```
// Program to explain class and objects.
class Demo
{
	int a;

	class InnerClass
	{
		int a;
		void innerdisplay()
		{
			System.out.println("Inside inner class.");
		}
	}

	void display()
	{
		System.out.println("Inside display method of Outer class.");
		InnerClass in = new InnerClass();
		in.innerdisplay();
	}

}

class Test
{
	public static void main(String args[])
	{
		System.out.println("Inside main method of class Test.");
		Demo d = new Demo();
		d.display();
```

```
	}

}

// Program to explain constructors.
class Demo
{
	int a;

	Demo()
	{
		System.out.println("Inside Parameterless constructor.");
	}
	Demo(int a)
	{
		this.a = a;
		System.out.println("Inside Parameterised constructor.");
	}
	Demo(Demo d2)
	{
		System.out.printf("%d", d2.a);
	}
}

class Test
{
	public static void main(String args[])
	{
		Demo d = new Demo();
		Demo d2 = new Demo(10);
		Demo d3 = new Demo(d2);
```

```
        }
}
```

```
/*

Que : Write a java program to illustrate final in java. (Final - field, method, local variable, outer class)
Owner: Rushikesh Sanjay Pokharkar
Batch: PPA9

*/

//                  ********** Solution ***********

import java.util.Scanner;

final class College
{
        final static String college_code, college_name = "AVCOE"; // created the final static fields.

        static // Created the static block to initialize the static field.
        {
                college_code = "deij54965"; // Initialize the final static field in static block.
        }

        final void books() // created the non-static final method.
        {
                final int a;
                a = 10;
```

```
		System.out.println("All books of college are avilable here.");
		System.out.println("The value of final local variable is: "+a);
	}

	static void labs() // created the static method.
	{
		System.out.println("All Labs Information is in this block.");
	}

}

class Students
{
	int rollNo, id;
	String name, div;
	final String StudentUnion;

	{ // Created the non-static block to assign the values to the final fields.
		StudentUnion = "Student_Union_Name";
	}

	Students(int rollNo, int id, String name, String div) // created the constructor to initialize the non-static fields.
	{
		this.rollNo = rollNo;
		this.id = id;
		this.name = name;
		this.div = div;
	}

	void print_details() // method to print the details of students including final static and non-static fields.
```

```
	{
		System.out.printf("College Name of Student: %s\n", College.college_name); // Access the static field using class name.
		System.out.printf("College code: %s\n", College.college_code); // Access the static field using class name.
		System.out.printf("Name of student: %s\n", name);
		System.out.printf("Division of student: %s\n", div);
		System.out.printf("Id of student: %d\n", id);
		System.out.printf("Roll no of student: %d\n", rollNo);
		System.out.printf("Student Union: %s\n", StudentUnion);
	}

	public static void main(String args[])
	{

		Scanner sc = new Scanner(System.in); // Created the scanner class object.

		System.out.print("Enter Name of Student1: ");
		String name1 = sc.nextLine();
		System.out.print("Enter Div: ");
		String div1 = sc.nextLine();
		System.out.print("Enter rollNo of Student: ");
		int rollNo1 = sc.nextInt();
		System.out.print("Enter id of Student: ");
		int id1 = sc.nextInt();
		Students s1 = new Students(rollNo1, id1, name1, div1); // created the first object of student class.
		s1.print_details();
		College c = new College();
		c.books(); // Accessed the non-static method by creating the object of the class.
```

```
                System.out.print("Enter Name of Student2: ");
                sc.nextLine();
                String name2 = sc.nextLine();
                System.out.print("Enter Div: ");
                String div2 = sc.nextLine();
                System.out.print("Enter rollNo of Student: ");
                int rollNo2 = sc.nextInt();
                System.out.print("Enter id of Student: ");
                int id2 = sc.nextInt();
                Students s2 = new Students(rollNo2, id2, name2, div2); // created the second object
of the class.
                s2.print_details();
                College.labs(); // Accessed the static method usint the class name

        }

}



/*

Que : Write a java program to illustrate static in java. (Static - block, field, method)
Owner: Rushikesh Sanjay Pokharkar
Batch: PPA9

*/

//                  ********** Solution **********

import java.util.Scanner; // Import necessary classes.
```

```
class College
{
	static String college_code, college_name = "AVCOE"; // created the static fields.

	static // Created the static block to initialize the static field.
	{
		college_code = "deij54965"; // Initialize the static field in static block.
	}

	void books() // created the non-static method.
	{
		System.out.println("All books of college are avilable here.");
	}

	static void labs() // created the static method.
	{
		System.out.println("All Labs Information is in this block.");
	}

}

class Students
{
	int rollNo, id;
	String name, div;

	Students(int rollNo, int id, String name, String div) // created the constructor to initialize the non-static fields.
	{
		this.rollNo = rollNo;
```

```
        this.id = id;
        this.name = name;
        this.div = div;
    }

    void print_details() // method to print the details of students including static and non-static fields.
    {
        System.out.printf("College Name of Student: %s\n", College.college_name); // Access the static field using class name.
        System.out.printf("College code: %s\n", College.college_code); // Access the static field using class name.
        System.out.printf("Name of student: %s\n", name);
        System.out.printf("Division of student: %s\n", div);
        System.out.printf("Id of student: %d\n", id);
        System.out.printf("Roll no of student: %d\n", rollNo);
    }

    public static void main(String args[])
    {

        Scanner sc = new Scanner(System.in); // Created the scanner class object.

        System.out.print("Enter Name of Student1: ");
        String name1 = sc.nextLine();
        System.out.print("Enter Div: ");
        String div1 = sc.nextLine();
        System.out.print("Enter rollNo of Student: ");
        int rollNo1 = sc.nextInt();
        System.out.print("Enter id of Student: ");
        int id1 = sc.nextInt();
```

```
		Students s1 = new Students(rollNo1, id1, name1, div1); // created the first object of student class.
		s1.print_details();
		College c = new College();
		c.books(); // Accessed the non-static method by creating the object of the class.



		System.out.print("Enter Name of Student2: ");
		sc.nextLine();
		String name2 = sc.nextLine();
		System.out.print("Enter Div: ");
		String div2 = sc.nextLine();
		System.out.print("Enter rollNo of Student: ");
		int rollNo2 = sc.nextInt();
		System.out.print("Enter id of Student: ");
		int id2 = sc.nextInt();
		Students s2 = new Students(rollNo2, id2, name2, div2); // created the second object of the class.
		s2.print_details();
		College.labs(); // Accessed the static method usint the class name

	}

}




// Program to explain access specifiers.
class Demo
{
	protected int a = 10;
	protected int b = 20;
```

```
	public int c = 30;
	public int d = 40;
	private int e = 50;
	private int f = 60;

	void display()
	{
		System.out.println("Protected a = "+ a);
		System.out.println("Protected b = "+ b);
		System.out.println("public c = "+ c);
		System.out.println("public d = "+ d);
		System.out.println("private e = "+ e);
		System.out.println("private f = "+ f);
	}
}

class Demo2 extends Demo
{
	void display()
	{
		System.out.println("\n");
		System.out.println("Protected a = "+ a);
		System.out.println("Protected b = "+ b);
		System.out.println("public c = "+ c);
		System.out.println("public d = "+ d);
	}
}

class Test
{
```

```
	public static void main(String args[])
	{
		Demo d = new Demo();
		d.display();

		Demo2 d2 = new Demo2();
		d2.display();
	}
}



// Program of this and super keyword for constructors.
class Parent
{
	Parent()
	{
		System.out.println("Parent Parameterless Constructor.");
	}

	Parent(int a)
	{
		this();
		System.out.println("Parent Parameterised Constructor.");
	}
}
class Child extends Parent
{
	Child()
	{
		super(20);
```

```
		System.out.println("Child Parameterless Constructor.");
	}

	Child(int a)
	{
		this();
		System.out.println("Child Parameterised Constructor.");
	}
}

class Test
{
	public static void main(String args[])
	{
		Child C = new Child(10);
	}
}



// Program of this and super keywords for fields.
class College
{
	String college_name;
	String college_id;
	String address;

	College(String college_id, String college_name, String address)
	{
		this();
		this.college_id = college_id;
```

```
		this.college_name = college_name;
		this.address = address;
	}
	College()
	{
		System.out.println("Inside College.");
	}
}

class Student extends College
{
	String stud_name;
	int stud_rollno;
	String address;

	Student(String stud_name, int stud_rollno, String address)
	{
		super("011", "AVCOE", "Ghulewadi, Sangamner.");
		this.stud_name = stud_name;
		this.stud_rollno = stud_rollno;
		this.address = address;
	}

	Student()
	{
		System.out.println("Inside Student.");
	}

	void printDetails()
	{
		System.out.println(this.stud_name);
```

```
                System.out.println(stud_rollno);
                System.out.println(this.address);
                System.out.println(super.college_name);
                System.out.println(this.college_id);
                System.out.println(super.address);
        }

        public static void main(String args[])
        {
                Student S = new Student("Rushikesh", 3227, "S.P.");
                S.printDetails();
        }
}



// Program of this and super keyword for methods.
class Parent
{
        void fun()
        {

                System.out.println("Parent fun method.");
        }

        void gun()
        {
                System.out.println("Parent gun method.");
        }
}
```

```
class Child extends Parent
{
	void fun()
	{
		System.out.println("Child fun method.");
	}

	void gun()
	{
		this.fun();
		super.fun();
		super.gun();
		System.out.println("child gun method.");
	}

	public static void main(String args[])
	{
		Child C = new Child();
		C.gun();
	}
}



// Program to explain encapsulation.
class Student
{
	private String name;
	private String address;
	private int rollno;

	void setname(String name)
```

```
	{
		this.name = name;
	}
	void setaddress(String address)
	{
		this.address = address;
	}
	void setrollno(int rollno)
	{
		this.rollno = rollno;
	}

	String getname()
	{
		return this.name;
	}
	String getaddress()
	{
		return this.address;
	}
	int getrollno()
	{
		return this.rollno;
	}
}

class Test
{
	public static void main(String args[])
	{
		Student s = new Student();
```

```
                s.setname("Rushikesh");
                s.setaddress("Sarole Pathar.");
                s.setrollno(63);

                String name = s.getname();
                String address = s.getaddress();
                int rollno = s.getrollno();

                System.out.println("Name = "+ name);
                System.out.println("Address = "+ address);
                System.out.println("Roll No = "+ rollno);
        }
}



// Program to explain composition.
class college
{
        int collegeid;
        String collegename;

        college(int collegeid, String collegename)
        {
                this.collegeid = collegeid;
                this.collegename = collegename;
        }

        void Display()
        {
                System.out.println("Inside Display method of college class.");
```

```
                System.out.println("College Id = "+ collegeid);
                System.out.println("College Name = "+ collegename);
        }
}

class Student
{
        college c = new college(111, "AVCOE");

        public static void main(String args[])
        {
                Student s = new Student();
                s.c.Display();
        }
}
```

12. List out and describe all java features on paper.

→

Features of Java.

1. Platform Independent
2. Architecture Independent
3. Portable
4. Simple
5. Object oriented
6. Multithreaded
7. Secure
8. High performance.
9. Distributed.
10. Interpreted
11. Robust
12. Reliable
13. Dynamic

1] Platform Independent:

Java program is software independent i.e. O.S. independent so it is called platform independent.

2] Architecture Independent:

Java program is run on different divices. If it get compiled on one device and compiled bytecode file is if run on another device then it get run successfully hence Java is Architecture Independent.

3] Portable:
If the output on different divices are same after compilation then it is called portable & Java also provides this feature.

4] Simple:
As it required less
1. time
2. memory
3. power consumption hence Java is simple.

5] Object oriented:
Java supportes all the feature required for object orientation hence it is object oriented And Java is pure object oriented language.

6] Multithreaded:
When one task is started after that anoth some task get started this process is multithreading and Java provides this feature.

7] Dynamic:
As all the memory allocation is decided on Run time and there is no static memory allocation in it hence Java is Dynamic language.

11. Prepare JVM architecture ?

→

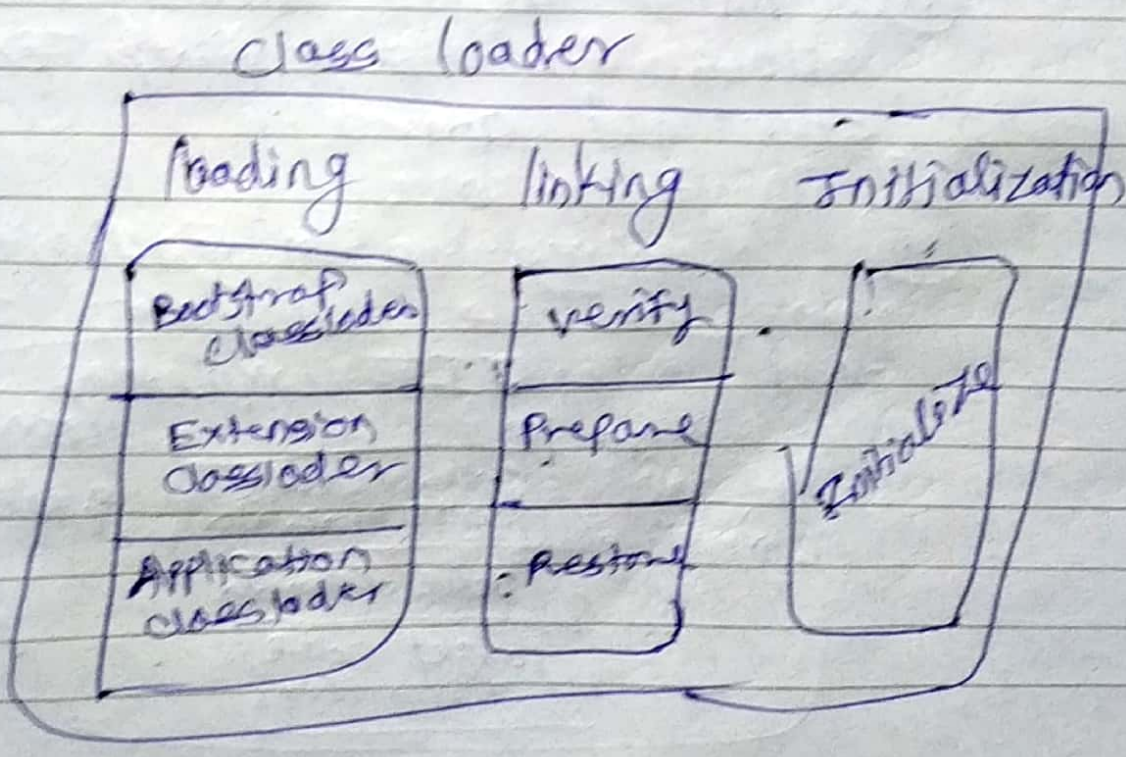
(i) When we compile the Java file by using Javac file.java command we get the .class file.

(ii) To run the .class file we required the java tool which is stored in the same folder where Javac is present.

(iii) When it get Run, then the class loader present in JVM goes towards your compiled file and get the .class file

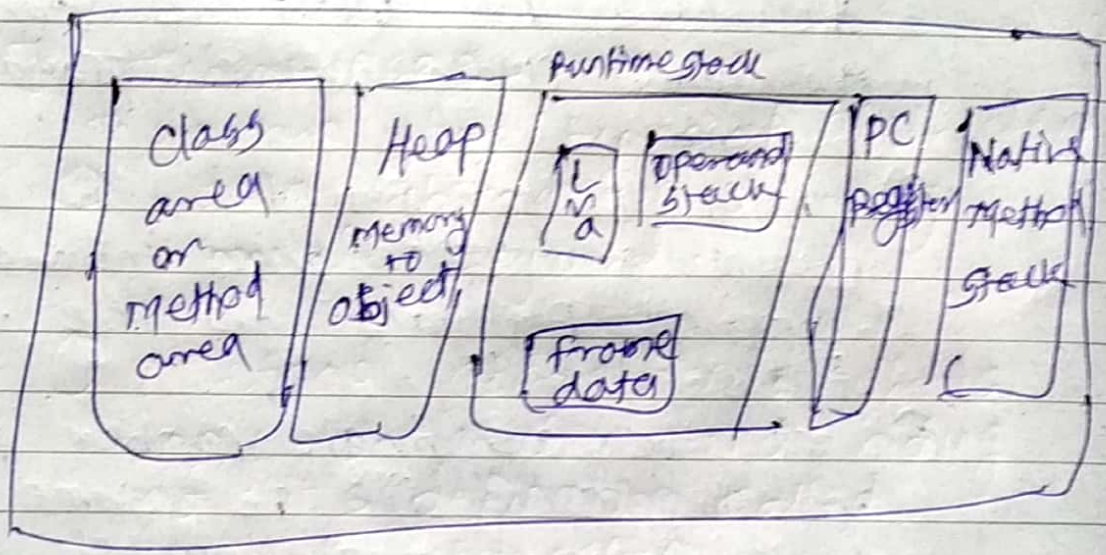
(iv) If the O.S. magic number present in the class file is same as of Java O.S. magic number, Cafebabe then the class gets loaded in the ~~Class tool~~ loader.

(v)



(iv) After all class loading memory allocation is done.
This memory is memory allocated by JVM.

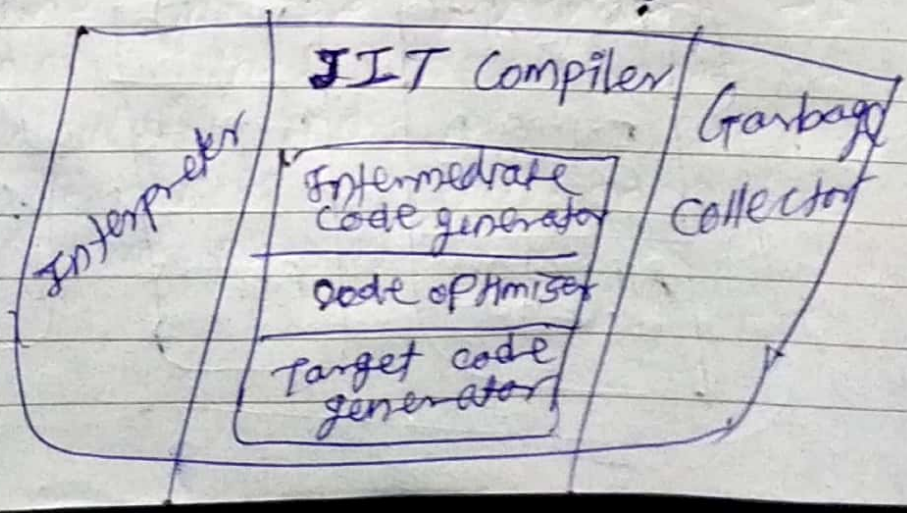
Memory Allocated by JVM



(v) In all above blocks the class get stored.

(vi) After geting memory to the class the next step execution step get arised.

(vii) Execution Engin



13. List out and write a short note on object oriented pillars.

→ There are 7 Pillars of object oriented programming.

1. class
2. object
3. Encapsulation
4. Abstraction
5. Inheritance
6. Polymorphism
7. Message passing

1] class :

class is collection of data members / fields and member functions / methods.

It is a collection of properties and behaviours.

2] object :

creation of physical existance in virtual class is called object.

The one person with name is an object of class human.

3] Encapsulation :

Binding of data members / fields and member functions / methods called Encapsulation.

4] Abstraction:

Hiding of unnecessary things and unhiding of necessary thing called Abstraction.

The os main function is Hardware abstraction.

5] ~~Inheritance:~~

There are two types of abstraction

① data abstraction : using private keyword.

② method abstraction : shows only declaration and hiding of it's defination called method abstraction.

5] Inheritance:

Inheritance means accessing parent class properties into the child class called Inheritance.

6] Polymorphism:

polymorphism means single name multiple behaviour. If one method has a name of calculate and it's calculates different function. Then it's polymorphism.