COMP102 Introduction to Computer Programming

LECTURE NOTES
by
MV GWETU

What is programming?

- Computers were invented to allow humans to process information faster, more efficiently and reliably.
- In order to process information they rely on programmers to write instructions known as ***programming code*** which specify precisely how this is to be done.
- Computers consist of internal memory for holding instructions on how to process data as well a processor which executes these instructions.
- These instructions are written in a programming language. ***A high level programming language*** contains instructions that humans can easily understand while ***a low level language*** contains direct instructions for the computer.
- Software such as compilers are used to convert high level code to low level.
- Programming is therefore normally used to refer to the task of writing high level language instructions that are ultimately intended for the computer in order for it to solve particular tasks.
- After these high level instructions have been converted to low level instructions, they are executed on a computer processor.

Programming languages

- In order to be effective, human languages need to have rules so that people can understand each other without confusion.
- Programming languages are also guided by rules to enable consistency and clarity.
- ***Syntax*** rules are rules on how to write programming instructions, similar to grammatical rules for writing text in human languages.
- Since every program has a task it is meant to solve, programmers need not only write syntactically correct programs but those whose instructions will actually solve the task at hand.
- The logic of a program is known as its ***semantics***.
- There are different types of languages and each has its own style, syntax rules and strengths.
- A ***compiler*** is software that is used to check whether a program's syntax is correct and also convert it to an executable format.
- An ***Integrated Development Environment (IDE)*** is software that assists programmers to edit their programs efficiently and to avoid syntax errors.

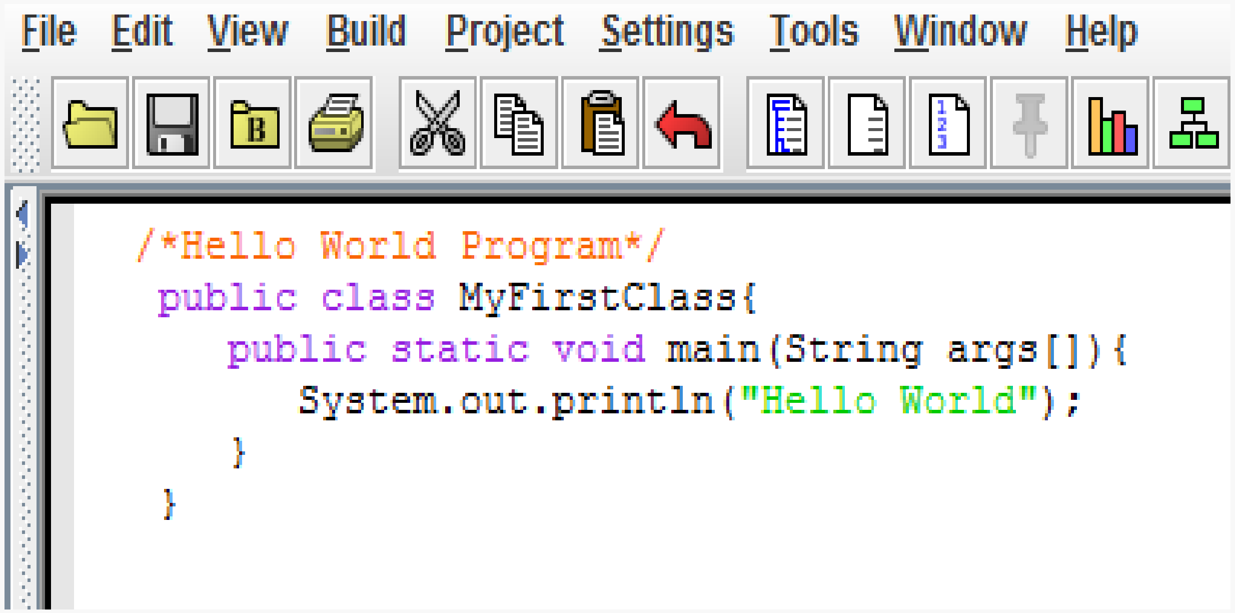
Object Oriented Programming (OOP) Languages

- ***Programming paradigms*** are the styles of programming which differ on how they express the solution to a problem.
- ***Functional programming*** is a paradigm which solves problems by breaking them up into subtasks, each of which calculates or performs a single value or task respectively. These subtasks can be combined to solve the main problem.
- ***OOP*** is a paradigm which designs virtual objects that can each store data (***attributes***) and perform functions (***methods***) which manipulate or access this data.
- The problem is then solved as these objects are created and instructed on how to work together to accomplish the main task.
- A ***class*** is the term used to refer to the design of each object.
- An object is ***instantiated*** when it is created based on its class.
- OOP therefore consists of two tasks:
  1) designing classes,
  2) and writing a ***main program*** to instantiate objects and instruct them on how to interact.

Java

- Java is an example of an OOP language.
- There are other OOP languages such as C#, Python and Delphi. Each has its own strengths.
- A ***Java Development Kit (JDK)*** is the software required to develop and execute java programs.
- The JDK consists of tools such as compilers for developing programs as well as a ***Java Virtual Machine (JVM)*** which is used to execute java programs.
- JDKs are created for specific environments such as operating systems and hardware.
- All code in java is written within a class, including the main program; therefore all java software will always consists of one or more classes.
- After a class is written, is it given to the compiler which checks it for syntax errors and if there are none, it creates ***java byte code*** which is a universal intermediate format.
- Byte code in itself can not be executed by a processor but it is merely a neutral format which all JVMs understand.
- Byte code can then be taken to any environment which has a JVM which will ***interpret*** it into that environment's low level language.
- When code is interpreted, instructions are converted to low level language and executed one at a time.
- Java is known to be ***portable*** in that regardless of where it is produced, byte code can be interpreted on any environment as long as it has a java virtual machine.

Java Hello World Program

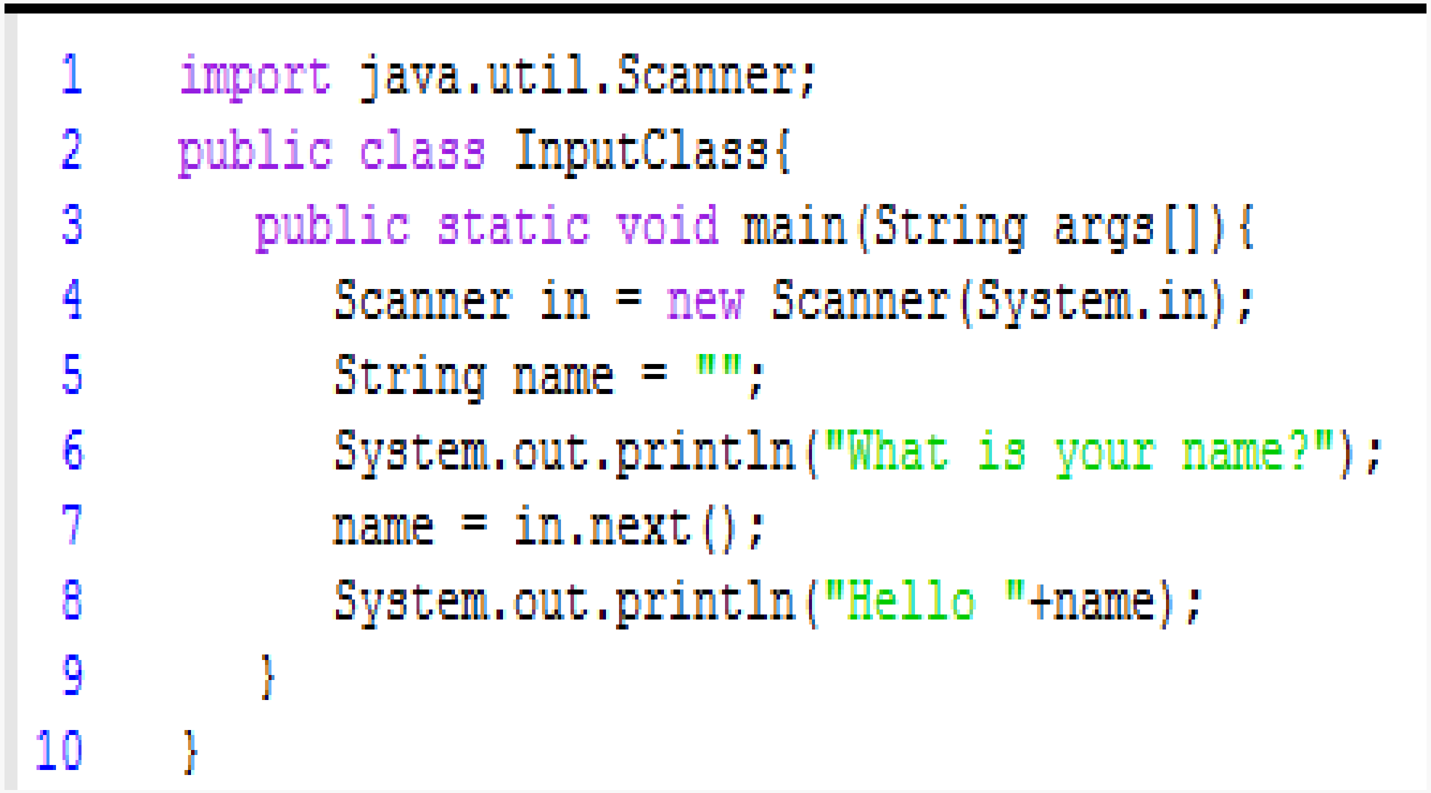


```
/*Hello World Program*/
 public class MyFirstClass{
     public static void main(String args[]){
         System.out.println("Hello World");
     }
 }
```

Hello World Program Explained

- This program has been edited using the JGrasp IDE. It has to be saved in a file with the same name as the public class. All java statements have to be within a class; they are normally within a method of a class.
- The ***comments*** are identified by the following enclosing characters /*…*/
- ***Key words*** are terms that are reserved and have a special meaning in a programming language. JGrasp has highlighted them in purple font colour.
- Each instruction/statement is terminated by a semicolon and blocks of instructions are enclosed by curly braces:{}. Every opening brace should have a corresponding closing brace.
- Every ***program*** in java is a class with a main method. Only programs can be executed directly.
- Execution starts in the main method, which contains the sequence of instructions/***statements*** to be executed.
- When it is run/executed, this program displays a hello world message.

User Input Example



```
import java.util.Scanner;
public class InputClass{
    public static void main(String args[]){
        Scanner in = new Scanner(System.in);
        String name = "";
        System.out.println("What is your name?");
        name = in.next();
        System.out.println("Hello "+name);
    }
}
```

User Input Example Explained

- Java contains some ***predefined classes*** with useful features that we can use in our programs.
- In this example we use the class ***Scanner*** which is found in the ***java.util*** package.
- When ever you want to use predefined classes you can ***import*** them so that they can be used without ***absolute name references*** in your class.
- Without the import statement, line 4 would have to look like this:

```
java.util.Scanner in = java.util.Scanner(System.in);
```

- When the program is run, it prompts the user for their name then displays a personal greeting.
- We define a ***variable*** of type String which is used to store the user input and another of type Scanner which is used to capture the users input.

User Input Example Explained: Strings

- Strings are a data type for storing text.
- The values of strings are in the form of characters enclosed in double quotes, for example: `“string”`
- The following is an example of ***declaring*** a string variable and ***initializing*** it with a value:

```
String s =“string value”;
```

- When a string’s value is displayed the double quotes are left out, however the enclosed characters are displayed as they are, including spaces.
- There are special characters that can be enclosed within string values such as `‘\t’` and `‘\n’` which represent a tab and new line respectively.

User Input Example Explained: String concatenation

- Two strings can be joined using the + operator. If `name` and `surname` are 2 string variables, then the following expression concatenates the 2 strings and puts a tab space between them: `name + ’\t’ + surname`
- The + operator can also be used to concatenate strings with numbers. For example:

```
“My favourite number is: ” + 24
```

- Be careful to leave a space at the end of the first string so that it formats properly when concatenated with the next variable.

The Anatomy of a Java Program: methods

- Class definition – generally all java code will exist inside a class. There are basically 2 types of classes: programs and non-program classes. For now we will focus on programs.

```
class HelloWorld { … }
```

- Main method – programs are classes which contain a main method. Program execution starts and stops in the main method.

```
class HelloWorld{
         public static void main(String[] args) { … }
}
```

- Other methods – At any point, the main method can transfer execution to (or call) another method which already exists. If this method is in the same class, its location in the class does not matter.

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

The Anatomy of a Java Program: statements

- A statement represents an instruction to be carried out by the computer. Statements can contain expressions, method calls and key words. A single statement is terminated by a semicolon while a block of statements is enclosed by curly braces.
- Keywords have a predefined meaning in java, there are over 50 such keywords.
- Import statements are one of the few types of statements that do not occur inside a method. In fact, they occur outside the class. They are used to access special features of java that are not available by default.

```
import java.util.*;
class Hello{
        public static void main(String[] args){
                ArrayList al = new ArrayList();
        }
}
```

- Examples of other java statements that occur inside methods:
  - Method call statement: `System.out.println(“Hello World”);`
  - Variable declarations: `int a = 6;`
  - Assignment statements: `a = 9;`
  - If and switch statements
  - loops

The Anatomy of a Java Program: Conditional statements

- If statements allow you to choose which of 2 instructions is executed depending on a condition.

```
if(age>18){
      System.out.println(“you are an adult”);
}
else{
      System.out.println(“you are not yet an adult”);
}
```

- Switch statements allow you to choose which of several instructions is executed depending on a condition.

```
switch(day){
      case 1: System.out.println(“it is Monday”); break;
      case 2: System.out.println(“it is Tuesday”); break;
      …
      case 7: System.out.println(“it is Sunday”); break;
      default: System.out.println(“invalid day of the week”); break;
}
```

- Conditional statements allow you to shorten an if-else statement into one instruction.

```
String output = ((age>18)? “you are an adult”:“you are not an adult”);
```

The Anatomy of a Java Program: loop statements

- Java offers for, while and do while loop statements for repeated execution of instructions.
- Indexed-for loops control the number of iterations using an initialization statement, termination condition and post cycle instruction. For example:

```
for(int i=0;i<10;i++){…}
```

- for-each loops simply loop through all the elements of a container with out an index. For example:

```
int[] array={1,2,3,4};
for(int a:array)
        System.out.println(a);
```

- While and do-while loops control repetition using an execution condition only. While loops check this condition before every cycle whereas do-while loops check it after every cycle. The following is an example of a nested loop using while and do-while loops:

```
int a=0,b=0;
while(a<5)
        do
                System.out.println((a++)*(b++));
        while(b<5);
```

The Anatomy of a Java Program: Scope

- Variables declared in a method are only accessible from the point of declaration to the end of the method.
- Variables declared in a class are called attributes.
- The components that found in a class are called its members; these are normally its attributes and methods.
- Class members are accessible anywhere (above or below the point of declaration) in the class.
- Class members can be declared using access modifiers such as **private**, **protected** or **public**.
- Private members can only be accessed from within the class while public members can be accessed from anywhere.
- The default access modifier restricts access to members of the same package. Protected scope will be introduced in COMP200.

Data types

- In java every variable must be declared with a type which influences the amount of memory reserved for the variable and the type of values it can store.
- Variables are made constant using the final keyword, for example: `final int A = 4;`
- Primitive data types allow variables to store data values. Examples include: int, long, float, double, char and boolean.
- Reference data types allow variables to store memory locations of other variables. Reference data types can be either inbuilt or user defined.
- The new keyword can be used to create objects whose location is stored by a reference variable. For example:

```
      String s = new String(“hello”);
```

Predefined reference data types: classes

- Besides arrays, reference data types are generally defined using classes.
- Classes are data types which can contain several member variables and methods.
- Predefined class are found in standard java packages. For example: `String` in `java.lang`, `ArrayList` in `java.util` and `Scanner` in `java.util`.
- The full name of a class includes its package, for example: `java.lang.String`; as opposed to the short name which does not show package information, for example: `String`. When a class full name is imported, it can then be referred to using its short name only.
- All classes in `java.lang` do not need to be imported; they can simply be referred to using their short names.
- It is possible to simultaneously import all the classes in a package. For example:

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

- The java documentation can be used to find out the exact members of a predefined java class.

Predefined reference data types: arrays

- Arrays are variables which contain several nameless variables of the same type. The number of elements in the array must be specified at declaration and can not be altered. For example:

```
int[] array = {1,2,3,4};//or int array[] = {1,2,3,4};
int[] array = new int[]{1,2,3,4};
```

- The elements of an array are accessed using a 0-based index. For example:

```
int array={1,2,3,4};
for(int i=0;i<array.length;i++)
        System.out.println(array[i]);
```

- The java.util.Arrays class has some useful predefined methods for manipulating (for example sorting and searching) arrays.
- Anonymous arrays are used to create nameless arrays that will only be used once. For example:

```
System.out.println(new int[]{1,2,3}[2]);
```

- Multi-dimensional arrays are simply arrays of arrays. For example:

```
int[][] matrix = {{1,2,3},{4,5,6},{7,8,9}};
for(int i=0;i<matrix.length;i++)
        for(int j=0;j<matrix[i].length;j++)
                System.out.println(matrix[i][j]);
```

Command-line arguments

- The main method is defined as follows:

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

- The **formal parameter** of the main method is an array of strings. This array can be used to provide input for the main method.
- After compilation, java programs (with the extension: .class) can be run from within an IDE or from the command prompt.
- The following commands are used to compile and run a java program on the command prompt, respectively:

```
javac HelloWorld.java
java HelloWorld
```

- When running a program, it can be provided with arguments, if required. For example, javac and java are actually java programs that require at least 1 argument which shows the file to be compiled or executed respectively. These arguments are referred to as command-line arguments.
- The main method is called by the java interpreter which extracts the command line arguments from the execution command places them in an array of strings: `args`. This is the **actual parameter** of the main method. To access the extracted command-line arguments, simply index `args`.

User defined data types

- Simple user defined data types can be created using enums, which are used to specify possible values for norminal data. For example:

```
enum colours { RED, BLUE, GREEN };
colours myColour = colours.RED;
```

- More complex user defined data types can be created using classes. For example:

```
class Student{
        public int studentNumber;
        public String name;
        public String surname;
        public boolean isRegistered;
        public int getYearsSinceFirstRegistration(){
                …
        }
}
```

- Every class has a constructor which initializes attributes to default values. If you do not specify one, java provides a **default constructor**.
- The new operator is used to create (or **instantiate**) a nameless object of a class using its specified constructor. This object must be referenced by a reference variable, otherwise it is recycled by the **java garbage collector**.
- The dot operator is used to access class members, for example:

```
Student student = new Student();
student.name = “Michael”;
```

- To find user defined classes, java searches through locations declared on the **class path**. This is why class names must correspond with class file names. Because the current directory is normally part of the class path, it is OK to simply save class files in the same folder as the program that uses them.

Static class members

- Although classes are primarily used as templates for creating objects, they can also store information that is shared by all objects created from that class.
- Any members of a class that are declared as static are not part of the template for creating objects, they are simply class information that is accessible at class level.
- Non-static members of a class are said to be **instance** members as they are part of the template used to create object instances.
- Static members are not able to access instance members but not vice versa.
- Static methods can be though of as utility methods that do not need to have access to instance members. For example we could add the following method to our Student class to validate student numbers:

```
class Student{
        …
        public static boolean isValidStudentNumber(int studentNumber){
                …
        }
}
```

- Because static and instance members can be accommodated in one class, it means we can actually have a class that acts as both a data type and a program. For example:

```
class Student{
        public int studentNumber;
        …
        public static void main(String args[]){
                Student student = new Student();
                …
        }
}
```

Recursive methods

- Recursive methods are special methods that can call themselves.
- Such methods generally consist of an if-else pair referred to as a base and recursive case.
- Recursion is often used to solve mathematical problems whose solutions are self-defined functions. For example: the Fibonacci sequence can be expressed as a function as follows:

$$fib(a) = \begin{cases} a & \text{if } a = 0 \text{ or } 1 \\ fib(a-1) + fib(a-2) & \text{otherwise} \end{cases}$$

- This function can easily be modelled as a recursive method:

```
public static int fib(int a){
        if(a==0 || a==1)
                return a;
        else
                return fib(a-1) + fib(a-2);
}
```