INTRODUCTION TO PROGRAMMING
LECTURE NOTES

INFO1103

TABLE OF CONTENTS

Week 1

Computer Basics:

A computer system consists of both hardware and software. Hardware refers to the physical components of the computer like the screen and keyboard. Software, however, refers to programs used to run the system, and is not physical.

The types of hardware used in each computer are very similar. Most computers have simple hardware like a keyboard, screen and mouse, and more complicated hardware like the Central Processing Unit (CPU). The CPU is the device inside the computer that performs the instructions of the program.

Memory:

Computers also have memory which holds data. There are two types of memory: Main Memory and Auxiliary Memory. Main memory devices, like RAM, hold the data of the programs currently being used. This type of memory is volatile, which means that it disappears once the computer is shut down. In contrast, auxiliary memory exists even after shutdown. This type of memory is held in devices such as hard drives and compact discs.

Memory is quantified in bytes. For example, 1 gigabyte of RAM is approximately 1 billion bytes of main memory. A byte is a small addressable unit of memory. The data of a single keystroke can be stored in one of these bytes. Each byte has an address which the computer uses to recover the data when necessary. A byte contains 8 binary digits (bits), which means that inside each byte is a combination of 1s and 0s which in different orders represent different types of data like letters or numbers. In auxiliary memory, bytes are grouped into much larger units known as files, and stored in a more permanent environment. Files can contain programs, pictures, text, etc. Once named, these files can be organised into folders or directories.

Programs:

A program is a set of instructions written for a computer to carry out a certain function. Anything you do on a computer uses programs, or software. Even just turning on the computer runs the operating system, which is a program that oversees the entire operation of the computer.

Programs are made of source code. Source code is written in programming languages such as C++, Visual Basic, and in this case Java. These are all examples of high-level languages, which are legible to us when we write in them. However, to be understood by the computer, it must be translated or 'compiled' into a low-level language called 'machine language' which cannot be deciphered by humans. We use a program called a 'compiler' to make this translation before we can run or execute the program. We input source code into the compiler, and it outputs object code.

The person who writes a program is called the programmer. The person who interacts with the program is called the user. It is the programmers job to create a simple environment with understandable instructions within the program, to make the program usable by the users.

Java the Universal Programming Language:

In most cases, a different compiler program is required to translate different programming languages on different types of computers. However, this is not the case for Java, since the Java compiler translates Java code into bytecode, which is easy to translate on almost all computer systems. The program that runs bytecode is called the Java Virtual Machine (JVM). To run a Java program, you first have to use a Java compiler to translate the program into bytecode, then use the Java Virtual Machine to translate the bytecode into machine language.

A Java program is hardly ever written as one piece of code. Instead, Java programs are comprised of different pieces of code called classes which are all connected together using a program called a class loader.

There are two main kinds of Java programs. Applications, which are regular programs designed to run on your computer, and Applets, which are programs designed to be sent to another location to run there, most probably within a web browser.

A Basic Program:

This is an example of a basic program called 'HelloWorld'.

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

When run, this program simply outputs the words 'Hello World'. We can break this program up into each different line to help us understand it.

```
public class HelloWorld {
```

'`public class`' defines the class or piece of code you are about to create as what is between the braces { and }.

'`HelloWorld`' is the name of the class and **must** be the exact same as what you named the text file when you created it, or what you are going to name it once you save it.

'`{`' is an open brace. Whatever is written between this brace and its close brace is part of the class HelloWorld.

```
public static void main(String[] args){
```

'`public static`' are words that define the accessibility of the main method.

'`main(String[] args)`' defines the main method which is a list of instructions executed by the computer when the program is run.

'`{`' is another open brace. Whatever is written between this brace and its close brace is part of the main method.

```
System.out.println(“Hello World”);
```

'`System.out`' is a special part of a class that allows us to use a method called println.

'`println()`' is a method that prints whatever is written in the brackets when the program is run.

'”`Hello World`”' is what will be printed when the program is run.

'`;`' is a semicolon which **must** be placed at the end of every statement, besides the beginning and ending ones where it will do nothing since there are braces there.

The final two lines consist of the close braces for each of the previous open braces.

Programs that allow the user to input information:

To allow programs to accept information input through the keyboard, we use the Scanner class, which is already built in to Java. So first we must import the class from a package called java.util. So we begin the program with the statement:

```
import java.util.Scanner;
```

Now to set things up so data can be entered from the keyboard we use the statement in the main method:

```
Scanner keyboard = new Scanner(System.in);
```

Lastly, where we want the user to input data, we use a statement like:

```
eggsPerBasket = keyboard.nextInt();
```

which will accept one integer from the keyboard. If we write a few of these last statements, integers entered must be separated by one or more spaces. Once the program has accepted input from the keyboard, it can now use that information to perform calculations and other functions. For example, we could use the statement:

```
System.out.println(“There are ” + eggsPerbasket + “ eggs per basket”);
```

Here is an example of a basic program that requires user input.

```
import java.util.Scanner;

public class InputProgram{
       public static void main(String[] args){
              System.out.println(“Hello there, write a word and I will repeat it”);
              String n1;
              Scanner keyboard = new Scanner(System.in);
              n1 = keyboard.nextLine();
              System.out.println(“You wrote the word: “ + n1);
  }
}
```

Writing, Compiling, and Running a Java Program:

Once you write a program in a text editor like Atom or Pluma, save the code as a java file by adding the .java extension. Next, open a command-line interface like terminal. Navigate to the folder the java file is saved in by typing ‘ls’ to view accessible folders, then in ‘cd folder_name’ to access the folder, then repeating until you have accessed the folder the program is saved in. Next, type in ‘javac filename.java’ and press enter to compile the code and the resulting bytecode is stored in a file with the extension .class. Finally, type in ‘java filename’ and press enter to run the program. Note that the filename **must** be the same name as the class.

If the program is not being compiled, then there is most likely an error in the code. Much of the effort used in programming is put into debugging errors. The command-line interface will outline the type and location of the error in the source code. A common error faced is the syntax error. Much of programming relies on the code written in a very specific way. Breaching or differing from this way even slightly will cause a syntax error. This can easily occur from accidentally leaving out a semicolon or capitalising a letter that should be lowercase, etc.

Variables and Expressions:

Variables in a program are used to store and label data such as numbers and letters to be used later in the program. The data inside a variable is called its value. To use variables, we must first declare some expressions as variables of a certain data type. Some of these data types include int (an integer), double (a number with decimal point), char (a single character), and boolean (true or false).

For example, the amount of eggs in a basket is an integer, so we write '`int NumberOfEggsInBasket`'.

Now that we have declared the variable, we can either allow the user to input a value as shown above, or use an assignment statement to give the variable a value of our choice. A simple assignment statement is '`answer = 42;`'. The equal sign is called the assignment operator. The expression we assign to the variable can also be an equation made with arithmetic operators like +, –, *, and /. A simple example of this is '`Score = NumberOfCards + NumberOfChips;`'. The same variable can appear on both sides of an assignment statement. For example, '`count = count + 10;`' is valid. This does not mean that the count is equal to itself plus 10, but instead it tells the computer to add 10 to the old value of the count. It is convenient to note that we can combine the variable declaration and the assignment statement. For example, '`int count = 10, num = 20;`'.

Identifiers:

The name of a class, method, or variable in Java is called its identifier. It cannot start with a digit, and can contain only letters, numbers, and the underscore symbol. It is also important to mention that Java is **extremely** case sensitive, so uppercase and lowercase letters are considered different characters. Generally, we always start the names of classes with uppercase letters, and we start the names of variables and methods begin with lowercase letters. The convention for naming classes, methods, or variables with more than one word is to capitalise the first letter of each word following the first. For example '`int numberOfCards`' or '`class HelloWorld`'.

Writing Decimals:

If a number has a decimal point, then it is not an integer. Instead we call it a floating-point number, and we use either the type '`float`' or more commonly the type '`double`'. We could write a floating-point number in floating-point notation simply like '2.5' or '865000000.0'. But for convenience, Java accepts numbers in 'e Notation'. This notation is like scientific notation except instead of the 'x 10^' we write an 'e'. For example, '8.65 x 10^8' would be written as '`8.65e8`'.

Constants:

Unlike variables, constants are values that do not change. When writing a large amount of code, Java provides us with a mechanism that allows us to name a constant for efficient use. The syntax is '`public static final Type VariableName = Constant`'. For example, we could give the name PI to the constant 3.14159 by using the statement:

```
public static final double PI = 3.14159;
```

The word '`final`' means that the value assigned to PI is final and the program is not allowed to change its value. The convention to naming constants is that the name is in uppercase and is separated by underscores if it is more than one word.

Type Casting:

A type cast changes the data type of a value. We could use a type cast to change the value 2.0 from double to int for example. Let's say we write the statement '`double distance = 9.0;`'. Now we cannot use the statement '`int points = distance;`' since we cannot assign a floating-point number to the integer class. But we can use the type cast '`(int)`' to make the statement legal: '`int points = (int)distance;`' which is essentially the same as writing '`int points = 9;`'. It is important to note that if the first statement was '`double distance = 9.99;`' then writing '`int points = (int)distance;`' would be the same as writing '`int points = 9;`'. The type cast does not round off. Instead, the type cast 'truncates' or discards the fractional part after the decimal point.

Assignment Compatibilities:

It is important to note that in some cases, the type cast is added automatically. For example 'double points = 7;' will automatically be read as 'double points = (double)7;'. The rule is that you can assign a value of any type to a type that appears further down the following list.

```
byte > short > int > long > float > double
```

Arithmetic Operators:

In Java, we can perform arithmetic equations using addition (+), subtraction (–), multiplication (*), and division (/). We can also use parentheses to group certain parts of equations. In these equations, we call variables and numbers 'operands'. In an equation containing multiple operands, if all of the operands are of the type int, then the result is of that type. However, if one of the operands is of a floating point type, then the result will be of a floating-point type. To be more specific, the result will be of the type used that is the farthest right in the list above. It is important to note that when using the division operator, the fractional part of a decimal may be truncated, such as in the statement '`int n = 9/2`' where they are both of the int class. Here '`n`' will be calculated as 4 rather than 4.5.

Java's Fifth Arithmetic Operator:

In Java, the '`%`' is also considered an arithmetic operator. It is called the remainder operator. We use the % to divide in the same way we would use the /, however, the remainder operator will calculate only the remainder. For example '`14/4`' yields a 3 if the operands are of integer class. However, '`14%4`' yields a 2 since 14 divided by 4 has 2 leftover. We generally use this operator to recover information equivalent to the fraction lost after the decimal point when dividing integers in Java.

The Increment and Decrement Operators:

There are two more operators that can be used in Java. These operators are used purely for convenience. The increment operator is written as '`++`' and simply increase the value of a variable by one. '`count++;`' is a quicker way of writing '`count = count + 1;`'. The decrement operator is written as '`--`' and simply decreases the value of a variable. These operators can be placed either before or after a variable. However, if used in an expression, if placed before, the variable will be changed before being used in the expression. If placed after, the variable will be used in the expression, and then it will be increased or decreased by 1.