Name: Bimal Wocl
Rollno: 07
: Advance java

---

1. Explain Java architecture in detail.

Ans

- Java Architecture is a collection of components i.e. JVM, JRE, & JDK. It integrates the process of interpretation & compilation.

- It defines all the processes involved in creating a java program. Java Architecture explains each & every step how a program is compiled & executed.

- Java Architecture can be explained by using following steps.

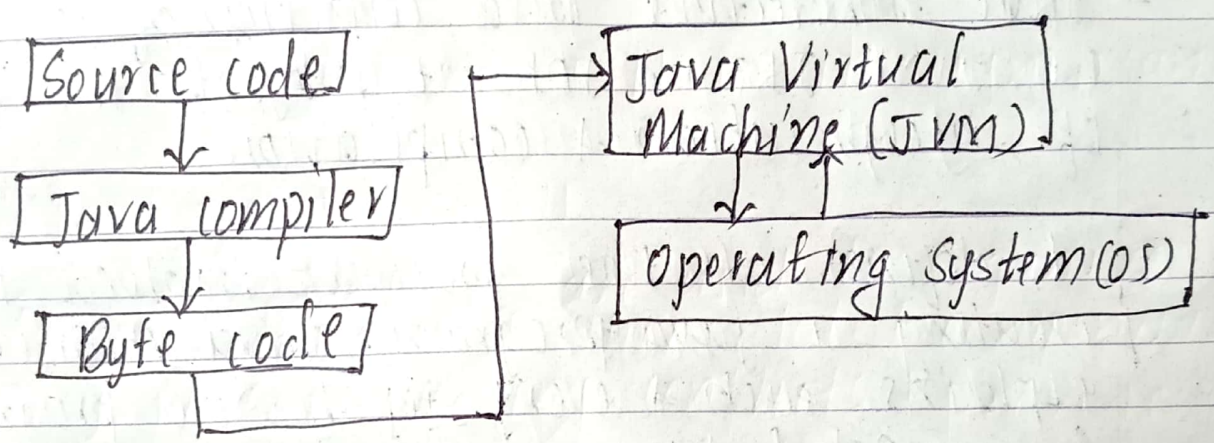
(i) There is a process of compilation & interpretation in java.

(ii) Java compiler converts the java code into ~~Java~~ byte code.

(iii) After that, the JVM converts the byte code into machine code.

(iv) The machine code is then executed by the machine

The following figure represents the java architecture in which each step is elaborate graphically.



Component of java

(i) Java Virtual Machine (JVM)
(ii) Java Runtime Environment (JRE)
(iii) Java Development Kit (JDK)

(i) Java Virtual Machine (JVM)

- The main feature of JVM is WORA. It stands, write Once Run Anywhere.

- In a java, if you are writting some code then run anywhere or on any OS.

- JVM main tasks is convert ~~Java~~ byte code into machine code.

(ii) Java Runtime Enviroment (JRE)
- It provides an enviroment in which java programs are executed.

- JRE takes our java code, integrates it with the required libraries & then starts the JVM to execute it.

(iii) Java Development Kit (JDK)
- It is a software development enviroment used in the development of java application & applets.

- JDK holds JRE, a compiler, an interpreter or loader, and several development tools in it

2. What is Classpath? Explain its significan

Ans:

Classpath is a parameter in the jvm or the java compiler that specifies the location of user-defined classes & packages.

- The parameter may be set either on the command-line or through an environment variable.

Significance of java classpath

Classpath
- In java, the classpath is an essential concept that helps the java virtual Machine (JVM) locate classes & resources during program execution.

Let me explain its significance:

(1) Package & imports
- In java, packages are used to organize classes, interfaces & other related components. They prevent naming conflicts & provide controlled access.
- When you use an import statement (e.g. import org.company.Menu;), you are making a class available in your current class.

(2) Classpath Enviromental variable
- The classpath enviroment variable specifies the locations where the JVM should look for classes & resource
- It can be include directories & JAR files.

(3) Setting the classpath:
- You can set the classpath in different ways:
  - Command prompt.
  - Enviromental variable
  - IDE

(3) Explain how java achieves platform independence.

Ans.

Java achieves platform independence through the following mechanisms.

(i) Bytecode compilation
- When you write java code, it is first compiled into an intermediate form called bytecode.
- Bytecode is a low-level representation of your java program i.e. independent of the underlying hardware & OS.

(ii) Java Virtual Machine (JVM).
- The JVM is responsible for executing java bytecode.
- It acts as an interpreter for bytecode, translating it into native machine code at runtime

(iii) Write once Run anywhere

(iv) Class Libraries (APIs)

(v) Security & Sandboxing.

(vi) Just-In-Time (JIT) compilation

(4) What is package in java? Also, explain all available access modifier with examples.

Ans

Package in Java

Package is the collection of related classes, interfaces & sub packages.

It is two types.

(a) Built-in package
(b) User-defined package.

Access specifier ~~(specifie~~ (Scope Modifier)

These are keyword which specify the scope of the given class, methods & variables.

(i) Private
(ii) default
(iii) Protected
(iv) Public

	Same class	Same Package	Different Package with Subclass	Different Package without Subclass
Private	✓	✗	✗	✗
default ~~Public~~	✓	✓	✗	✗
Protected	✓	✓	✓	✗
Public	✓	✓	✓	✓

⑤ Explain the use of this in java.

Ans

'this' is a keyboard which is identify the instance & local variable.

Use of this in java.

- It is used to separate the instance & local variable.
- It is used to called the constructor if the multiple constructor present in program.
- Returning current class object.
  (In some cases, methods return the current object (i.e. this) to allow method chaining)
- Passing 'this' as an argument
- Passing "this" in constructor calls

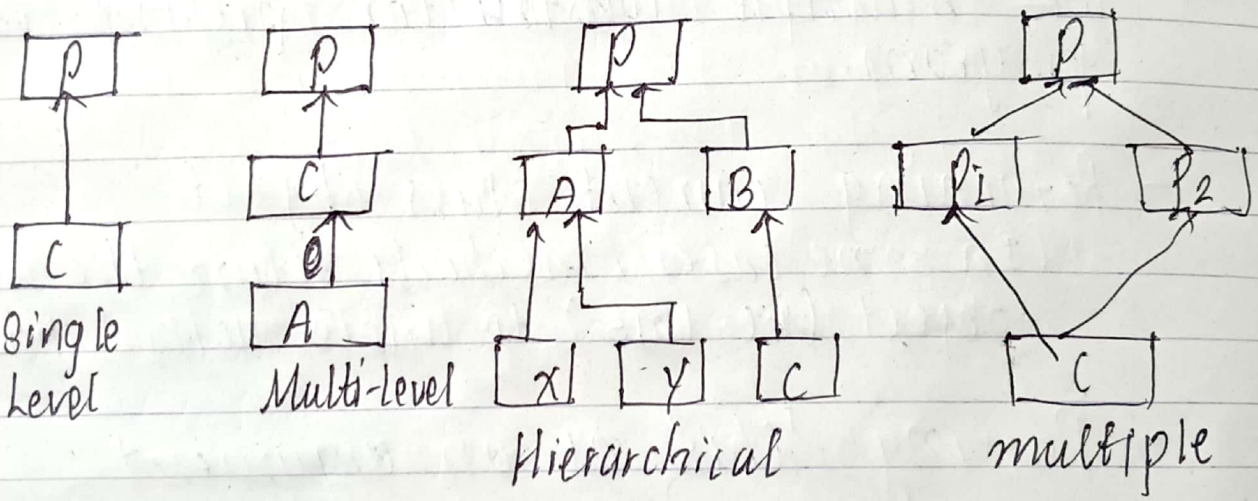
⑥ Define ~~inheritance~~ inheritance. Explain different types of inheritance.

Ans

Inheritance.

Inheritance is a feature of OOP used for reuseability.

- 'extends' keyword is used.
- Inheritance can be used when classes share 'IS-A' relationship.



- In java, ~~multiple~~ inheritance is not allowed using classes.

- Different classes may have methods for same name but with different implementation. When a class inherits/extends from those classes, ~~java~~ JVM cannot decide which implementation to be implemented.

- To avoid such ambiguity, java doesn't allow multiple inheritance.
- We can achieve multiple inheritance in java using interfaces.

1. Single level.

```
class P
{

}
class C extends P.
{

}
```

2. Multilevel

```
class P
{

}
class C extends P
{

}
class A extends C
{

}
```

3. Hierarchical

```
class P
{
}
class A extends P
{
}
class B extends P
{
}
class x extends A
{
}
class y extends A
{
}
class C extends B
```

(7) Why multiple inheritance is not allowed in java ? How can we use multilevel inheritance ?

Ans

By restricting multiple inheritance & promoting interfaces & composition java aims to maintain simplicity, clarity, robustness in its object oriented design, making it easier

for developers to write, maintain, & understand code.

Ⓜsample program of multilevel inheritance

```
class A
{
    String name;
    int roll;
    public A(String name, int roll)
    {
        this.name = name;
        this.roll = roll;
    }
}
Class B extends A
{
    int no_sub;
    public B(String name, int roll
            , int no_sub) {
        super(name, roll);
        this.no_sub = no_sub;
    }
class C extends B
{
    int marks;
    public C(String name, int roll
```

```
        int no-sub, int marks)
    {
        super(name, roll, no-sub);
        this.marks = marks;
    }
}

class Test {
    public static void main(String[] args)
    {

        public String getInfo() {
            String Info = "name:" + this.name + "Roll:"
                + this.roll + "Number of subject:"
                + this.no-sub + "Marks:" + marks;
            return Info;
        }
    }
class Test {
    public static void main(String[] args)
    {
        C c = new C("Bikash", 05, 6, 95);
        System.out.println(c.getInfo());
    }
}
```

⑧ Write a program to override toString ()

Ans

```
Class Student {
    int roll;
    String name;
    public student (int roll, string
                    name)
    {
        this.name = name;
        this.roll = roll;
    }
    Public String toString ( ) {
        String Info = "Roll:" + this.roll +
                      "Name:" + this.name;
        return Info;
    }
}
Class Test {
    public static void main (String [] args)
    {
        Student s = new student ("Bikash", 05)
        System.out.println (s);
    }
}
```

⑨ Differentiate between interface & abstract class with a simple program.

Ans

Interface	Abstract Class
- All methods are abstract by default. Since java add we can have default static methods	- It can have both abstract & non-abstract methods.
- It doesn't have a constructor	- It can have a constructor
- Members of interface must be final & static	- Variables can be final, non-final, static & non-static.
- All member of interface are public by default.	- member of abstract class can have many access modifier.
- Interface can be extended by another interface or implemented by another class	- It can be implemented by interface.
- It provides abstraction level of 100%.	- It can provide abstraction level from 0 to 100%.

→ Sample program:

```
Interface A {
    void print();
}
Interface B {
    void print();
}
class x implements A, B
{
    @Override
    public void print()
    {
        System.out.println("hi");
    }
}
```

→ Sample program

```
abstract class Payment {
{
    abstract public void auth();
    public void makepayment()
    {
        balance_pri'te_'
    }
}
```

(10) Design a class student with name, faculty & address. The address consists of the street name, ward number & province. Also write a method to display the details of the student. Use composition.

Ans:

Composition

Composition is used when two or more classes are related in such a way that an object of one class cannot be created without the other.

It can be used when classes share 'Has-A' relationship.

```
Class Address {
    String st, province;
    int wardn;
    public Address (String st, String
        province, int ward)
    {
        this.st = st;
        this.province = province;
        this.ward = ward;
    }
@Override
    public String toString()
    {
        return "Street = " + this.st +
        "province = " + this.province +
        "ward = " + this.ward;
    }
}
Class Student {
    String name;
    int roll;
    Address addr;
    public Student (String name, int
        roll, Address addr)
    {
        this.name = name;
        this.roll = roll;
        this.Address = addr;
    }
```

```
@Override
    public String toString()
    {
        return "name:" + this.name + "Roll:"
                + this.roll + this.addr;
    }
}

Class Test {

    Public static void main (String IJ args)
    {
        Address addr = new Address ("Barkumari",
                "Bagmati", 09);
        Student s = new Student ("Alex", 2,
                addr);
        Student st = new Student ("Bob", 3, addr)
        System.out.println (s);
    }
}
```

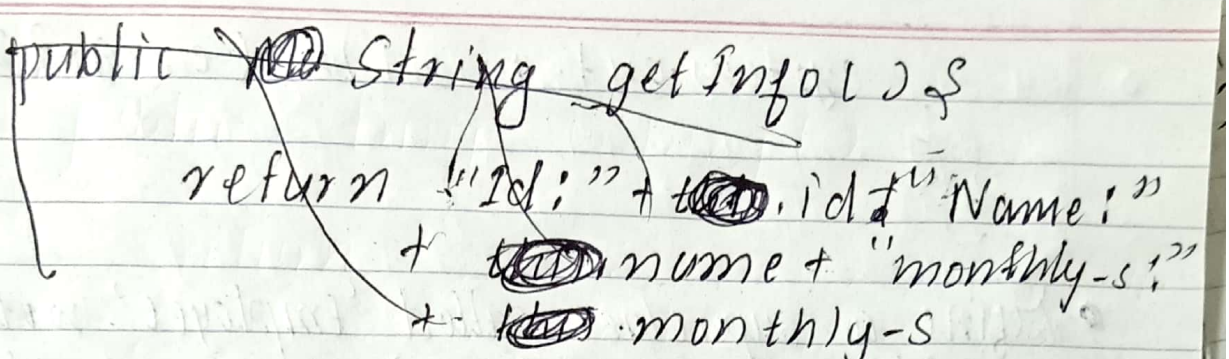
11 Consider the following scenario:

- You are tasked with designing a system to manage employees in a company. There are different types of employees, such as regular employees & managers. All employees have a name, an employee ID, & a basic monthly salary.

Ans

```
Class Emp {
    String name;
    int id, monthly-s;

    Public Emp (String name,
    Public Emp ( int id, String name,
                 int monthly-s )
    {
        this.id = id
        this.name = name;
        this.monthly-s = s;
    } Public String getInfo () {
        return "Id:"
Class RegEmp extends Emp.
{
    public RegEmp (int id, String name,
                   int monthly-s)
    {
        Super (id, name, monthly-s);
    }
```




```
Public String getInfo()
{
    return "Id:" + id + "Name:" + name +
           "monthly-s:" + monthly-s;
}
}

class Manager extends {
    public Manager(int id, String
                   name, int monthly-s)
    { super(id, name, monthly-s);
    }
    public string getInfo()
    {
        return "Id:" + id + "Name:" + name
               + "monthly-s" + monthly-s;
    }
}
class Test {
    public static void main (String[] args)
    { RegEmp e1 = new RegEmp(1, "abc", 1200);
      Manager e2 = new Manager(2, "c", 1000);
System.out.println(e1.GetInfo());
```

```
        System.out.println(ex.getInfo());
    }
}
```

- Define a superclass called 'Employee' with instance variable for name (String), employee ID (int), & basic salary (double). Include a constructor to initialize these variables & methods to get & set each variable.

Ans

```
Class Employee
{
    String name;
    int id;
    double salary;
    public Employee(String name, int id,
                    double salary)
    {
        this.name = name;
        this.id = id;
        this.salary = salary;
    }
    public String getInfo()
    {
        return "Name:" + this.name + "Id:" +
               this.id + "Salary:" + this.salary;
    }
```

```
public static void main (String [] args)
{
    Employee e = new Employee ("Bikash", 05, 1000);
    System.out.println (e.getInfo());
}
}
```

- Create a subclass called "Manager" that extends 'Employee'. Managers have an additional instance variable for their department (String). ~~methods~~ Include a constructor to initialize all variables, including the department & methods to get & set the department.

Ans

```
Class Employee
{
    String name;
    int id;
    double salary;
    public Employee (String name, int id,
                     double salary)
    {
        this.name = name;
        this.id = id;
        this.salary = salary;
    }
    public String getInfo ()
    {
```

```
        return "ID: " + this.id + "Name: " + this.name
                + " salary: " + this.salary;
    }
}

Class Manager extends Employee
{
    String dept;

    public Manager (String name, int id,
                    double salary, String dept)
    {
        super (name, id, salary);
        this.dept = dept;
    }
    @Override
    public String getInfo ()
    {
        return super.getInfo + "Department: " +
                this.dept ;
    }
}
class Test {
    public static void main (String [] args)
    {
        Employee e = new Employee ("Arun", 02, 1200);
        System.out.println (e.getInfo());
        Manager m = new Manager ("Bikash", 05, 1200, CE);
        System.out.println (m.getInfo ());
    }
}
```

- Another subclass called 'Regular Employee' should also extend 'Employee'. Regular employee do not have any additional instance variable. Implement the necessary constructor & methods for this subclass.

Ans

```
Class Employee
{
    String name;
    int id;
    double salary;

    public Employee (String name, int id,
                     double salary)
    {
        this.name = name;
        this.id = id;
        this.salary = salary;
    }
    public String getInfo()
    {
        return "ID:" + this.id + "Name:" + this.name
               + "Salary:" + this.salary;
    }
}
class RegularEmployee extends Employee
{
    public RegularEmployee(String name, int id,
                           double salary)
```

```
    {
        super (name, id, salary);
    }
@ override
    public String getInfo()
    {
        return super.getInfo();
    }
}
Class Test
{ public static void main (String[] args){
    Employee e = new Employee ("Arun", 04, 1200);
    System.out.println (e.getInfo());
    RegularEmployee r = new RegularEmployee
                        ("Bikash", 05, 1000);
    System.out.println (r.getInfo());
  }
}
```

- Write a java program that demonstrate the use of these classes. Instantiate objects for one manager & two Regular employees, & display their information including name, ID, salary & department (if applicable).

```
class Employee
{
    String name;
    int id;
    double salary;

    public Employee (String name, int id,
                     double salary)
    {
        this.name = name;
        this.id = id;
        this.salary = salary;
    }
    public String getInfo()
    {
        return "Name:" + this.name + "ID:"
               + this.id + this: "Monthly salary:"
               + thisSalary;
    }
}
class Manager extends Employee
{
    string dept;
    public Manager (String name, int id, double
                    salary, String dept;
    {
        super (name, id, salary);
        this.dept = dept;
    }
```

@override

```
public String getInfo()
{
    return super.getInfo() + "Department:" + this.dept;
}
}
class RegularEmployee extends Employee
{
    public RegularEmployee(String name, int id
                           , double salary)
    {
        super(name, id, salary);
    }
@Override
    public String getInfo()
    {
        return super.getInfo();
    }
}
class Test
{
    public static void main(String[] args)
    {
        Manager m = new Manager("Alex", 05, 1500);
        System.out.println(m.getInfo());
        RegularEmployee r = new RegularEmployee("Bob", 06, 1200);
        System.out.println(r.getInfo());
        RegularEmployee r1 = new RegularEmployee("Bikash",
                                                 07, 1200);
        System.out.println(r1.getInfo());
    }
}
```

- Demonstrate how method overriding can be utilized within this inheritance hierarchy for example by creating a method 'calculateSalary()' in both the 'Manager' & 'RegularEmployee' classes to calculate the monthly salary after including any additional bonuses or benifits specific to each type of employee.

Ans

```
Class Employee
{
    String name;
    int id;
    double msalary;

    public Employee (String name, int id,
                     double msalary)
    {
        this.name = name
        this.id = id;
        this.msalary = msalary;
    }
    public String getInfo ()
    {
        return "Name:" + this.name + "ID:" +
               this.id + "monthly-salary:" +
               this.msalary;
    } public int calculateSalary ()
    {
        return msalary;
    }
}
```

```
Class Manager extends Employee
{
    String dept;
    int bonuse.
    public Manager(string name, int id,
            double msalary, String dept, int bonuse)
    {
        this.name = name;
        this.id = id;
        super (name, id, msalary);
        this.dept = dept;
        this.bonuse = bonuse;
    }
    @Override
    public int calculateSalary()
    {
        return super.caducalateSalary
            + bonuses * 10;
    }
@Override
    public String getInfo()
    {
        return super.getInfo + "Department:"
            + this.dept + "Bonuses:" + this.bonuses;
    }
}
```

```
Class RegEmpD extends Employee
{
    int bonuse;
    public RegEmp (String name, int id,
                   double msalary, int bonuse);
    {
        super (name, id, msalary);
        this.bonuse = bonuse;
    }
    public
@Override
    public int calculateSalary ()
    {
        return super.calculateSalary + bonuse;
    }
@Override
    public String getInfo ()
    {
        return super.getInfo + "Bonues fo:
        Regular employee" + this.Bonuse;
    }
}
Class Test {
    Employee e = new Employee ("A", 1, 100);
    Manager m = new Manager ("abc", 2, 100, 20);
    RegEmp r = newManager ("Alex", 3, 1000, 10);
    System.out.println (e.getInfo ());
    System.out.println (m.getInfo ());
    System.out.println (r.getInfo ());
}
```

Explain the concept of inheritance & how it helps in code reusability & Organization within context of this scenario.

Inheritance.

It helps to use the code again & again in a program, so it provides reusability, increase efficiency of the program.

- Discuss the benefits of using inheritance over other technique for code organization & management in java application.

Ans

benefits of using inheritance over other technique in java are.-

(i) Code Reusability

Inheritance in java allows for code reuse, saving you ample time & effort during software development.

(ii) Facilitates Polymorphism

Inheritance in java derived class object can call/invoke base class so one class can have call multiple times hence the polymorphism

(ii) Facilitates polymerphism

(iii) Class Hierarchy.
Inheritance in java allows class hierarchy.

(iv) Code Maintainability

(v) Code Modularity.

(vi) Abstraction.

(vii) Code Flexibility.