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

A: **Role of JVM in java:**

JVM stands for "java virtual machine" which is abstract or virtual computing machine is the implementation of java virtual machine specification. It interprets the compiled java code known as the byte code and helps in program execution depending upon the specific platform.

Java is platform independent language i.e it can run on any platform without rewriting the code. This feature is supported by JVM.

Each JVM has:

- A instruction set
- A stack
- A garbage collected heap
- Method area
- A set of registers
- An execution environment.

Basically, JVM is set of computer programs and datastructures that run compiled byte code on any machine making java "compile once, run anywhere" language.

JVM is virtual, because the environment in which it runs byte code is such that it remained separated from the OS and can be removed or installed without affecting it.

JVM converts the byte code into machine or platform specific code and then runs it.

Steps in JVM implementation:

- Loads the class file.
- check whether .class file has the required byte code.
- Interprets the byte code and convert them into machine specific code.
- Removes useless objects and do garbage collection.

Source code(program.java) --- compile ---> Bytecode(program.class)

Bytecode (program.class) --- JVM ---> Machine code.

Role of API (Java Application Programming Interface)

Java application programming interface (API) is a list of all classes that are part of the Java development kit (JDK). It includes all Java packages, classes, and Interfaces, along with their methods, fields, and constructors. These prewritten classes provide a tremendous amount of functionality to a programmer.

In order to use a class from java API, one needs to include an import statement at the start of the program. For example, in order to use the Scanner class, which allows a program to accept input from the keyboard, one must include the following import statement:

```
import java.util.Scanner;
```

The above (1) import statement allows the programmer to use any method listed in the Scanner class. Another choice for including the import statement is the wildcard option shown below:

```
import java.util.*;
```

Another java package that has several commonly used classes is the java.lang package. It is automatically imported in a java program and does not need an explicit import statement. commonly used classes in the java.lang package are: Double, Float, Integer, String, StringBuffer, System, and Math.

eg: ~~MyPackageClass.java~~

Eg:
```
import java.util.Scanner;
class MyClass {
    public static void main ( String [ ] args) {
        Scanner myobj = new Scanner ( System.in);
        System.out.println(" Enter username");
        String userName = myObj.nextLine();
        System.out.println ("username is :" + userName);
    }
}
```

O/P:
```
Enter username
abcd
username is : abcd
```

(2) with an example program explain the concept of classes and nested classes in java.

A:

class :

A class is a blueprint that defines the variables and the methods common to all objects of a certain kind.

Nested classes :

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 Nested class does not exist independently of class Outerclass.
- A nested class is also a member of its enclosing class.

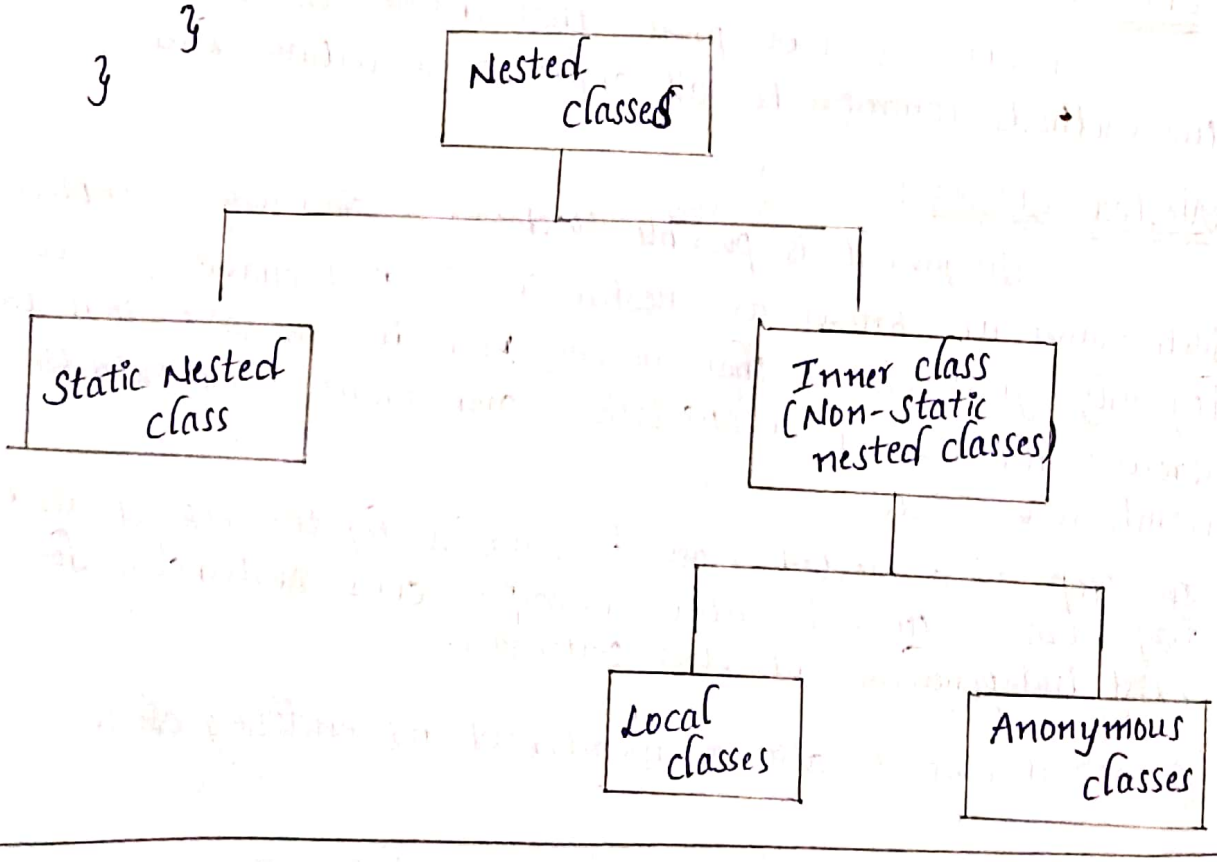
- A nested class has access to the members, including private members, of the class in which it is nested. However, the reverse is not true i.e; the enclosing class does not have access to the members of the nested class.
- As a member of its enclosing class, a nested class can be declared private, public, protected, or package private (default).
- Nested classes are divided into two categories:

1. <u>static nested class</u>: Nested classes that are declared static are called static nested classes.

2. <u>Inner class</u>: An inner class is a non-static nested class.

<u>Syntax</u>:

```
class outerClass
{
//---
    class NestedClass
    {
      //---
    }
}
```



Static Nested class:

A static class i.e created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.

Syntax:

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

Inner classes:

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();
```

Example for static Nested class & Inner class

```
public class Outerclass{
    // instance method of the outer class
    void my_Method() {
        int num = 23;
    // method_local inner class
    class MethodInner {
        public void print() {
            System.out.println("This is method innerclass "+
                                num);
        }
    }// end of inner class
    // Accessing the inner class
    MethodInner inner = new MethodInner();
    inner.print();
}
public static void main(String args[]) {
    Outerclass outer = new outerclass();
    outer.my_Method();
}
}
```

o/p: This is method innerclass 23

Static Nested class:

```
class outerclass
{
    static int outer_x = 10;
    int outer_y = 20;
    private static int outer_private = 30;
    static class StaticNestedClass
    {
        void display()
        {
            System.out.println("outer_x = " + outer_x);
            System.out.println("outer_private = " + outer_private);
        }
    }
}
// Driver class
public class StaticNestedClassDemo
{
    public static void main(String[] args)
    {
        OuterClass.StaticNestedClass nestedObject = new outerclass.
                                                    StaticNestedclass();
        nestedobject.display();
    }
}
```

Output:

```
outer_x = 10
outer_private = 30
```

Regd. No : 19BQ1A05L7

(7)

(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 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 the input for name, 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:

Type of coaches	amount
First_AC	700
Second_AC	500
Third_AC	250
Sleeper	None

void display() : To display all details of a customer suchas name, coach, total amount & mobile number.
Write a main() method to create an object of the class and call the above methods.

code:

```
import java.io.*;
import java.util.Scanner;
class RailwayTicket{
        String name, coach;
        long mobno;
        int amt, totalamt;
        void accept() throws IOException
        {
```

```
Scanner scanner = new Scanner(System.in);
System.out.print("Enter name :");
name = scanner.nextLine()
System.out.print("Enter coach :");
coach = Scanner.nextLine();
System.out.print("Enter mobno :");
mobno = scanner.nextLong();
System.out.print("Enter amt :");
amt = scanner.nextInt();
}
public void update()
    if (coach.equalsIgnoreCase("First_AC")) {
        totalamt = amt+700;
    } else if (coach.equalsIgnoreCase("Second_AC")){
        totalamt = amt+500;
    } else if (coach.equalsIgnoreCase("Third_AC")) {
        totalamt = amt+250;
    } else if (coach.equalsIgnoreCase("Sleeper")) {
        totalamt = amt;
    }
    public void display() {
        System.out.println("Name :" + name);
        System.out.println("Coach :" + coach);
        System.out.println("Mobile Number :" +mobno);
        System.out.println("Amount :" + amt);
        System.out.println("Total Amount :" + totalamt);
    }
```

Regd.NO : 19BQ1A0527

```
void main() {
        RailwayTicket railwayTicket = new RailwayTicket();
        railwayTicket.accept();
        railwayTicket.update();
        railwayTicket.display();
    }
}
```

output :

```
Enter name : Sri
Enter coach : First_AC
Enter mobno : 7732510639
Enter amt  : 3000
Name : Sri
coach  : First_AC
Mobile Number : 7732510639
Amount  : 3000
Total Amount : 3700
```

(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 * 22/7 * r * r * r

ii] double volume(double h, double r) - with height 'h' and radius 'r' as the arguements, returns the

V = 22/7 * r * r * h

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

V = l * b * h

Code :

```
//public class volume
//{
// public void volume(double r)
//{
 // double v=4.0/3 *22.0/7*r*r*r;

import java.io.*;
import java.util.*;
public class overloaded volume
{
   double volume(double r)
   {
      double v = 4/3*22/7 *r*r*r;
      return v;
   }
   double volume (double h, double r)
   {
      double v = 22/7 *r*r*h;
      return v;
   }
   double volume (double l, double b, double h)
   {
      double v = l*b*h;
      return v;
   }
   public static void main(String args[])
   {
      Scanner sc = new Scanner(System.in);
      volume ob = new volume();
      System.out.println(" volume of Sphere"+ob.volume(6.4));
      System.out.println(" volume of cylinder "+ob.volume(8.2,3.9));
      System.out.println(" volume of cuboid "+ob.volume(5.3,4.5,8.6));
   }
}
```

Regd. No : 19BQ1A0517

Output :

volume of Sphere : 786.4380000000002
volume of cylinder = 374.16599999999994
volume of cuboid = 205.10999999999999