ECM1410 COURSEWORK

3 Concepts & Basics

3.1 Explain the differences between a compiled and an interpreted programming language.

A compiled language is a programming language which are generally compiled and not interpreted. It is one where the program, once compiled, is expressed in the instructions of the target machine; this machine code is undecipherable by humans.

An interpreted language is a programming language which are generally interpreted, without compiling a program into machine instructions. It is one where the instructions are not directly executed by the target machine, but instead read and executed by some other program.

In the compiled language, once the program is compiled it is expressed in the instructions of the target machine, while in the interpreted language the instructions are not directly executed by the target machine.

Compiled language: there are at least two steps to get from source code to execution;

Interpreted language: there is only one steps to get from source code to execution.

Compiled programs run faster than interpreted programs, while interpreted programs can be modified while the program is running.

Compiled language: compilation errors prevent the code from compiling; this language delivers better performance.

Interpreted language: all the debugging occurs at run-time; this languages delivers relatively slower performance.

3.2 Why Java is secure and platform-independent?

Java platform-independent

Java compiler produces a unique type of code called bytecode unlike c compiler where compiler produces only natively executable code for a particular machine.

When the Java program runs in a particular machine it is sent to java compiler, which converts this code into intermediate code called bytecode. This bytecode is sent to Java virtual machine (JVM) which resides in the RAM of any operating system. JVM recognizes the platform it is on and converts the bytecodes into native machine code. Hence java is called platform independent language.

Java secure

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

3.3 Explain the differences and similarities between variables and constants.

Constants: Data values that stay the same every time a program is executed are known as constants. Constants are not expected to change. **Literal constants** are actual values fixed into the source code. An example of this might be the character string "hello world". The data value "hello world" has been fixed into the code.

Variables: Variables are data values that can change when the user is asked a question, for example, their age. Variables may change during program execution.

Differences: their value remains/ do not remain the same

Similarities: Both constants and variables can only hold one piece of data at a time. When a new value is assigned to variable (using an assignment statement) it automatically replaces whatever was stored previously.

3.4 What are the 8 primitive data types in Java? What are the differences between the types? For each type (separate lines), give a Java statement creating a variable and assigning a value.

The 8 primitive data types in Java: boolean, byte, short, int, long, float, double, char.

Boolean: 1 bit; stores true or false values

Byte: 1 byte; stores whole numbers from -128 to 127

Short: 2 bytes; stores whole numbers from -32,768 to 32,767

Int: 4 bytes; stores whole numbers from -2,147,483,648 to 2,147,483,647

Long: 8 bytes; stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Float: 4 bytes; stores fractional numbers; sufficient for storing 6 to 7 decimal digits

Double: 8 bytes; stores fractional numbers; sufficient for storing 15 decimal digits

Char: 2 bytes; stores a single character/letter or ASCII values

```
boolean isJavaFun = true;

byte aNumber = 100;

short anotherNumber = 5000;

int myAge = 20;

long uglyNumber = 15000000000L;

float fractionalNumber = 1.57f;

double aNumberTo = 19.99d;

char myInitial = ‘y’;
```

3.5 What is casting? Explain the differences between implicit and explicit casting.

Taking an Object of one particular type and “turning it into” another Object type; this process is called casting a variable.

In Java, type casting is classified into two types:

- Widening Casting (Implicit)
- Narrowing Casting (Explicitly done)

Widening Type casting take place when,

- the two types are compatible
- the target type is larger than the source type

Narrowing or Explicit conversion:

- you are assigning a larger type value to a variable of smaller type
- If we don't perform casting then compiler reports compile time error

So, the differences between those two types of casting consist of compatibility.

3.6 What is overflow? Please, give an example of overflow.

Overflow happens when we assign a value that is out of range of the declared data type of the variable. If the (absolute) value is too big, we call it overflow.

If we define a variable *m* of type *int* and attempt to assign a value that is too big (e.g., *21474836478 = MAX_VALUE + 1).*

A possible outcome of this assignment is that the value of *m* will be undefined or that there will be an error.

3.7 What are the four main features of Object-Oriented Programming? Please, give a definition of each key feature.

The four main features of Object-Oriented Programming are:

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

Encapsulation is a mechanism of wrapping data (attributes) and code acting on the data (methods) together as a single unit. It is known as data hiding.

Inheritance is defined on the fact that classes can be derived from other classes, thereby inheriting fields and methods from that class.

Polymorphism: performing a single action in different ways

- Method overloading
- Method overriding

Abstraction means hiding all but relevant data in order to reduce complexity and increase efficiency. It is accomplished by using abstract classes and interfaces.