the code do

- **Every bit** as important as functional code

– One of the major differences between ‘good’ programming and ‘bad’ programming!

– Marks in this module and other programming modules will depend on good use of comments

Bits and Pieces - Comments

- Different kinds of comments:

– Single-line: // *Comment goes here*
– Multi-line: /* *Comment goes here* */
– Documentation: /** *Document info* */

- Used by the **javadoc** tool to automatically generate written documentation for Java software from comments in the source code. The javadoc tool “scrapes” your code for comments and produces professional looking documentation (html web pages) to tell other what your codes does and how it should be used.

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Class Definitions

Classes

– We'll explore the concept of classes in greater detail later

– For now, just treat your class and your program as the same thing

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Class Definitions

Classes

– The class name in this case is *HelloWorldApp*

– The Java compiler is pedantic

- **All code is case-sensitive**
- **helloWorldApp != HelloWorldApp**

– The source code filename for this program should be **HelloWorldApp.java**

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Methods

Methods

– Methods are the fundamental chunks of code in a program that perform some task(s)

– Every program will have a ‘main’ method, the starting point of whatever it is that program will do.

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Methods

- Methods (sometimes called functions)

– Every function has **modifiers, a name and a return type.**

- Below we can see that the function's name is *main*
- The method's return type is *void*
- The method has the modifiers *public and static*
- The return-type and modifiers will be explained later!

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Methods

- Methods

– Methods are functions inside a class

– In our example below, our class **HelloWorldApp contains just one method:** the ‘main’ method.

–

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Blocks

- Blocks

– The term ‘block’ refers to a section of code grouped together.

– Classes, Methods and Loops are typically written as blocks

– Blocks are usually enclosed by braces

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Braces

- Braces / Curly Brackets

– These brackets denote the starts and end of a block

– Everything between the curly brackets of the **HelloWorldApp block is part of the HelloWorldApp class**

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Braces

- Braces / Curly Brackets

– The code between the brackets of the main method is what will happen when the method is *called*

– The code between the brackets of the class is what will be considered part of the **HelloWorldApp class**

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Braces

- Indentation

– Like comments, indentation also makes the difference between *'good' and 'bad' code*

– **Readability**

– A program will still work regardless of good or bad indentation, but don't expect anyone to enjoy reading *(or marking!) it*.

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Braces

- Braces / Curly Brackets

– The code between the brackets of the main method is what will happen when the method is *called*

– The code between the brackets of the class is what will be considered part of the **HelloWorldApp class**

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Braces

• Braces / Curly Brackets (where am I on the keyboard?)
– The code between the brackets of the main method is what will happen when the method is *called*
– The code between the brackets of the class is what will be considered part of the **HelloWorldApp class**

```
1  /**
2   * The HelloWorldApp class implements an application that
3   * simply prints "Hello World!" to standard output.
4   */
5  class HelloWorldApp
6  {
7      public static void main(String[] args)
8      {
9          System.out.println("Hello World!"); // Display the string.
10     }
11 }
```

Bits and Pieces – Statements

• Statement
– Think of a statement as being like a command
– Telling the computer something you want to be taken as fact; or giving it a command
– **Always end with a semicolon ;**

• This tells the compiler where the end of the statement is

```
5   class HelloWorldAppString
6   {
7       public static void main(String[] args)
8       {
9           //System.out.println("Hello World!"); // Display the string. Old code commented out!
10          String helloStr = "Hello World! This is a string variable!";
11          System.out.println(helloStr); // Display the string VARIABLE.
12      }
13  }
```

Bits and Pieces – Variables

- Variables

– Places in memory to hold information in a program
– *helloStr is a 'String' variable, meaning it holds* a string of characters (i.e.; text)

– Variables are *declared & defined (int myNumber;)*

– Values are *assigned to them (myNumber = 17;)*

– *'Passed' to methods as arguments (squareNumber(myNumber);)*

```
1  class HelloWorldAppStringDecl
2  {
3      public static void main(String[] args)
4      {
5          String helloStr;                                    // Declaration
6          helloStr = "Hello World! This is a string variable!";   // Assignment
7          System.out.println(helloStr);                       // Passing
8      }
9  }
```

Bits and Pieces – Variables

- Passing variables to methods (this will be revisited)

– In **HelloWorldApp we're using the**
System.out.println() method to print our message to the screen.

– The first argument to that method is a String we want to print.
– We can concatenate (means add together) multiple strings to be sent as one to the method:

```
1  class HelloWorldAppStringDeclConcat
2  {
3      public static void main(String[] args)
4      {
5          String helloStr = "Hello World! This is a string variable!";
6          String helloStr2 = "This is another string variable!";
7          System.out.println(helloStr + " " + helloStr2);
8      }
9  }
```

Problems faced by a novice.

List of issues faced by a novice:

1) Learning to **interact with the computer**, e.g. use a keyboard, **learning to type** and locate the special characters and symbols used in a programming language. **(learn to touch type if you have the time)**

2) Learning to find your way **around the file system** of your account or computer. How do I write information into a file and where are my files saved to when I save them?

3) Learning to use an IDE (Integrated Development Environment) such as JCreator, **BlueJ** or Eclipse to write and compile a program.

Learning the special grammatical and **"syntax rules"** of a particular programming language. **Unlike natural spoken language you need to be precise** with the language text in order to be understood by the computer.

Learning the meaning or **"semantics"** of the special phrases in the programming language.

Learning to read a problem assignment "specification" and come up with an algorithm to solve it.

Best practice when starting out?

Write lots of very small "trivial" programs to try out some aspect of the language in isolation before you attempt a larger problem. **(Write ~100s/1000s of these)**. In addition to completing your assignments, you should experiment with small (~10 lines long) snippets of code.

Do NOT rely solely on the compiler (javac) to find your computer program bugs or errors. Develop a high degree of certainty about both the **Syntax** and **Semantics** of your chosen programming language.

Write your program in such a way that you can check to see if subsequent parts of your code are working. (**Don't** go for the "big bang" approach to writing a program and getting it to run correctly as designed).

Never take the short cut of using someone else's code as you will deprive yourself of an essential learning opportunity.

Use a pen and paper to develop your algorithms and to examine the trickier parts of your problem solution. Don't be tempted to start typing in to the computer straight away. Use **text and diagrams** to describe your solution.

Opinions and facts

For the reasons listed previously, your first months of programming can be **extremely challenging**. You need to master so many skills simultaneously and **it can be a very slow and difficult process.**

However, once you complete your first months, you will have the essential skills to tackle the subsequent computer programming related material and although the material becomes more advanced **you will have the skills to deal with it**.

If you learn to program with the Java programming language you will have a reduced learning curve for other high level and objected oriented languages.

The **unexpected return** for all your hard work, If you know Java you will know….

60% of the language C
70% of the languages C# and C++
60% of the language Visual Basic
60% of the language Python
50% of the language PhP
90% of the language javascript
……

And so on for many other high level procedural and/object oriented languages

Higher Diploma in Science (Software Development) - fulltime [ICT Skills Conversion] 2017-2018

Language used	Module Code	Module Name
PHP	CS130	DATABASES
	CS143	INTRODUCTION TO COMPUTER SYSTEMS
Java, C, C++, Python	CS210	ALGORITHMS & DATA STRUCTURES 1
JavaScript, PHP	CS230	WEB INFORMATION PROCESSING
Java	CS265B	SOFTWARE TESTING
	CS270	MEDIA PROGRAMMING
Java	CS353C	SOFTWARE PROJECT
Java	CS385	MOBILE APPLICATION DEVELOPMENT
Java	CS620C	STRUCTURED PROGRAMMING
Java	CS627B	OBJECT-ORIENTED PROGRAMMING
	SEDF5	SCIENCE (SOFTWARE DEVELOPMENT)

Opinion and facts (continued)

You get such a high return for your efforts due to the relationship between Java and these other programming languages because many high level languages have been designed to have the basic underlying fundamentals listed below.

1 **Sequence** (lines of code executed one after the other in sequence, sequential execution of code)

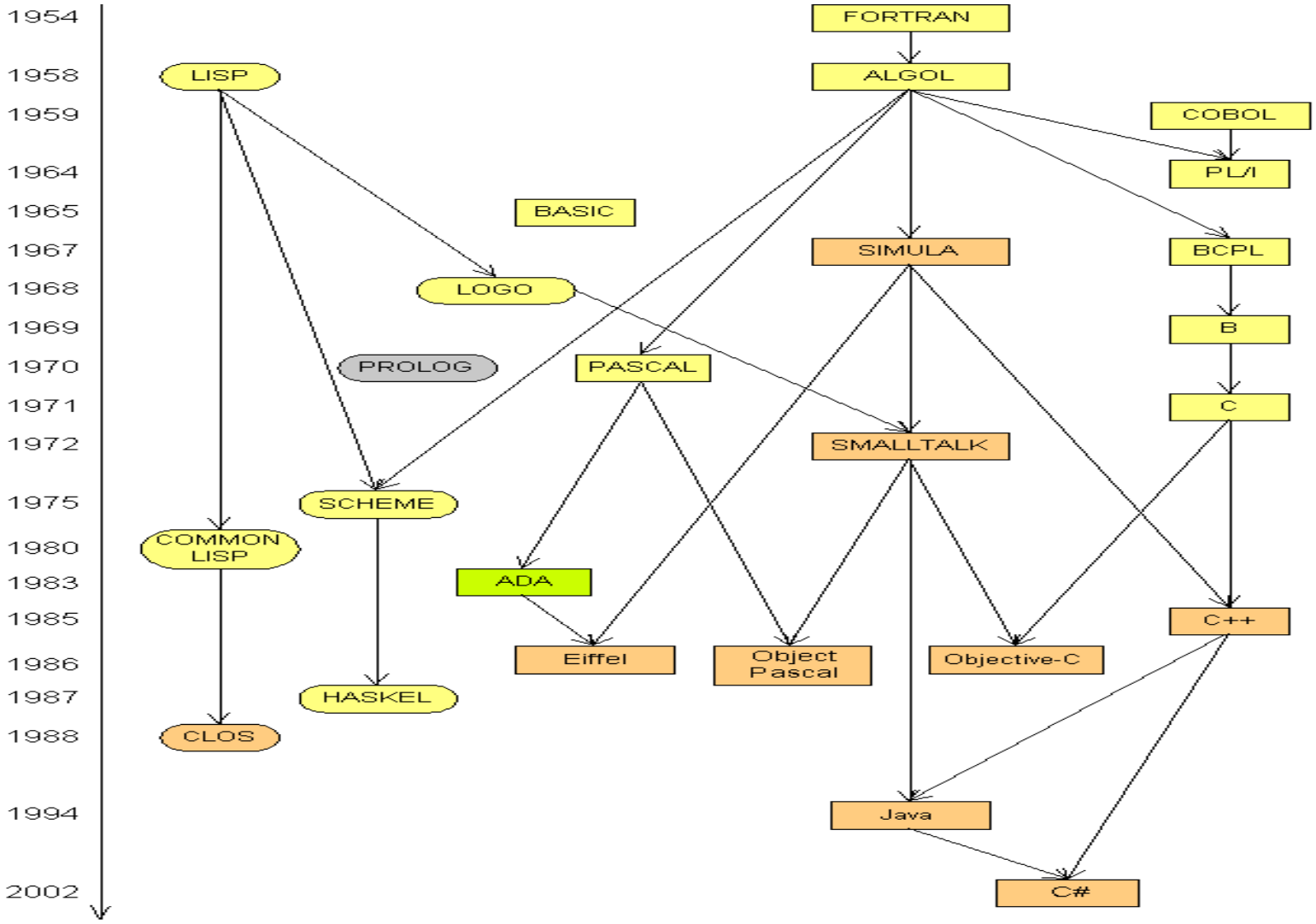
2 **Selection** (**if else, switch, ternary** operator, code is executed depending on test condition(s) being met)

3 **Repetition** (**while**, **do while**, **for** loops causing code to execute repeatedly)

• Once you know these concepts in one programming language, you know them for all other similar languages.

.

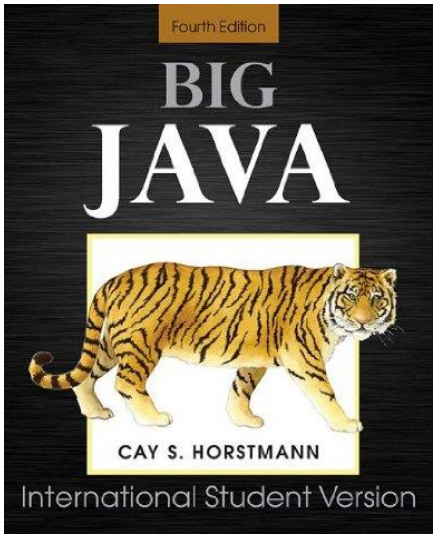
Java's Genealogy



Tracing the family tree of the development of different programming languages.

Recommended Texts

Big Java by Cay S. Horstmann
This book covers the current version of Java 7 and the proposed new version 8.
This book has been the recommended java course book for Computer Science students at NUIM for the last number of years. Strongly recommended.



Java How to Program by Deitel and Deitel.
I am using an earlier version of this book.
When I was learning the C programming language the book “C How to program” by the same authors was by far the best book available.