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

A

Role of java virtual machine (JVM)

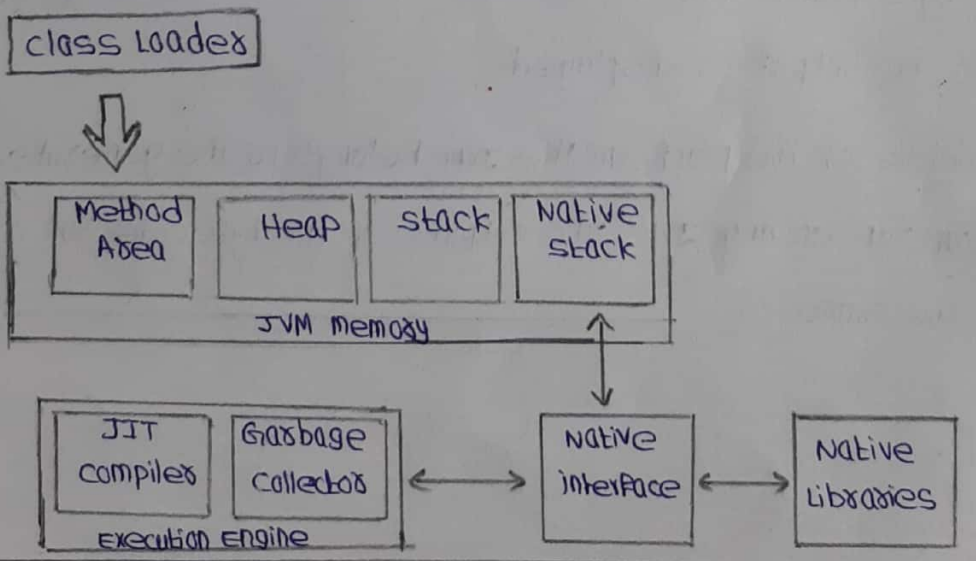
A java virtual machine (JVM) is a virtual machine capable of executing Java byte code. It is the code execution component of the Java software platform. A java virtual machine is software that is implemented on virtual and non-virtual hardware and on standard operating systems. A JVM provides an environment in which Java byte code can be executed.

JVM's are available for many hardware and software platforms. The use of the same byte code for all JVM's on all platforms allows Java to be described as a "write once, run anywhere" programming language, as opposed to "write once, compile anywhere", which describes cross-platform compiled languages.

The JVM performs following operations:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment.

The components of a JVM are shown in diagrammatic representation.



Java's security model has three components to look after: class loader, byte code verifier and security manager. the class loader loads all the required class files in to the disk. the byte code verifier ensures that the java programs have been compiled correctly, that they will obey the virtual machine's access restrictions, and the byte codes will not acess 'private data' the security manager implements a security policy for the virtual machine (VM). the security policy determines which activities the VM is allowed to perform, and under what circumstances.

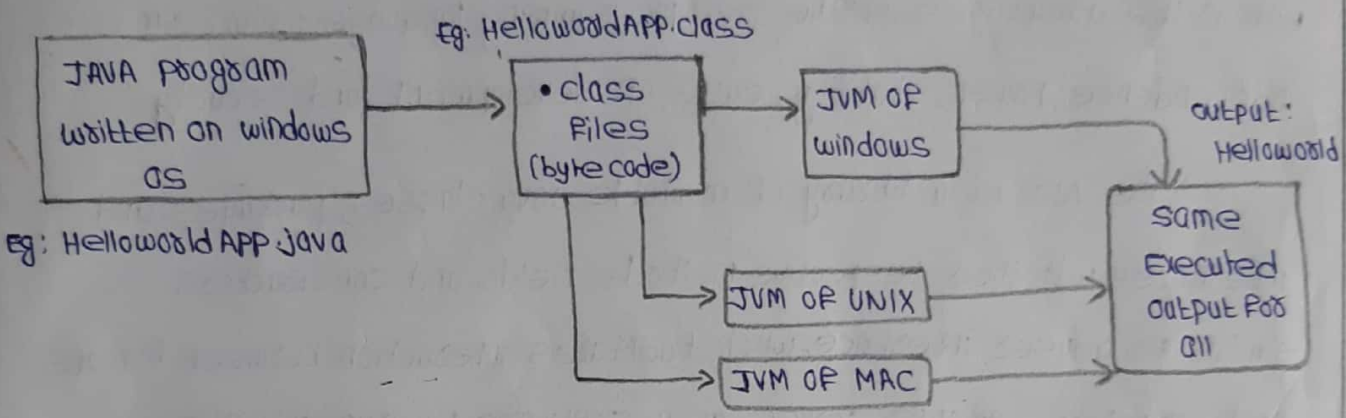
When a program is written and compiled in JAVA a separate file is created for a compiled program. the file (.class) is called bytecode in java. the .class file created cannot be executed directly. It does not include executable codes. Instead it will converted into executable code by a virtual machine in the system. these bytecodes generated by the compiled program are to achieve the purpose of platform independency. Byte code generated in a particular platform can be executed in any other platform i.e. the byte code generated in windows OS can also be executed in unix OS. the one which makes this possible is the JVM (JAVA VIRTUAL MACHINE). when the program is written and compiled the compiler send the generated bytecodes to the JVM resent in the machine and this JVM converts the byte codes into native code which is readable by a particular machine. thus the output is displayed.

Irrespective of the platform the JVM belongs to, the generated byte code can run on any JVM. the outputs of the bytecode run on any JVM will be the same.

Example Java program

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

This program can be saved as HelloworldApp.java



The above figure explains clearly about the program execution in JAVA.

The JVM fetches classes from a disk or from the network, and then verifies that the byte codes are safe to be executed. In addition to the byte code verifier checks for the following.

→ Access restriction violation.
→ object mismatching
→ operands stack over or under flow.
→ Incorrect byte code parameters.
→ Illegal data conversion.

Role of JAVA API

All Java platforms consist of a Java virtual machine (JVM) and an application programming interface (API). An API is a collection of software components that you can use to create other software components or applications. Each Java platform provides JVM and an API, and this allows applications written for that platform to run on any compatible system with all the advantages of the Java programming language: platform-independence, power, stability, ease-of development, and security.

The API is a library of available Java classes, packages and interfaces with their respective methods, fields and constructors. Similar to a user interface, which facilitates interaction between humans and computers, an API serves as a software program interface facilitating interaction.

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 three API types are as follows:

→ official Java core API, which is bundled with JDK download.

→ optional official Java API's, which may be downloaded if needed.

→ unofficial API's, which are third party API's that may be downloaded from source websites.

The API's help programmers determine class or package functions, parameters and other necessary information. 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.

Roll no: 19BQ1A05D7

There are three frames when an API starts, as follows:

1. The first frame shows all API components (classes and packages).
2. When a particular package is selected, the second frame shows all the interfaces, classes and exceptions of that particular package.
3. The third and primary frame provides an overview of all of API packages, which can be expanded in the main frame to show the index, class hierarchy and help sections.

Example: When you use an application on your mobile phone, the application connects to the internet and sends data to a server. The server then retrieves that data, interprets it, performs the necessary actions and sends it back to your phone. The application then interprets that data and presents you with the information you wanted in a readable way.

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

A

Java class

Before you create objects in java, you need to define a class. A class is a blueprint for the object.

We can think of the class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

Since many houses can be made from the same description, we can create many objects from a class

Syntax to define a class in Java

```
class className {
    // variables
    // methods
}
```

Example

```
public class Dog {
    String breed;
    int age;
    String color;
    void barking() {
    }
    void hungry() {
    }
    void sleeping() {
    }
}
```

A class can contain any of the following variable types

Local variables: variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

Instance variables: Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

Class variables: class variables are variables declared within a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry(), and sleeping() are methods.

Constructors

When discussing about classes, one of the most important subtopic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example for constructor

```
public class Puppy {

    public Puppy() {
    }
    public Puppy(String name) {
        // this constructor has one parameter, name
    }
}
```

Java also supports singleton classes where you would be able to create only one instance of class.

Nested classes in java

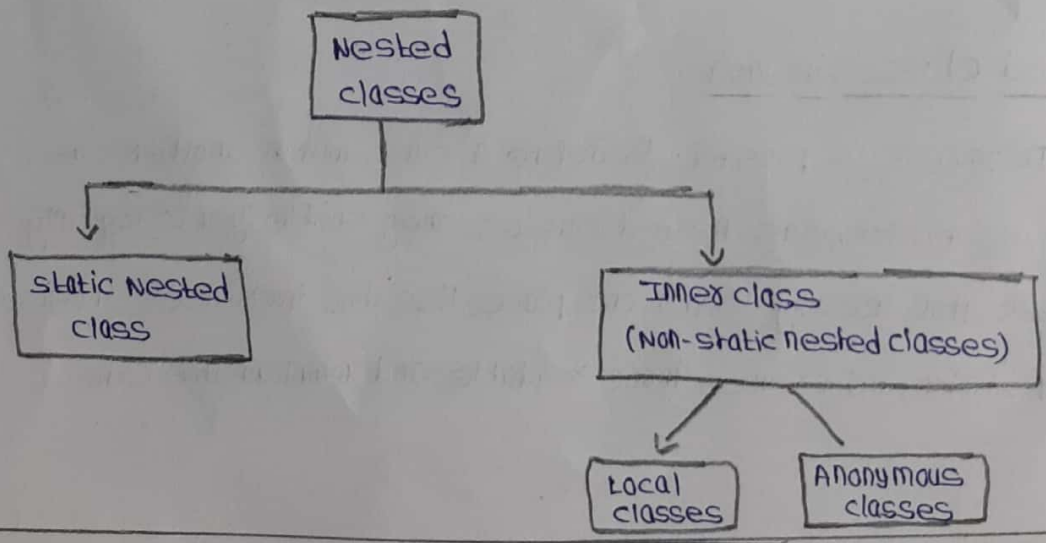
In Java, it is possible to define a class within another class, such classes are known as nested classes. They enable you to logically group classes that are only used in one place, thus this increases the use of encapsulation, and creates more readable and maintainable code.

- the scope of a nested class is bounded by the scope of its enclosing class. Thus in above example, class NestedClass does not exist independently of class OuterClass.
- 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 i.e., the enclosing class does not have access to the members of the nested class.
- 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 two categories

1. static nested class: Nested classes that are declared static are called static nested classes.
2. inner class: An inner class is a non-static nested class.

Syntax:

```
class OuterClass
{
  ...
    class NestedClass
    {
      ...
    }
}
```



Static nested classes

In the case of normal or regular inner classes, without an outer class object existing, there cannot be an inner class object. i.e., an object of the inner class is always strongly associated with an outer class object. But in the case of static nested class, without an outer class object existing, there may be a static nested class object i.e., an object of a static nested class is not strongly associated with the outer class object.

As with class methods and variables, a static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class: it can use them only through an object reference.

they are accessed using the enclosing class name

```
OuterClass.StaticNestedClass
```

For example, to create an object for the static nested class, use this syntax:

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

Example

```
// Java program to demonstrate accessing
// a static nested class
// outer 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()
        {
            // can access static member of outer class
            System.out.println("outer_x = " + outer_x);
            // can access display private static member of outer class
            System.out.println("outer_private = " + outer_private);
            // The following statement will give compilation error
            // as static nested class cannot directly access non-static member
            // System.out.println("outer_y = " + outer_y);
        }
    }
}
// Driver class
public class StaticNestedClassDemo {
    public static void main(String[] args) {
        OuterClass.StaticNestedClass nob = new OuterClass.StaticNestedClass();
        nob.display();
    }
}
```

Roll no: 19BQ1A05D7

Output:

outer_x = 10

outer_private = 30

Inner classes: To instantiate an inner class, you must first instantiate the outer class. Then create the inner object within the outer object with this syntax:

```
Outerclass.InnerClass innerObject = outerObject.InnerClass();
```

There are two special kinds of inner classes:

1. Local inner classes
2. Anonymous inner classes

Example

```
// Java program to demonstrate accessing
// a inner class
// a outer 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.println("outer_x = " + outer_x);
            System.out.println("outer_y = " + outer_y);
            System.out.println("outer_private = " + outer_private);
        }
    }
}
```

```
// Driver class
public class InnerClassDemo {
    public static void main (String[] args)
    {
        OuterClass outerObject = new OuterClass();
        OuterClass.InnerClass innerObject = outerObject.new InnerClass();
        innerObject.display();
    }
}
```

output :

outer_x = 10

outer_y = 20

outer_private = 30

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 (consu) 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 name, coach, mobile number and amount

void update() : to update the amount as per the coach selected : Extra amount (and mobile number) to be added in the amount as follows :

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.

Roll no: 19BQ1A05D7

A Program

```
import java.io.*;
import java.util.Scanner;
class RailwayTicket {
    String name,coach;
    long mobno;
    int amt,totalamt;
    public void accept() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter customer name: ");
        name = sc.next();
        System.out.println("Enter mobile number: ");
        mobno = sc.nextLong();
        System.out.println("Enter coach (First_AC/Second_AC/Third_AC/
                            sleeper): ");
        coach = sc.next();
        System.out.println("Enter basic amount of ticket: ");
        amt = sc.nextInt();
    }
    public void update() {
        if(coach.equals("First_AC")) {
            totalamt = amt + 700;
        }
        else if(coach.equals("Second_Ac")) {
            totalamt = amt + 500;
        }
        else if (coach.equals("third_Ac")) {
            totalamt = amt+250;
        }
        else {
            totalamt = amt;
        }
}
```

```
    void display(){
        System.out.println("Name: "+name);
        System.out.println("coach: "+coach);
        System.out.println("Total Amount: "+totalamt);
        System.out.println("Mobile no:"+mobno);
    }
    public static void main(String[] args){
        RailwayTicket t = new RailwayTicket();
        t.accept();
        t.update();
        t.display();
    }
}
```

Input:

Enter customer name:

Ramya

Enter mobile number:

9432652402

Enter coach(First_AC/Second_AC/third_AC/sleeper):

First_AC

Enter basic amount of ticket:

20

Output:

Name: Ramya

coach: First_AC

Total Amount: 720

Mobile no: 9432652402

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:

$V = 4/3 \times 22/7 \times r^3$

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

$V = 22/7 \times r^2 \times h$

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

$V = l \times b \times h$

A Program

```
class MethodoverloadingEx {
        double volume(double r){
                System.out.println("volume of sphere is:");
                double v =(4.0/3)*(22.0/7)*r*r*r;
                return v;
        }
        double volume(double h, double r) {
                System.out.println("volume of cylinder is:");
                double v = (22.0/7)*r*r*h;
                return v;
        }
        double volume(double l, double b, double h) {
                System.out.println("volume of cuboid is:");
                double v = l*b*h;
                return v;
        }
```

```
    public static void main(String[], args) {

        MethodoverloadingEx obj = new MethodoverloadingEx();
        double x = obj.volume(4.6);
        System.out.println(x);
        double y = obj.volume(11.5, 6.5);
        System.out.println(y);
        double z = obj.volume(5.3, 3.5, 9.5);
        System.out.println(z);
    }
}
```

Output:

volume of sphere is:

407.8841904761903

volume of cylinder is:

1527.0357142857142

volume of cuboid is:

176.225

Resources

https://www.techopedia.com/definition/25133/application-programming-interface-api-java.

https://www.programiz.com/java-programming/class-objects

https://www.google.com/amp/s/www.geeksforgeeks.org/nested-classes-java/amp/