Set-1

1. Write about the role of JVM, JAVA API in developing platform independent java program with suitable example.

Ans The meaning of platform-independent is that the java compiled code (byte code) can run on all operatings systems.

Consider a simple java program of sum of two numbers:

⇒
```
import java.io.*;
Public class AddTwoNumbers{
    public static void main(String[] args){
        int n1 = 12; n2 = 24;
        int sum;
        sum = n1+n2;
        System.out.println(sum);
    }
}
```

Role of JVM while executing JAVA programs:

Java, being a platform independent programming language, doesn't work on one-step-compilation. It involves two-step execution, first through an OS independent compiler, and second JVM.

First, the source AddTwoNumbers.java file for above example or in general any *.java file is passed through the compiler, which then encodes the source code into Byte Code. The Content of each class contained in the source file is stored in a seperate AddTwoNumbers.class or *.class file.

The class files generated by the compiler are independent of the machine or the OS, which allows them to be run on any system. To run, the main class file is passed to the JVM, and then goes through three main stages before the final machine code is executed. These stages are

Class Loader:

The main class is loaded into the memory by passing its AddTwoNumbers.class file to the JVM, through invoking the latter. All the others classes if referenced in the program are loaded through the class loader.

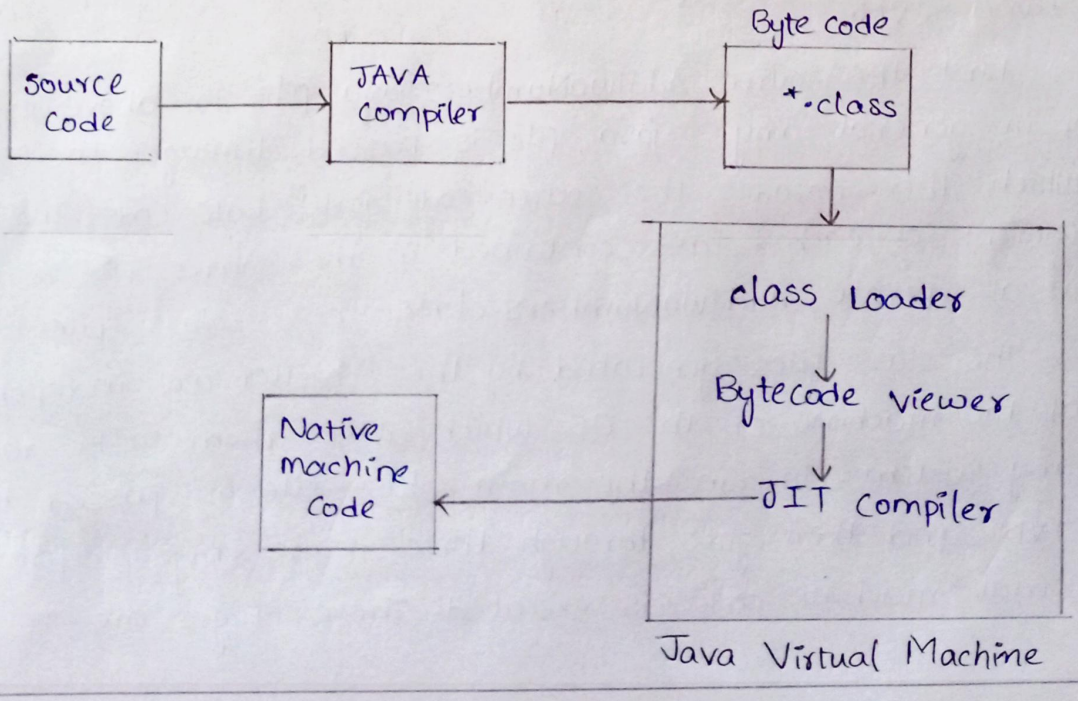
There are two types of class loaders: primordial, and non-primordial. primordial class loader is embedded into all the JVMs, and is the default class loader. A non-primordial class loader is a user-defined class loader, which can be coded in order to customize class-loading process.

Byte code Verifier:

After the Byte code of the AddTwoNumbers.class is loaded by the class loader, it has to be inspected by the byte code verifier, whose job is to check that the instructions don't perform damaging actions. some of the checks like:

1. Variables are initialized before they are used.
2. The runtime stack doesn't overflow.
3. Local variables accesses fall within the runtime stack, etc.

Just



Just-In-Time Compiler:

This is the final stage encountered by the java program, and its job is to convert the loaded bytecode into machine code. When using a JIT compiler, the hardware can execute the native code, as opposed to having the JVM interpret the same sequence of bytecode repeatedly and incurring the penalty of a relatively lengthy translation process.

Through these steps, JVM produces machine code which makes a java program platform independent.

Role of API:

The full form of API is Application Programming Interface. It is a document which gives us the list of all the packages, classes, and interfaces, along with their fields and methods. using this java API, we can know how to use the methods, fields, classes, interfaces provided by java libraries.

```
Public class Student{
        private String name;
        private int rollno;
        private int age;
        public int getAge(){
                return age;
        }
        public String getName(){
                return name;
        }
        public int getRollno(){
                return rollno;
        }
        public void setAge(int age){
                this.age = age;
        }
        public void setName(String name){
                this.name = name;
        }
```

```
        public void setRollno(int rollno){
                this.rollno = rollno;
        }
}
public class TestStudent{
        public static void main(String[] args){
                Student S1 = new Student();
                S1.setName("peter");
                S1.setAge(19);
                S1.setRollno(36);
                System.out.print(S1.getName()+S1.getAge()+
                                                S1.Rollno());
        }
}
```

The prewritten classes provided in the API provide a tremendous amount of functionality to programmer. In writing the above code, we can know the usage and syntaxes of writing different methods with the help of java API.

In java, most basic programming tasks are performed by the API's classes and packages, which are helpful in minimizing the number of lines written within pieces of code. The java API, included with JDK, describes the functions of each of its components. In Java, many of the these components are prewritten code via the java API. After refferring to the available API classes and packages, we can easily invoke the necessary code classes and packages for implementation.

The official API includes packages, e.g., applet packages, graphics and GUI swing packages, input/output (IO) packages and Abstract windows Toolkit (AWT), among others.

Source: geeksforgeeks.org/compilation-execution-java-program/
Techopedia/API/

2 With an example program explain the concept of classes and nested classes in java

Class:

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.

→ Creating a class Account maintaining instance variables attributes such as its name and balance and providing some methods to explain the concept of classes.

```
// Account.java
public class Account {
        Private String name ; // instance variable
        //method to set the name in the object
        public void setName( String name) {
                this.name = name;
        }
        //method to retrieve the name from the object
        public String getName( ) {
                return(name);
        }
```

Class Declaration:

```
public class Account
```

The keyword public is an access modifier. Each public class declaration must be stored in a file having the same name as the class and ending with .java file (Account.java).

Identifiers and Camel Case Naming:

class names begin with an initial uppercase letter, and method names and variable names begin with lowercase initial.

Instance Variable 'name'

An object has attributes, implemented as instance variables and carried with it throughout the lifetime. Instance Variables

exist before methods are called on an object, while the methods are executing and after the methods complete execution. Each object of the class has its own copy of the class's instance variables. A class normally contains one or more methods that manipulate the instance variables belonging to particular objects of the class.

```
private String name;
```

Access Modifiers public and private

most instance-variable declarations are preceded with the keyword private. Like public, private is an access modifier. Variables or methods declared with access modifier private are accessible only to methods of the class in which they are declared.

setName Method of class Account:

First line of method declaration is called method header. The method's return type specifies the type of data the method returns to its caller after performing its task. void represents no return value.

Method setName recieves parameter name of type String- which represents the name that will be passed to the method as an argument.

Parameters are declared in a parameter list, which is located inside the paranthesis that follow the method name in the method header. Each parameter must specify a type followed by a variable name. Variables declared in a particular method body are 'local variables'

setName Method Body:

Every method body is delimited by a pair of braces contain-

ing one or more statements that perform method task. If a method contains a local variable with the same name as instance variable, that method's body will refer to local variable rather than instance. 'this' keyword is used in method body to refer shadowed instance variable explicitly.

getName Method of class Account

Method getName returns a particular Account object's name to the caller. The method has empty parameter list. returns a String. A statement that calls getName on an Account object expect to recieve the Account's name as a String.

→ class Account cannot execute by itself because it doesn't contain a main method. You can either declare a seperate class that contains main method or place a main method in class Account.

```
// AccountTest.java
import java.util.Scanner;
public class AccountTest {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        Account myAccount = new Account();
        System.out.println(myAccount.getName());
        String theName = input.nextLine();
        myAccount.setName(theName);
        System.out.println(myAccount.getName());
    }
}
```

null
Aman
Aman. } output.

Nested classes:

In java, it is possible to define a class within another class, such classes are known as nested classes.

→ The scope of a nested class is bounded by the scope of its enclosing class

→ A nested class has access to the members, including private members, of the class in which it is nested. However, the reverse is not true.

→ A nested class is also a member of its enclosing class.

→ As a member of its enclosing class, a nested class can be declared private, public, protected, or package private (default)

→ Nested classes are divided into 2 categories.

1. static nested class 2. inner class.

Normal/Regular inner class	static Nested class
1. without an outer class object existing, there can't be an inner class object. That is, the inner class object is always associated with outerclass object.	1. without an outerclass object existing, there may be a nested class object. that is, static nested class object is not associated with the outerclass object.
2. static members cannot be declared.	2. static members can be declared.
3. As main() method can't be declared, regular inner class can't be invoked directly from Command prompt.	3. As main() method can be declared, the static nested class can be invoked from Command prompt.
4. Both static and non-static members of outerclass can be accessed directly.	4. only a static member of a outerclass can be accessed directly.

To create an object for the static nested class,

Syntax:

```
OuterClass.StaticNestedClass nestedObject = new
                                    OuterClass.StaticNestedClass();
```

// Java program to demonstrate accessing a static nested class

```
class OuterClass {
        // static member
        static int Outer_x = 10;
        // instance (non-static) member
        int Outer_y = 20;
        // private member
        private static int outer_private = 30;
        // Static nested class
        static class StaticNestedClass {
                void display(){
                        System.out.println("Outer_x = " + outer_x);
                        System.out.println("outer_private" +
                                                outer_private);
// We can't access outer_y because, we can't access non-static member
        // inside a static class.
                }
        }
}
public class StaticNestedclassDemo {
        public static void main(String[] args) {
                OuterClass.StaticNestedClass =
                        new OuterClass.StaticNestedClass();
                nestedObject.display();
        }
}
```

Output

```
Outer_x = 10
Outer_private = 30
```

To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object

Syntax:

```
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

```
// Java program to demonstrate accessing a inner class
class OuterClass {
        // static member
        static int outer_x = 10;
        // instance (non-static) member
        int outer_y = 20;
        // private member
        private int outer_private = 30;
        // inner class
        class InnerClass {
                void display(){
                        System.out.printf("%d %d %d", outer_x,
                                        outer_y, outer_private);
                }
        }
}
public class InnerClassDemo{
        public static void main(String[] args) {
                OuterClass outerObject = new OuterClass();
                OuterClass.InnerClass innerObject =
                                outerObject.new InnerClass();
                innerObject.display();
        }
}
```

Output:

```
10 20 30
```

Source: Java, How TO Program; Paul Deitel, Harvey Deitel
geeksforgeeks.org/nested-classes-java/

3. Design a class RailwayTicket with the following description:

Instance Variables/data members:

String name : to store the name of the customer

String coach : to store the type of coach the customer wants to travel.

long mobno : to store customer's mobile number.

int amt : to store basic amount of ticket.

int totalamt : to store the amount to be paid after updating the original amount.

Methods:

Void accept() : to take input for nome, coach, mobile number and amount.

Void update() : to update the amount as per the coach selected. Extra amount to be added in the amount as follows :

Types of Coaches	Amount
First_AC	700
Second_AC	500
Third_AC	250
Sleeper	None

Void display() : To display all details of a customer such as name, coach, total amount and mobile number.

Write a main() method to create an object of the class and call the above methods.

```
import java.io.*;
import java.util.Scanner;

class RailwayTicket{
        private String customer_name;
        private String coach_type;
        private long mobno; // mobile number
        private int amt; // basic amount of ticket
        private int totalamt;

        public void accept(){
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter your Name : ");
                customer_name = sc.nextLine()
                System.out.println("Types of compartments available:
In First_AC(700/-)Second-AC(500/-)Third-AC(250/-)Sleeper(None)");
                System.out.println("Choose the type of compartment");
                coach-type = sc.nextLine();
                System.out.println("Enter a valid mobile number
                                        (10 digits):");
                mobno = sc.nextLong();
                int a = (String.valueOf(mobno)).length();
                while(a!=10){
                    System.out.println("Warning! Enter valid
                                mobile number : ");
                    mobno = sc.nextLong();
                    a = (String.valueOf(mobno)).length();
                }
                System.out.println("Enter basic amount to
                        reach destination : ");
                amt = sc.nextInt();
                totalamt = amt;
        }
        public void update(){
                if(coach_type.equalsIgnoreCase("First_AC")){
```

```
                totalamt += 700;}
        else if(coach_type.equalsIgnoreCase("Second_AC")){
                totalamt += 500;
        }
        else if(coach_type.equalsIgnoreCase("Third_AC")){
                totalamt += 250;
        }
        else{
                totalamt += 0;
        }
        public void display(){
                System.out.println("..........");
                System.out.println("Name of the passenger : "+
                                                customer_name);
                System.out.println("Mobile Number".+ mobno);
                System.out.println("Compartment Type :"+ coach_type);
                System.out.println("Amount Paid" + totalamt);
                System.out.println("Happy and Safe Journey!!");
        }
        public void display(){
        public static void main(String[] args){
                RailwayTicket t1 = new RailwayTicket();
                t1.accept();
                t1.update();
                t1.display();
        }
}
```

```
Enter your Name :
Tony Stark
Types of Compartments available:
First_AC(700/-) Second_AC(500/-) Third_AC(250/-) Sleeper(None)
choose the type of compartment :
Second-AC
Enter a valid mobile number(10 digits):
9287631110
```

Enter the basic amount according to your ~~decision~~ destination :
50

Output :

.....

Name of the Passenger : Tony Stark
Mobile Number : 9287631110
Compartment Type : Second-AC
Amount Paid : 550
Happy and Safe Journey !!

4. Design a class to overload a function volume() as follows:

i) double volume(double r) - with radius 'r' as an argument, returns the volume of sphere using the formula.

$$V = \frac{4}{3} \times \frac{22}{7} \times r^3$$

ii) double volume(double h, double r) - with height 'h' and radius 'r' as the arguments, returns the volume of the cylinder using the formula :

$$V = \frac{22}{7} \times r^2 \times h$$

iii) double Volume(double l, double b, double h) - with length 'l', breadth 'b' and height 'h' as the arguments, returns the volume of cuboid using the formula

$$V = l \times b \times h$$

```
import java.io.*;
import java.util.Scanner;
class Volume{
        double Volume(double r) {
                return((4.0/3)*(22/7.0)*r*r*r);
        }
        double Volume(double h, double r){
                return((22.0/7)*r*r*h);
        }
        double Volume(double l, double b, double h){
                return(l*b*h);
        }
Public class VolumeCalculation{
        Public static void main(String[] args){
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter the radius of Sphere : ");
                double rad = sc.nextDouble();
                System.out.println("Enter radius of cylinder : ");
                double rad-c = sc.nextDouble();
                System.out.println("Enter height of cylinder : ");
                double height_c = sc.nextDouble();
                System.out.println("Enter length of cuboid : ");
               //System.out.println("Enter length of cuboid : ");
                double len = sc.nextDouble();
                System.out.println("Enter breadth of cuboid : ");
                double bredth = sc.nextDouble();
                System.out.println("Enter height of cuboid : ");
                double heit= sc.nextDouble();

                Volume obj1 = new Volume();
```

```
        double sphere = obj1.volume(rad);
        double cylinder = obj1.volume(rad_c, height_c);
        double cuboid = obj1.volume(len, bredth, heit);
        System.out.println("Sphere's Volume" + sphere);
        System.out.println("Cylinder's Volume" + cylinder);
        System.out.println("Cuboid's Volume" + cuboid);
    }
}
```

Enter the radius of sphere :
45.6
Enter the radius of cylinder :
6.7
Enter the height of cylinder:
8.9
Enter length of cuboid :
2.3
Enter breadth of cuboid :
4.5
Enter height of cuboid:
~~69.345~~ 6.7

Output:

Sphere's Volume 397335.99085714284
Cylinder's Volume 1667.936285714258
Cuboid's Volume 69.345

References

1st Question : geeksforgeeks.org/compilation-execution-java-program/
Techopedia/API/

2nd Question : Java, How to program, Paul Deitel, Harvey Deitel
geeksforgeeks.org/nested-classes-java/