**HELLO THERE THIS THE KNOWLEDGE GAINED IN THE JAVA TUTORIAL.**

***Knowledge precedes understanding, understanding precedes change.***

JAVA can be used for mobile and enterprise applications.

Why use eclipse Integrated Development Environment?

1. It is free.
2. It is used industry-wide.

***HOW JAVA WORKS***

The code written in the IDE is the ***source code.*** It is converted into ***byte code*** using thee compiler. The byte code is then fed into the ***Java Virtual Machine.***

**Variables-** This are containers. They are called variables because you can change their value. ***Ensure that all variable names are logical. e.g. int num =5 instead of int a=5;***

**Primitive Data Types**.

* Character-2 bytes(16bits)
* Byte-1 byte- Stores range of -128 to 127
* Short-2 bytes- Stores range of -32768 to 32767
* Integer-4 bytes- Stores range of -2,147,483,648 to 2,147,483,647
* Float-4 bytes
* Long-8 bytes
* Double-8 bytes
* Boolean-N/A

**DISCLAIMER.**

By default, all decimal values in Java are of data type ***double.*** To use datatype ***float,*** we have to include ***f*** at the end if the value ***e.g. float 5.5f;***

This also applies to the data type ***long.***

***Implicit and explicit type conversion.***

**Implicit conversion*. e.g. double a =5 //when you print it, it prints out as 5.0.***

**Explicit conversion *e.g. int a= (int)5.5 // prints out 5.***  This is also called **type casting.**

**Naming Conventions.**

1. **Interface & Class –** The first letter **should be CAPITAL.**  For an interface, the name should be an ***adjective. e.g. Readable, Serializable.***  For a class, the name should be a ***Noun e.g. Student, Person, MyFirstJavaClass***
2. **Method –** The first letter is a **small letter.**  The name should be a ***verb. e.g. add, read***
3. **Constant - ALL WORDS** should be in **CAPITAL *e.g. PI, MAX\_PRICE***
4. **Packages & Variables –** Should be **small letters. *e.g. stockprice, bank\_details***
5. The right naming conventions make the code to be more readable.

**Logical Operators.**

***Ternary operator:***

Syntax: ***condition? expression1: expression2 –*** If the condition is ***true, the FIRST expression is executed, else the second operation is executed.***

The ternary operator is used to replace/shorten the if-else statement.

***Switch statement***

This is used for multiple cases where writing multiple if-else statements is cumbersome. It is only used with the **int and char data types.**

**Difference of print () and println ().**

Print () function prints out the output in the same line continuously, say when looping.

Println () function prints the output in a new line.

**Break and Continue statements.**

The break statement stops the loop.

The continue statement jumps the value stated in the condition. *e.g. You are printing numbers from 1 to 10 but want to jump 5. We specify an if-else condition i.e. if (n==5) {continue;* ***//jump 5 and continue looping}***

**Classes and objects.**

Classes are called the blueprint of building anything in Java. Objects are used to access the functionality of a class. To create an object in Java, we specify the keyword **new *e.g. Assuming our class name is Calc, to create an object we state as follows: Calc obj = new Calc ()***

***Constructors.***

A constructor is a member method with the ***same name as the class name.*** Every class has a constructor even when the user does not specify it. It is used to allocate memory to the objects in the heap memory.

**It does not return anything hence we do not specify the return type.**

**Any time you intend to use objects, you MUST include a constructor.**

It can be used to initialize values too. Every time you create an object, it calls the constructor automatically.

***Calc obj = new Calc ()*** This is a constructor.

**Use of “this” keyword.**

This keyword is used to specify the instance variable from the local variable. In a case where both variables have the same name *e.g. Two variables num1 and num2 in the class. When used in the constructor, we specify* ***this.num1(instance variable specified in the class) =num1(local variable specified in the constructor) and this.num2=num2***

**Method overloading.**

This is when you have the **same method name but different parameters.**

**WE CAN ALSO DO CONSTRUCTOR OVERLOADING IN THE SAME WAY.**

**Static keywords.**

***QUESTION: What are static methods? e.g. public STATIC void main (String args [])***

When we make a variable static, it will be same for all objects, i.e. it is not object specific thus when you change it in one object, it is the same for all objects in that class.

To access a **static variable**, ***we do not need an object*** i.e. Take for instance the static variable***, static String ceo*** in a class **Employee.** To access the variable, we **use the class name *i.e. Employee.ceo*** instead of using an object, say, ***obj.ceo.***

Methods can also be made static. e.g. **public static void main (String [] args). This means that we do not need an object to access it.**

When we compile a class, it is **first loaded in to a special; area in the JVM where classes are loaded**. After loading, we can then create an object, it is allocated memory in the **heap memory.**

A constructor is called every time a new object is created, thus for 2 objects, it is called twice. This means that any variable that has been initialized in the constructor is called twice too. For a static variable, **it requires to be called only once. As a result, we should not initialize it in the constructor. We initialize it in a** static **block *e.g. static {***

ceo=” Larry”***}***

The static block is executed first then the objects are created. **We cannot use non-static variable in a static block. To do that, we have to make the variable static.**

**A class within a class.**

We can create a class within a class. To access the inner class, we use the objects of the inner class. ***E.g. Assuming we have an OuterClass and InnerClass respectively.***

To create the object of the **Outer class**, we would write: **Outer obj = new Outer ();**

To create the object of the **Inner class**, we would write: **Outer.Inner obj1 = obj.new Inner ();**

**Arrays.**

**1D Array -** A collection of elements of the same data type.

**2D Array –** Arrays of same size. (not elaborate though)

**Jagged Array –** Arrays of different sizes

*Creation of Arrays.*

1. In Java, arrays are created like objects then the size specified in the brackets.

***e.g. int nums [] = new int [5];***

1. **If we know the values of the array** **beforehand, creation of an object is NOT necessary. *i.e. int nums [] = {1,2,3,4,5}***

When you create an array, **all values are set to the default value of 0.** We can also create an array of objects.

**2D Array.**

The number of elements in a 2D array is equal. i.e. {1,2,3,4}, {5,6,7,8}. All columns are equal.

**Jagged arrays.**

The number of elements in the columns are different.

Say we have an array S, to print it out, we specify the length of rows as **s.length** and the length of columns as **s[i].length.**

***Enhanced For Loop.***

This loop is used to avoid the repetition of code to print out all elements of an array. **FOR ARRAYS ONLY.** It is written as follows: **Say we have an array a, *for (int k:a){ System.out.println(k) }; //****this means that we do not have to keep track of the length of the array e.g. i<4 or i<=4 etc. The loop goes picking one value at a time in array a.*

**Variable Arguments.**

Sometimes we want to perform an operation on a number of parameters. When we create a method, we assign a number of parameters **e.g. public int add (int I, int j;)**

When we call the function, we pass the arguments equal to all the number of parameters in the method definition **i.e. add ()1,2)**

**However, at times we want to pass more arguments than the number of parameters specified. To do so, we use variable arguments in the method definition so that when we enter the arguments, the method reads the values entered as an array. This is done as follows:**

**Public int add (int … i) – If you any number of INTEGERS, they will be added.**

**Inheritance.**

It allows us to utilize the functionalities of an existing class in a new class. This involves the methods and variables of the existing class. The existent class from which is being inherited is called the **super/base/parent class** while the one which inherits is called the **sub/derived/child class.**

In Java, the syntax is: **class A extends B**

We can achieve **both single** and **multilevel inheritance.** When one class inherit a parent class, this is **single inheritance.** When another class inherits the class that had inherited the parent class, it is **multilevel inheritance.**

***IS-A and HAS-A keywords.***

**IS-A** refers to refer to a class that extends another class. **HAS-A** refers to a class in which the objects of another class are created.