- Define JVM, JRE and JDK and explain the ececution process of a Java Program

JVM

- JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist.
- It is a specification that provides a runtime environment in which Java byte code can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.
- JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other.
- However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance.

The JVM performs the following main tasks:

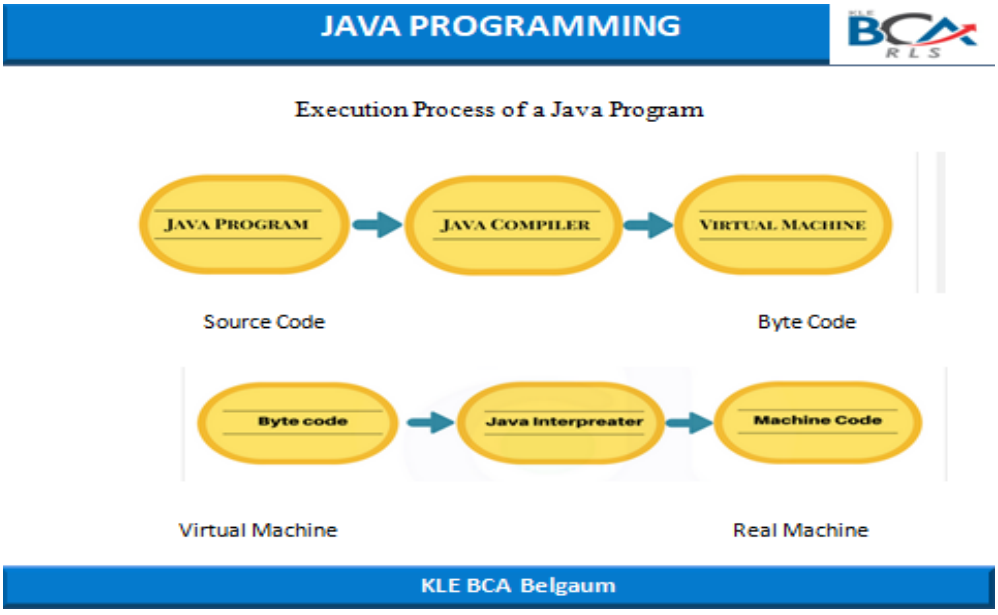
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

- JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications.
- It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.
- The implementation of JVM is also actively released by other companies besides Sun Micro Systems.

JDK

- JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.
- JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:
    - Standard Edition Java Platform
    - Enterprise Edition Java Platform
    - Micro Edition Java Platform
- The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



Java, being a platform-independent programming language, doesn't work on the one-step compilation. Instead, it involves a two-step execution, first through an OS-independent compiler; and second, in a virtual machine (JVM) which is custom-built for every operating system.

The two principal stages are explained below:

Principle 1: Compilation

First, the source '.java' file is passed through the compiler, which then encodes the source code into a machine-independent encoding, known as Bytecode. The content of each class contained in the source file is stored in a separate '.class' file.

Principle 2: Execution

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 (the class that contains the method main) is passed to the JVM and then goes through three main stages before the final machine code is executed.

These states do include:

1. ClassLoader
2. Bytecode Verifier
3. Just-In-Time Compiler

Program 1: Java Program to display student information using class and object

```
class student
{
        int USN=12345;
         String name="ABC";
        String dept="BCA";
        int sem=2;
        char div='D';
}
class pgm1
{
         public static void main(String args[])
{
        student s=new student();
        System.out.println("Student Information");
        System.out.println("USN="+s.USN);
        System.out.println("Name="+s.name);
        System.out.println("Department="+s.dept);
        System.out.println("Semester="+s.sem);
        System.out.println("division="+s.div);
}
}
```

Program 2: Write a Java program to demonstrate different data types in Java.

```
class pgm2
{
public static void main(String args[])
{
        char a = 'G';
        int i = 89;
        byte b = 4;
        short s = 56;
        double d = 4.355453532;
        float f = 4.7333434f;
        long l = 12121;
        String n="RLSI";
        System.out.println("char: " + a);
        System.out.println("integer: " + i);
        System.out.println("byte: " + b);
```

```
        System.out.println("short: " + s);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
        System.out.println("long: " + l);
        System.out.println("String: " + n);
}
}
```

Program 3: Write a Java program to demonstrate type casting.

```
public class pgm3
{
        public static void main(String[] args)
        {
                int x = 7;
                long y = x;
                float z = y;
                double p=z;
                System.out.println("Before conversion, int value "+x);
                System.out.println("After conversion, long value "+y);
                System.out.println("After conversion, float value "+z);
                System.out.println("After conversion,double value "+p);
        }

}
```

Program 4: Write a Java program to display student grade using switch statements

```
import java.util.Scanner;
public class prg4 {
   public static void main(String[] args)
{
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter grade from (A, B, C or F) : ");
        String str = scanner.next();
        char grade = str.charAt(0);
        switch (grade) {
             case 'A':
                System.out.println("Grade A - marks >=80");
```

```
                break;

            case 'B':
                System.out.println("Grade B - marks >=60");
                break;

            case 'C':
                System.out.println("Grade C - marks >=40");
                break;

            case 'F':
                System.out.println("Grade F - marks <40 FAIL");
                break;

            default :
                System.out.println("Invalid Grade");
                break;
        }
    }
}
```

Program 5: Write a Java Program to demonstrate default constructor

```
class defcon
{
        int id;
        String name;
        public void print()
        {
                System.out.println("Example of default constructor");
                System.out.println("I'D is : "+id);
                System.out.println("Name is : "+name);
        }
}
class defcon1
{
        public static void main(String args[])
        {
                defcon d=new defcon();
```

```
                d.print();
        }
}
```

Program 6: Write a Java Program to demonstrate parameterized constructor

```
class parac
{
        int id;
        String name;
        parac(int i,String n)
        {
                id=i;
                name=n;
        }
        public void display()
        {
                System.out.println("Example of parameterized constructor");
                System.out.println("I'D is : "+id);
                System.out.println("Name is : "+name);
        }
}
class parac1
{
        public static void main(String args[])
        {
                parac p=new parac(123,"Rohit");
                p.display();
        }
}
```

Program 7: Write a Java Program to demonstrate constructor overloading

```
class cover
{
        int id;
        String name;
        int age;
        cover(int i,String n)
        {
                id=i;
```

```
		name=n;
	}
	cover(int i,String n,int a)
	{
		id=i;
		name=n;
		age=a;
	}
	public void display()
	{
		System.out.println("Example of constructor overloading");
		System.out.println("I'D is : "+id);
		System.out.println("Name is : "+name);
		System.out.println("Age : "+age);
	}
}
class cover1
{
	public static void main(String args[])
	{
		cover c=new cover(123,"Rohit");
		cover c1=new cover(222,"ABC",31);
		c.display();
		c1.display();
	}
}
```

Program 8: Write a Java Program to demonstrate method overloading by changing parameter list

```
class metoverpara
{
	public void add(int a,int b)
	{
		int c;
		c=a+b;
		System.out.println("Addition of two integer numbers is"+c);
	}
	public void add(int a,int b,int c)
	{
```

```
                double d;
                d=a+b+c;
                System.out.println("Addition of three integer numbers is"+d);
        }
}
class metoverpara1
{
        public static void main(String args[])
        {
                metoverpara m=new metoverpara();
                System.out.println("Example of Method overloading by changing parameter
list");
                m.add(10,20);
                m.add(10,20,30);
        }
}
```

Program 9: Write a Java Program to demonstrate method overloading by changing data types

```
class metoverdata
{
        public void add(int a,int b)
        {
                int c;
                c=a+b;
                System.out.println("Addition of two integer data type numbers is"+c);
        }
        public void add(double a,double b)
        {
                double c;
                c=a+b;
                System.out.println("Addition of two double data type numbers is"+c);
        }
}
class metoverdata1
{
        public static void main(String args[])
        {
```

```
        metoverdata m=new metoverdata();
        System.out.println("Example of Mrthod overloading by changing data types");
        m.add(10,20);
        m.add(10.5,20.5);
    }
}
```

Program 10. Write a Java Program to demonstrate the concept of method overriding.

```
    class metride
    {
        public void print()
        {
            System.out.println("From base class");
        }
    }
    class metride1 extends metride
    {
        public void print()
        {
            System.out.println("From derived class");
        }
        public static void main(String args[])
        {
            metride1 m=new metride1();
            System.out.println("Example of Method Overriding");
            m.print();
        }
    }
```