Introduction to Java:

Java is an Object Oriented Programming Language developed by Sun Microsystems which was later taken over by Oracle Corporation. It is a high level programming language various types of applications including Android, Desktop and Web. There are many versions of Java released but for this training session we are using Java 8.

Low Level Programming Language is used by the machine to communicate in 1's and 0's whereas for the readability of the user the High Level Language was developed and few examples are Java, C++ etc.

Java files have '.java' as extension.

JDK

JDK also called as Java Development Kit contains tools used for developing and testing programs that are used for running Java.

To run Java on a system it needs JDK installed in it to be able to execute the programs and JDK can be found in oracle's website.

JDK includes JVM (JAVA VIRTUAL MACHINE) and JRE (JAVA RUN TIME ENVIRONMENT)

JVM acts an interpreter to communicate between JAVA and Operating System. JRE is used to interpret the code.

Eclipse is an IDE used to write the Java code.

Important terms in Java

Class – It is the blueprint/template for creating the objects.

Object – It is an instance of a Class.

Variable – It is a container used to store the values or data of various data types to utilize in the program.

Local Variable – It is valid in the body of the declared method.

Global Variable – It is declared at the initial state and is valid throughout the program.

Data Type – It is a used to distinguish data of different sizes and values that are stored in a variable.

Sample Program

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

Method - It is the code which is executed when it is called.

```
public class Test {
        int add(int a,int b){
                return a+b;
        }
        public static void main(String[] args) {
                Test t = new Test();
                System.out.println(t.add(1,2));
        }
}
```

Where the yellow text is method definition and green text is method call. The 'return' keyword is used to return a value of given type.

Console input is read and output is printed using the following code.

```
import java.util.Scanner;

public class Test {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in); //Input
        System.out.println(sc.next()); // Output
    }
}
```

```
<terminated>
Hello
Hello
```

Constructor - it is used for creating objects. It has two types default and parameterized.

```
public class Test {
    String productType = "Jeans";
    float amount = 100;
    Test(){
    }
    public Test(String productType,float amount){
        this.productType=productType;
        this.amount = amount;
    }
    public static void main(String[] args) {
        Test shop = new Test();
        System.out.println("Default\n");
        System.out.println(shop.productType);
        System.out.println(shop.amount);
        System.out.println("\nParameterised\n");
```

```
        Test shop2 = new Test("Shoes",200);
        System.out.println(shop2.productType);
        System.out.println(shop2.amount);

        }
}
```

Where highlighted Test() is default and Test("Shoes",200) is parameterised.

Comments – They are used to explain the code and are not executed.

Data Types

Primitive Data Type – int, byte, short,long,float,double,char,boolean

Non - Primitive Data Type – Array, String,Class,Inteface

1. single line comments – denoted by //
   // System.out.println("Hello World!");
2. multi line comments – starts with ./* and ends with */.

```
./*
    System.out.println("Hello World!");
    System.out.println("Welcome!");
    */.
```

Examples of declaration:

<u>int</u>

```
int i = 0;
```

<u>char</u>

```
char c = 'a';
```

<u>boolean</u>

```
boolean b = true;
```

double

double j = 15.5d;

float

float k = 10.0f;

String

String s = "Hello";

array

int w[] = new int[3];

w[0] = 1;

w[1] = 2;

w[2] = 3;

Conditions

The conditions used in Java are if,else if,else and switch case.

Examples are mentioned below

Test.java

```
import java.util.Scanner;

public class Test {

    public static void main(String[] args) {
        int marks = 55;
        if(marks>30){
            System.out.println("Pass");
        }
        else {
            System.out.println("Fail");
        }
    }
}
```

Console

```
<terminated> T
Pass
```

```java
import java.util.Scanner;

public class Test {

    public static void main(String[] args) {
        char c = 'd';
        switch(c){
        case 'a': System.out.println(c);
        break;
        case 'b': System.out.println(c);
        break;
        default : System.out.println("Invalid input");
        break;
        }
    }
}
```

```
<terminated> Test (1
Invalid input
```

Loops

The loops used in Java are for,while,do while,for each.

Example ‘for’ loop :

```java
public class Test {

    public static void main(String[] args) {

        int[] arr = new int[5];

        for(int i = 0;i<5;i++){
            arr[i] = i+1;
        }
        for(int i = 0;i<5;i++){
            System.out.println(i+1);
        }

    }
}
```

```
<terminated> Te
1
2
3
4
5
```

Example ‘for each’ :

```java
public class Test {

    public static void main(String[] args) {

        int[] arr = new int[5];

        for(int i = 0;i<5;i++){
            arr[i] = i+1;
        }
        // for each below
        for(int i:arr){
            System.out.println(i);
        }

    }
}
```

```
<terminated>
1
2
3
4
5
```

Example ‘while’ loop :

Test.java

```java
public class Test {

    public static void main(String[] args) {

        int[] arr = new int[5];

        for(int i = 0;i<5;i++){
            arr[i] = i+1;
        }
        // while loop below
        int j = 0; // initialization
        while(j<5){ // condition check
            System.out.println(j+1);
            j++;// increment
        }

    }
}
```

Console

```
<terminated> Te
1
2
3
4
5
```

Example ‘do while’ loop :

Test.java

```java
public class Test {

    public static void main(String[] args) {

        int[] arr = new int[5];

        for(int i = 0;i<5;i++){
            arr[i] = i+1;
        }
        // do while loop below gets executed at least once
        int j = 0; // initialization
        do{
            System.out.println(j+1);
            j++;// increment
        }while(j<5);// condition check

    }
}
```

Console

```
<terminated> T
1
2
3
4
5
```

Example Ternary Operator :

Test.java

```java
public class Test {

    public static void main(String[] args) {

        int i = 1, j = 2;
        // Ternary operator check if i and j are equal
        // print true(1) or false(0)
        System.out.println(i==j ? true : false);

    }
}
```

Console

```
<terminated> T
false
```

What makes Java an Object Oriented Programming Language?

There are four features which epitomises an Object Oriented Programming Language:

1. **Encapsulation: Data binding into single unit.**
2. **Abstraction : Hiding internal details and showing functionality**
3. **Inheritance: Acquiring all the properties and behaviours of a parent object.**
4. **Polymorphism: One task can be performed in many ways. → Method Overriding and Method Overloading**

Examples of Method Overloading and Overriding

Test.java

```java
public class Test {
    //Method Overloading
    int add(int a,int b){
        return a+b;
    }
    int add(int a,int b,int c){
        return a+b+c;
    }
    public static void main(String[] args) {
        Test t = new Test();
        System.out.println(t.add(1,2));
        System.out.println(t.add(1,2,3));
    }
}
```

Console

```
<terminated> T
3
6
```

Test.java

```java
public class Test {
    //Method Overriding
    void run(){
        System.out.println("This is test");
    }
    public static void main(String[] args) {
        Test t = new Test();
        t.run();
        Demo t2 = new Demo();
        t2.run();
    }
}
class Demo extends Test{
    void run(){
        System.out.println("This is Demo!");
    }
}
```

Console Outli

```
<terminated> Test (1) [J
This is test
This is Demo!
```

Packages

Packages are a functionality provide to encapsulate all the similar classes or subcategories.

Example:

package variable;

Package names should always be in lowercase.

Access Specifiers

Access specifiers are used to define a scope for a given for method, variable, field, constructor or class.

Access specifiers used in java are default, public, private and protected.

Demo.java | *Variable2.java

```java
package variable;

public class Demo {
    int a = 10;
    public int b = 12;
    private int c = 14;
    protected int d = 16;

    public int getC() {
        return c;
    }
    public void setC(int c) {
        this.c = c;
    }

}
```

```java
package variable.a;

import variable.Demo;

public class Variable2 extends Demo {
    public static void main(String[] args) {
            Variable2 v = new Variable2();
            int x = v.a;
            int y = v.b;
            int k = v.getC();
            int z = v.d;

    }
}
```

Here variable has ‘a’ default access type so it can’t be accessed in another class.

The variable ‘b’ being of public access type it can be accessed.

The variable ‘c’ is of private access type so to access it getters and setter need to be used.

The variable ‘d’ is protected access type so it can be accessed in the child class only when extends keyword is used

Static

Example for static

```java
public class Test {
    //Method Overriding
    static{
        System.out.println("This is static text!");
    }
    void run(){
        System.out.println("This is text from run method!");
    }
    public static void main(String[] args) {
        Test t = new Test();
        t.run();
    }
}
```

```
<terminated> Test (1) [Java Application] C:\Pr
This is static text!
This is text from run method!
```

Abstract Class

The parent class has addition method which has a body so it is not abstract unlike subtract and multiply so the keyword abstract is added before data type and to the parent class.

The child class when extends to the parent class the unimplemented methods which are abstract in parent class need to be implemented in the child class.

Child.java | Parent.java

```java
public abstract class Parent {

    public int addition(int a,int b){
        return a+b;
    }
    public abstract int subtract(int a,int b);
    public abstract int multiply(int a,int b);

}

```

Child.java | Parent.java

```java
public class Child extends Parent{

    @Override
    public int subtract(int a, int b) {
        // TODO Auto-generated method stub
        return a-b;
    }

    @Override
    public int multiply(int a, int b) {
        // TODO Auto-generated method stub
        return a*b;
    }

}
```

Interface

Interface is a blueprint of a class. It consists of static constants and abstract methods. It is used to achieve abstraction.

MyInterface.java

```java

public interface MyInterface {
    int i = 0; // static constants
    int add(int a,int b); // Abstract methods
}

class Operation implements MyInterface{

    @Override
    public int add(int a, int b) {
        return a+b;
    }

    public static void main(String[] args) {
        Operation o = new Operation();
        System.out.println(o.add(1, 2));
    }
}

```

Console

```
<terminated> (
3
```

The unimplemented methods are implemented in the Operation class. The implements keyword is used to provide the connection between the interface and the class.

File Reading and Writing

File reading done is using the File Class and File writing is performed using Filewriter and methods like write(). Here try catch are utilized in order to handle any exceptions which occur during execution.

Test.java | newFile.txt

```java
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class Test {
    public static void main(String[] args) {
        try{
        File f = new File("newFile.txt");
        FileWriter fw = new FileWriter(f);
        fw.write("Hello This is a new file!");
        fw.close();
        } catch(IOException e){

        }

    }
}
```

Output file is shown below.