PRACTICAL I

Objective: Study of JAVA & its Installation.

Theory: Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to byte code (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users. Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

The original and reference implementation Java compilers, virtual machines, and class libraries were developed by Sun from 1991 and first released in 1995. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java and GNU Class path.

PRINCIPLES

There were five primary goals in the creation of the Java language:

1. **It should be "simple, object-oriented and familiar"**
2. **It should be "robust and secure"**
3. **It should be "architecture-neutral and portable"**
4. **It should execute with "high performance"**
5. **It should be "interpreted, threaded, and dynamic"**

PLATFORM INDEPENDENT

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java byte code, instead of directly to platform-specific machine code. Java byte code instructions are analogous to machine code, but they are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware.

End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets.

Standardized libraries provide a generic way to access host-specific features such as graphics, threading, and networking.

A major benefit of using byte code is porting. However, the overhead of interpretation means that interpreted programs almost always run more slowly than programs compiled to native executables would. Just-in-Time (JIT) compilers were introduced from an early stage that compiles byte codes to machine code during runtime.

Java Platform, Micro Edition (Java ME) — targeting environments with limited resources.

Java Platform, Standard Edition (Java SE) — targeting workstation environments.

Java Platform, Enterprise Edition (Java EE) — targeting large distributed enterprise or Internet environments.

JDK

The Java Development Kit (JDK) is an implementation of either one of the Java SE, Java EE or Java ME platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, Mac OS X or Windows. Since the introduction of Java platform, it has been by far the most widely used Software Development Kit (SDK). On 17 November 2006, Sun announced that it would be released under the GNU General Public License (GPL), thus making it free software. This happened in large part on 8 May 2007, when Sun contributed the source code to the Open JDK.

The JDK has as its primary components a collection of programming tools, including:

- **appletviewer – this tool can be used to run and debug Java applets without a web browser**
- **java – the loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment. The old deployment launcher, jre, no longer comes with Sun JDK, and instead it has been replaced by this new java loader.**
- **javac – the Java compiler, which converts source code into Java byte code**
- **javadoc – the documentation generator, which automatically generates documentation from source code comments**

- **jar – the archiver, which packages related class libraries into a single JAR file. This tool also helps manage JAR files.**
- **jdb – the debugger**

JRE

The Java Runtime Environment (JRE), also known as Java Runtime, is part of the Java Development Kit (JDK), a set of programming tools for developing Java applications. The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files.

JVM

A Java virtual machine (JVM) is a virtual machine that can execute Java byte code. It is the code execution component of the Java platform. Sun Microsystems has stated that there are over 5.5 billion JVM-enabled devices.

TROUBLESHOOTING:

1) **MY COMPUTER -----> PROPERTIES -----> ADVANCED -----> ENVIRONMENT VARIABLES ---> PATH -----> EDIT -----> PASTE the PATH of JAVA COMPILER (javac) after ; and place ; after it also. Press OK then OK then OK.**
2) **MY COMPUTER -----> TOOLS -----> FOLDER OPTIONS -----> FILE TYPES ----> NEW ---> FILE EXTENSION is .java -----> OK -----> OK ---> CLOSE.**
3) **Open Command Prompt. Reach the Folder "DRIVE/PATH/JAVA/JDK VERSION/BIN" and type "set CLASSPATH=;" and press ENTER.**

Program:

```
class Message
{
public static void main(String[] args)
{
System.out.println("\n My name is Praveen");
}
}
```

OUTPUT: My name is Praveen

Result: The Study of JAVA and its Installation has been done successfully.

PRACTICAL II

Objective: Implementation of Simple Arithmetic Operations in JAVA

Theory: JAVA has a number of Arithmetic Operators as Tokens that can perform any Arithmetic Operations specified as Instructions or Statements. The various Arithmetic Operators supported by JAVA are:

1) +: It is used to perform Addition / Sum of Two or More Integer / Floating Type Variables.
2) -: It is used to perform Subtraction of Two Integers / Floating Point Variables.
3) *: It is used to calculate the Product of Two or More Integers / Floating Point Variables.
4) /: It is used to compute the Quotient / Division of Two Integers / Floating Point Variables.
5) %: It is used to generate the MOD / Remainder of Two Integers.

Program I:

```
class direct
{
public static void main(String[] args) {
int x=20, y=30, d=0;
d=x*y;
System.out.println("\n Product is :" + d);
} }
```

Program II:

```
class command
{
public static void main(String[] args) {
int x=0, y=0, d=0;
x=Integer.parseInt(args[0]);
y=Integer.parseInt(args[1]);

d=x*y;
System.out.println("\n Product of These Numbers is :" + d);
} }
```

Result: The Program in JAVA for performing Simple Arithmetic Operations has been Implemented successfully.

PRACTICAL III

Objective: Program in JAVA to take Input from User and generate the Output & to perform Type Casting.

Theory: In JAVA, there are four ways to take the Input from User as:

1) **Direct Assignment of Value to a Variable**
2) **Passing Value to a Variable at Run Time through Command Line Arguments**
3) **Design a Form as Graphical User Interface (GUI) and take Input in Fields specified**
4) **Taking Input from User through various available JAVA I/O Streams (Data Input Stream)**

Type Casting means to change the Data Type of a Variable whenever the need arises.

Program I:

```
import java.io.*;
class inout        {
public static void main(String[] args) {
DataInputStream dis = new DataInputStream(System.in);
int x=0, y=0, d=0;
try {
System.out.println("\n Enter First Number :");
x=Integer.parseInt(dis.readLine());
System.out.println("\n Enter Second Number :");
y=Integer.parseInt(dis.readLine());
}
catch(IOException e) {  }
d=x-y;
System.out.println("\n Difference of These Numbers is :" + d);
} }
```

Program II:

```
class type                    {
public static void main(String[] args) {
int a=0, b=0;
float d;
a=Integer.parseInt(args[0]);
b=Integer.parseInt(args[1]);
d=(float)a/b;
System.out.println("\n Division is :"+ d);
} }
```

Result: The Program in JAVA for User Input and Type Casting has been implemented successfully.

PRACTICAL IV

Objective: Program in JAVA to implement Conditional Statements

Theory: Conditional Statements are those which gets Executed whenever the Condition specified for their Execution is satisfied. These Statements are categorized as:

1) **Simple IF Statements: The Statements inside IF Block will be executed if the Condition specified is TRUE.**
2) **IF ELSE Statements: If Condition is TRUE then Statements inside IF Block will gets executed otherwise the Statements inside the ELSE Block will be executed.**
3) **IF ELSE Ladder: It is used whenever One Condition from Multiple Conditions specified will be TRUE.**
4) **Nested IF ELSE: Nested means IF inside another IF and ELSE inside another ELSE.**
5) **SWITCH CASE: It is used to remove the Drawbacks of IF ELSE LADDER. IT is also used when one out of multiple Conditions needs to be TRUE.**

PROGRAM I:

```
import java.io.*;
class condition       {
public static void main(String[] args)    {
DataInputStream dis = new DataInputStream(System.in);
int num=0;
System.out.println("\n Enter a Number : ");
try   {
num=Integer.parseInt(dis.readLine());
}

catch(Exception e)   {   }

if(num%2==0)
System.out.println("\n Number is EVEN");
else
System.out.println("\n Number is ODD");
} }
```

PROGRAM II:

```
import java.io.*;
class switchex {
public static void main(String[] args) {
DataInputStream dis = new DataInputStream(System.in);
int ch=0;


try {
System.out.println("\n 1 for Rectangle \n 2 for Square \n Enter Your Choice : ");
ch=Integer.parseInt(dis.readLine());
}
catch(Exception e) {  }
switch(ch) {
case 1:  {
int l=0, b=0, a=0, p=0, pr=0;
System.out.println("\n Enter Length & Breadth of Rectangle :");
try {
l=Integer.parseInt(dis.readLine());
b=Integer.parseInt(dis.readLine());
}
catch(Exception e)  {   }
p=l + b;
pr=2*p;
a=l*b;
System.out.println("\n Area of Rectangle is : " + a + "\n Perimeter of Rectangle is : " +
pr);
break; }
case 2: {
int s=0, a=0, p=0;
System.out.println("\n Enter Side of Square : ");
try  {
s=Integer.parseInt(dis.readLine());
}
catch(Exception e) { }
a=s*s;
p=4*s;
System.out.println("\n Area of Square is : " + a + "\n Perimeter of Square is : " + p);
break; }
default: {
System.out.println("\n Invalid Choice \n Please Enter a Valid Choice");
break;
}
}
}
}
```

Result: The Program in JAVA for Conditional Statements has been implemented successfully.

PRACTICAL V

Objective: Program in JAVA to implement Looping Statements.

Theory: Looping Statements are used to perform similar Operation a number of times. They are classified as:

1) **FOR Loop: It is mainly used with Arithmetic Operands. It is the simplest of all Looping Statements. It is an Entry Controlled Loop as Loop Statements executes only if the Condition is satisfied.**
2) **WHILE Loop: It is used with Character Operands mainly. It is also an Entry Controlled Loop.**
3) **DO WHILE Loop: It is also used with Character Operands mainly. It is an Exit Controlled Loop as it executes at least once even if the Condition specified inside the Loop is FALSE.**

PROGRAM I:

```
import java.io.*;
class fabonacci     {
public static void main(String[] args) {
DataInputStream dis = new DataInputStream(System.in);
int a=0, b=0, limit=0, c=0;
System.out.println("\n Enter Initial Values : ");
try {
a=Integer.parseInt(dis.readLine());
b=Integer.parseInt(dis.readLine());
}
catch(Exception e) { }
System.out.println("\n Enter the Limit : ");
try {
limit=Integer.parseInt(dis.readLine());
}
catch(Exception e) { }
System.out.println(a);
System.out.println(b);
for(c=a+b;c<=limit;c=a+b)   {
System.out.println(c);
a=b;
b=c;
} } }
```

Result: The Program in JAVA for Looping Statements has been implemented successfully.

PRACTICAL VI

Objective: Program in JAVA to implement Arrays & Strings.

Theory: These are commonly used User Defined Data Types. An Array is a Homogeneous Collection of Data, normally Numeric. The number of Elements present in the Array is called the Size of the Array. Each Element in the Array has a position commonly called Index. The Index of first element is usually 0. The Address of First Array Element is called the Base of the Array. The basic Operations to be performed on Array are Insertion, Deletion, Updation, Search, Traversal, etc.

Character Arrays are commonly known as Strings. The number of Characters present in a String is called the Length of the String. For String Operations, JAVA already has a STRING CLASS in JAVA.LANG Package that specifies the basic operations to be performed on Strings like Conversion to Upper Case and Lower Case Letters, Concatenation, etc.

PROGRAM I:

```
import java.io.*;
class array      {
public static void main(String[] args)   {
DataInputStream dis = new DataInputStream(System.in);
int n=0, i;
System.out.println("\n Enter the Size of Array : ");
try {
n=Integer.parseInt(dis.readLine());
}
catch(Exception e) { }
int[] num = new int[n];
System.out.println("\n Enter Elements of Array : ");
try {
for(i=0;i<n;i++)
num[i]=Integer.parseInt(dis.readLine());
}
catch(Exception e) { }
System.out.println("\n Elements of Array are : \n");
for(i=0;i<n;i++)
System.out.print("\t"+num[i]);
int min=num[1], max=num[1];
for(i=0;i<n;i++) {
if(min>num[i])
min=num[i];
if(max<num[i])
max=num[i];
}
System.out.println("\n MINIMUM ELEMENT is : " + min + "\n MAXIMUM ELEMENT
is : " + max);
}   }
```

Result: The Program in JAVA for Arrays & Strings has been implemented successfully.

PRACTICAL VII

Objective: Program in JAVA to implement Functions.

Theory: Function is a block of Code written in order to perform an Operation. It promotes Reusability as same Function can be used a number of times and at different places to perform the same operation. It also promotes Modularity. To use a Function, we have to declare and define it and then call it inside the MAIN(). Functions are classified as:

1) **No Return & No Argument Functions**
2) **No Return & Argument Functions**
3) **Return & No Argument Functions**
4) **Return & Argument Functions**

Program:

```
import java.io.*;
class function    {
public static void main(String[] args) {
DataInputStream dis=new DataInputStream(System.in);
int x=0, y=0, z=0;
System.out.println("\n Enter two Numbers : ");
try   {
x=Integer.parseInt(dis.readLine());
y=Integer.parseInt(dis.readLine());
}
catch(Exception e) { }
z=prod(x, y);
System.out.println("\n PRODUCT is : " + z);
}
public static int prod(int a, int b)
{
int t=a*b;
return t;
}
}
```

Result: The Program in JAVA for Functions has been implemented successfully.

PRACTICAL VIII

Objective: Program in JAVA to implement Constructors.
Theory: Constructor is a special type of Function inside Class whose name is similar to the name of the Class inside which it is defined. It has no Return Type not even VOID. It is called automatically whenever the Object of a Class is initialized. It is basically used to initialize the Variables of the Class.

Program:

```
import java.io.*;
class sum   {
int x, y;
sum(int a, int b)   {
x=a;
y=b;
}
public void calculate()  {
int z=x+y;
System.out.println("\n SUM is : " +z);
}   }
class constructor {
public static void main(String[] args) {
DataInputStream dis = new DataInputStream(System.in);
int m=0, n=0;
System.out.println("\n Enter 2 Numbers : ");
try {
m=Integer.parseInt(dis.readLine());
n=Integer.parseInt(dis.readLine());
}
catch(Exception e) { }
sum s= new sum(m, n);
s.calculate();
} }
```

Result: The Program in JAVA for Constructors has been implemented successfully.

PRACTICAL IX

Objective: Program in JAVA to implement Inheritance.

Theory: Inheritance means the concept of using the Data & Functions defined in a Class to be used in another Class. The Keyword used to perform Inheritance in JAVA is "EXTENDS". It promotes Reusability and is one of the Concepts of Object Oriented Programming Paradigm. It can be classified as:

1) **Simple / Single Level Inheritance: There is one Base Class and one Child Class.**
2) **Multi Level Inheritance: There is a Base Class and a Child Class, where a Child Class in turn acts as a Base Class for another Child Class.**
3) **Hierarchical Inheritance: There is one Base Class and a number of Child Classes.**
4) **Hybrid Inheritance: A Combination of Two or More Types of Inheritance Relationships.**

Inheritance Relationships are based on Access Modifiers like PRIVATE, PUBLIC, PROTECTED and PACKAGE / FRIENDLY. A Child Class can use the Data and Functions of Base Class based on the Type of Access Modifier to be used for Inheritance.

Program:

```
import java.io.*;
class first
{
String name;
DataInputStream dis = new DataInputStream(System.in);
public void getname()
{
System.out.println("\n Enter Name : ");
try
{
name=dis.readLine();
}
catch(Exception e) { }
}
public void showname()
{
System.out.println("\n Your NAME is : " + name);
}
}
```

```
class second extends first
{
int age=0;
DataInputStream dis = new DataInputStream(System.in);
public void getage()
{
System.out.println("\n Enter your Age : ");
try
{
age=Integer.parseInt(dis.readLine());
}
catch(Exception e) { }
}
public void showage()
{
System.out.println("\n Your AGE is : " + age);
}
}
class inherit
{
public static void main(String[] args)
{
second s = new second();
s.getname();
s.getage();
s.showname();
s.showage();
}
}
```

Result: The Program in JAVA for Inheritance has been implemented successfully.

PRACTICAL X

Objective: Program in JAVA to implement Applet using AWT Controls.

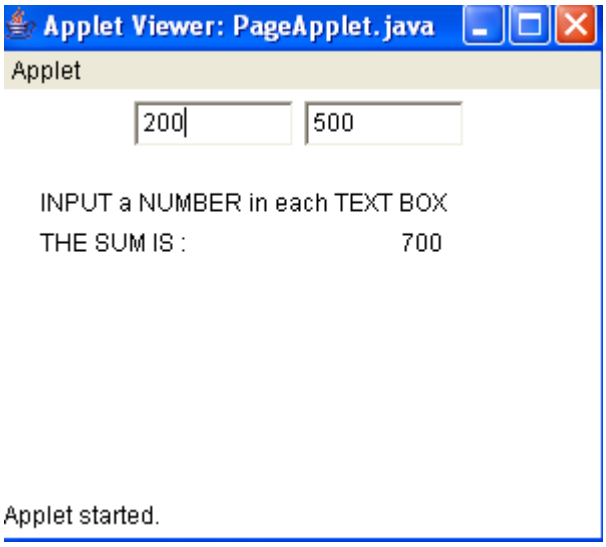
Theory: Applets are the Applications of JAVA that require the Web Browser for their Execution. For Applet Programming, some HTML Statements are embedded in JAVA File. First, the JAVA Applet will be Compiled and then executed on Browser or through the Applet Viewer. For this, it is compulsory to import the JAVA.APPLET.* Package in our File.

AWT stands for Abstract Window Tool Kit. It is used for designing of Front End Forms (GUI). For using AWT Controls, we have to import the JAVA.AWT.* Package in our Source File. Some AWT Controls are Labels, Text Boxes, Text Area, Buttons, Check Boxes, Radio Buttons, Drop Down Lists, etc.

Program:

```
import java .awt.*;
import java.applet.*;
public class PageApplet extends Applet
{
TextField txt1, txt2;
public void init()
{
txt1 = new TextField(8);
txt2 = new TextField(8);
add(txt1);
add(txt2);
txt1.setText("0");
txt2.setText("0");
}
public void paint(Graphics g)
{
int x=0, y=0, z=0;
String s1, s2, s;
g.drawString("INPUT a NUMBER in each TEXT BOX", 20, 60);
try
{
s1=txt1.getText();
x=Integer.parseInt(s1);
s2=txt2.getText();
```

```
y=Integer.parseInt(s2);
}
catch(Exception ex)
{
}
z=x+y;
s=String.valueOf(z);
g.drawString("THE SUM IS : ", 20, 80);
g.drawString(s, 200, 80);
}
public boolean action(Event evnt, Object obj)
{
repaint();
return true;
}
}
/* <applet code="PageApplet.java", width="300" height="200"> </applet> */
```



Result: The Program in JAVA for Applet using AWT Controls has been implemented successfully.