OOP’S

1. What is Object Oriented Programming Language and Non-Object Oriented Programming ?

   ▼ Ans

2. Is Java fully Object Oriented Programming Language ?

   ▼ Ans

   - No, Java is not fully or 100% Object Oriented Programming Language, because it uses the primitive datatypes and static members.
   - In Java Primitive datatypes are not an object, so that is why Java is not 100% or fully Object Oriented Programming Language. whether Java provides auto-boxing also it is not an Fully OOP language because we are using primitive datatypes in form of non-object.
   - Another one reason is static members, due static member we can write Java code so that is why it is not fully object oriented programming language.

2. **What is use of OPP’S in java ? or Why we use OOP’S in java? or What is Object Oriented ?**

   ▼ Ans

   - **OOP’s stands for Object Oriented Programming System and It will provides us to achieve code**
     1. **Code Reusability.**
     2. **Removes code Duplications.**
     3. **Security for Java Application by data hiding.**
     4. **Prevents Unauthorized Access by Encapsulating the class and making the data members as private.**
     5. **Prevents Data-Mishandling through Getter’s and Setter’s.**
     6. **Removes complexity of viewing things by hiding internal implementation through the concept called Abstraction.**

We can achieve all these by using OPP'S 6principles i.e.

1. **Class**
2. **Objects**
3. **Inheritance**
4. **Polymorphism**
5. **Encapsulation**
6. **Abstraction**

3. **Explain all OOP's concept?**

▼ Ans

- **OOP'S means Object Oriented Programming System, which has 4 main principles i.e. INHERITANCE, POLYMORPHISUM, ENCAPSULATION, ABSTRUCTION. In order to achieve these principles we need CLASSES and OBJECT.**

▼ 1). INHERITANCE

- **It is the process of one class acquiring properties and behavior from another class through 'extends' keyword is called as Inheritance.**
- **Advantages of Inheritance or Why we use Inheritance**
  1. **We can achieve Code Reusability.**
  2. **We can avoid Code duplication.**
  3. **We can achieve Generalization.**
  4. **We can achieve Polymorphism.**

▼ Program for Inheritance

```
public class test
{
  String name;
  int id;

  void test1()
  {
    System.out.println("A");
  }
  void test2()
```

```
  {
    System.out.println("B");
  }
  public static void main(String[] args) {

  }
}
 class base extends test
 {
   public static void main(String[] args)
   {
    base b1=new base();
    b1.name="manu";b1.id=125;
    System.out.println(b1.name);
    System.out.println(b1.id);
    b1.test1();
    b1.test2();

  }
 }
OUTPUT:
manu
125
A
B
```

▼ **Single Level Inheritance. (Achieve by class)**

- **If a class takes properties and behavior from only one class then we call it as Single level Inheritance.**
- **Real Time Example :- Any inbuilt class is extending Object class. (ex: string class, Scanner class, String class)**

▼ **Multi-level Inheritance. (Achieve by class)**

- **If a child class is inheriting parent class as well as that child class also act as the parent class to other class then we call it as Multi-level Inheritance.**
- **Real Time Example: Throwable class—>Exception class —> RuntimeException class.**

▼ **Program**

```
//Multi level Inheritance
class test
{
  String name;
```

```
}

class base extends test
{
  int id;
}
public class Derived extends base
{
   public static void main(String[] args)
   {
     Derived d1=new Derived ();
     d1.name="manu";
     d1.id=125;
     System.out.println(d1.name+" "+d1.id);
  }
}
OUTPUT:
manu 125
```

▼ Hierarchical Level Inheritance. (Achieve by class)

- **If a class has more then one child class then it can be called as Hierarchy level Inheritance.**
- **Real Time Example: Collection Interface —> List Interface and Set Interface. List Interface —> Vector class, ArrayList class, LinkedList class**

▼ Program

```
//Hierarchy level Inheritance
class test
{
  String name;
  int id;
}

class base extends test
{
  public static void main(String[] args)
   {
     base b1=new base ();
     b1.name="manu";
     b1.id=125;
     System.out.println(b1.name+" "+b1.id);
  }
}
public class Derived extends test
{
   public static void main(String[] args)
```

```
    {
      Derived d1=new Derived ();
      d1.name="manu";
      d1.id=125;
      System.out.println(d1.name+" "+d1.id);
   }
}
```

▼ **Multiple Inheritance.** **(Achieve by Interface)**

- **Multiple inheritances means one class having more than one parent class. It is not possible through class because of Diamond problem (or Multiple Inheritance is one class implementing more then one Interface)**

▼ What is Diamond problem in java? / Why it is not possible to achieve multiple inheritance using class ? / Why a class cannot extends more than one class ?

- Diamond problem means if class extending more then one class then we are going face diamond problem.
- In diamond problem we have 2 problem

▼ Constructor chaining ambiguity

- If a class extending more then one class, then child class constructor as a inbuilt super calling statement, it will call parent class constructor. so in this case that child class constructor gets confused which constructor should i call. hence this the problem of constructor chaining ambiguity.

▼ Method binding ambiguity

- If a class extending more then one class and both class contains same method, if we call those methods in child class. JVM will get confused which method should i inherit. so this is the method binding ambiguity.

```
//Multiple Inheritance using class /// Diamond Problem not Solution
//-----------------------------------------------------------------
public class Father
{
  Father()
  {
```

```
    System.out.println("F1");
  }
  void care()
  {
    System.out.println("Father care");
  }
}
public class Mother
{
  Mother()
  {
    System.out.println("M1");
  }
  void care()
  {
    System.out.println("Mother care");
  }
}
public class Son extends Mother,Mother
{
  Son()
  {
    System.out.println("Son");
  }
  public static void main(String[] args)
  {
      Son s1=new Son();
      s1.care();
  }
}
OUTPUT:-
Compilation Error
```

▼ **Real Time Example:**

1. Arraylist class implements List Interface, Random access Interface, clonable Interface, serialization Interface.
2. Thread class implements Runnable Interface and Object class also.

▼ **Hybrid Inheritance. (Achieve by Interface)**

- **Hybrid Inheritance is a combination multiple inheritance and hierarchical inheritance. means a class contains more then one parents class and more then one child class.**

```
//Hybrid Inheritance using Interface
//-----------------------------------------------------------------------
public interface Switch
```

```
{
  abstract void on();
  abstract void off();
}
public interface Regulator
{
  abstract void incspeed();
  abstract void decspeed();
}
public class Fan implements Switch,Regulator
{
  String brand;
  String color;
  int price;

  @Override
  public void incspeed()
  {
    System.out.println("Fan Speed is Increased");
  }

  @Override
  public void decspeed()
  {
    System.out.println("Fan Speed is Decreased");
  }

  @Override
  public void on()
  {
    System.out.println("Fan on");
  }

  @Override
  public void off()
  {
    System.out.println("Fan off");
  }

  public void details()
  {
    System.out.println(this.brand+" "+this.color+" "+this.price);
  }
}
public class Table_Fan extends Fan
{
  public static void main(String[] args)
  {
    Table_Fan f=new Table_Fan();
    f.brand="Balaji";
    f.color=" White";
    f.price=2800;
    f.details();
    f.on();
    f.off();
```

```
    f.incspeed();
    f.decspeed();
  }
}
public class Ceiling_Fan extends Fan
{
  public static void main(String[] args)
  {
    Ceiling_Fan c1=new Ceiling_Fan();
    c1.brand="Crompton";
    c1.color="Black";
    c1.price=12000;
    c1.details();
    c1.on();
    c1.off();
    c1.incspeed();
    c1.decspeed();
  }
}
```

- **Real Time Example**

▼ **2). POLYMORPHISUM**

- **Polymorphism in Java allows us to perform the same action in many different ways.**

▼ **Compile time Polymorphism/(static binding/early binding).**

- In Compile Time Polymorphism the method binding decision is done by compiler based on the parameter and argument passed during compilation time. Hence the name compile time polymorphism.
- Just because method binding decision is done during the compilation time so compile time polymorphism is also called as Early Binding.
- Compile Time Polymorphism can be achieved by using **Method Overloading**.

▼ Non-Static Methods Overloading

▼ join() method of Thread class.

- join();
- join(long);
- join(long, int);

▼ Substring() method in String class

- public String substring(int);
- public String substring(int, int);

▼ add() method in ArrayList class

- public boolean add(E);
- public void add(int, E);

▼ Static Methods Overloading

▼ sleep() method in Thread class

- public static native void sleep(long);
- public static void sleep(long, int);

▼ sort() method in Collections class

- public static void sort(List);
  public static void sort(List , Comparator object);

▼ Program

```
public class Derived
{
  static void display(int x)
  {
    System.out.println("A");
  }
  static void display(String x)
  {
    System.out.println("B");
  }
  public static void main(String[] args)
  {
    display("SHb");
    display(10);
  }
}
```

▼ **Run time Polymorphism/(dynamic binding/late binding).**

- In Run Time Polymorphism the method binding decision is done by JVM based on the object creation during run time. Hence the name Run Time Polymorphism.

- Just because method binding decision is done during the run time so Run Time Polymorphism is also called as Late Binding.
- Run Time Polymorphism can be achieved by using **Method Overriding**.
- Real Time Example: toSting method, eqauls() method, hashcode method, which are present in objects class .

▼ Program

```
class test
{
  void display()
  {
    System.out.println("A");
  }
}
public class Derived extends test
{
  @Override
  void display()
  {
    System.out.println("B");
  }
  public static void main(String[] args)
  {
    Derived d1=new Derived();
    test t1=new test();
    d1.display();
    t1.display();
  }
}
OUTPUT:
B
A
```

▼ **Advantages of Polymorphism / Why we use Polymorphism**

1. It helps the programmer to reuse the codes
2. Single variable can be used to store multiple data types
3. Easy to debug the codes

▼ **3). ENCAPSULATION**

- **Encapsulation is process of warping or binding the private preparties with public data-handler method i.e. getter and setter**

method.

- **Setter method :- are used to write the data where the setter method must be of void type and must contains one parameter.**
- **Getter method :- is used to read the data and it must be return type method and should not contain any parameter.**

▼ **Main Advantages of getter and setter method**

- Some properties we can make read only by using getter() method. Ex: Id property in Thread class
- Some properties we can make write only by using setter() method.
- Some properties we can make read as well as write by using getter() and setter() method. Ex: Name and Priority properties in Thread class

- **Real Time Example: Thread class is the best example for fully Encapsulated class because all properties of Thread class are private with a getter and setter methods.**

▼ **Rules for java beans or Encapsulation:**

- Class must be public and non-abstract.
- The properties must be private.
- Each and every private property must have public getter and setter method.
- We should avoid writing parameter constructor.

▼ **Advantages of Encapsulation / Why we use Encapsulation**

1. We can achieve data hiding.
2. We can avoid data mishandling.
3. We can achieve data validation.
4. We can make the data Read only or Write only.

▼ **Purpose of Encapsulation:**

- Purpose of Encapsulation is prevent the unauthorized access. how to make this. by making all the member as private. If admin wants to checks the date then he will use getter and setter method.

▼ Program

```
class Student
{
  private String name;
  private int age;
  private double perc;

  //Setter method
  public void Setname(String name)
  {
    this.name=name;
  }
  public void Setage(int age)
  {
    if(age>=0 && age<=100)
    {
      this.age=age;
    }
    else
    {
      System.out.println("Invalid Input");
    }
  }
  public void Setperc(double perc)
  {
    if(perc>=0.0 && perc<=100.00)
    {
      this.perc=perc;
    }
    else
    {
      System.out.println("Invalid Input");
    }
  }
  //Getter method
  public String getname()
  {
    return this.name;
  }
  public int getage()
  {
    return this.age;
  }
  public double getperc()
  {
    return this.perc;
  }
}
public  class Demo
{
  public static void main(String[] args)
  {
```

```
    Student s1=new Student();
    s1.Setname("Manu");
    s1.Setage(22);
    s1.Setperc(99.99);

    System.out.println(s1.getname());
    System.out.println(s1.getage());
    System.out.println(s1.getperc());
  }
}
---------------------
OUTPUT:
Manu
22
99.99
```

▼ 4). ABSTRACTION

- **Abstraction is the process of hiding internal implementation and showing only essential information to the user is called Abstraction.**
- **Real Time Example :- CompareTo() method in Compareble Interface, run() method in Runnable class and put() method in Map interface. Because it does showing internal implementation.**

▼ Advantages of Abstraction or Why we use Abstraction

- It reduces the complexity of viewing things.
- Avoids code duplication and increase the code reusability.
- Helps to increase the security of an application by showing only essential details.
- It provides the functions like without affecting end-user we can do any type of modification in our internal system.

▼ Abstraction is mainly used for Loose coupling

- If change in the implementation does not affect the user. then it is called loose coupling.

▼ Tight coupling

- If change in the implementation will affect the user. then it is called loose coupling.

▼ Concrete class and Concrete method

Concrete class: A concrete class is a class that has an implementation for all its methods. They cannot have any unimplemented methods.

Concrete method: If the method has method declaration and method implementation then it is called Concrete Method.

▼ Abstract class and Abstract method

Abstract class: A class which is declared with the abstract keyword is known as an abstract class. It can have abstract and non-abstract methods.

Abstract method: If the method contains only method declaration but no implementation then it is called as abstract method where the method has to be terminated and method has to be prefixed with abstract keyword.

- Whenever the parent class contains any abstract method it is mandatory to Override all the abstract method in child class. is not we need make the class as abstract.

▼ Difference between abstract class and concrete class

- abstract class prefixed with “abstract” keyword.
- concrete class does not prefixed with “abstract” keyword.
- abstract class contains both abstract method and concrete method.
- concrete class contains only concrete method.
- In abstract class it is not possible to not possible to create object where as concrete class it is possible to create object.

▼ Program for Abstraction

```
abstract class Switch
{
  abstract void Swithchon();
  abstract void Swithchoff();
}

public class Fan extends Switch
{
  @Override
  public void Swithchon()
  {
    System.out.println("ON");
  }
```

```
  @Override
  public void Swithchoff()
  {
    System.out.println("OFF");
  }
  public static void main(String[] args)
  {
    Fan f1=new Fan();
    f1.Swithchon();
    f1.Swithchoff();
  }
}
--------
OUTPUT:
ON
OFF
```

▼ Can we make abstract class as final ?

- Making the class as final is to avoid inheritance but abstract class has to be inherited and abstract method has to be overriden. so hence it is not possible to make abstract class as final.

▼ Can we make abstract method as final ?

- The purpose of making a method as final is to avoid method overriding but abstract method has to be overridden in the child class. so hence abstract method cannot be final.

▼ Can we make abstract method as private ?

- Private method cannot be inherited but abstract method has to be in inherited and Overridden. so hence we cannot make abstract method as private.

▼ Can we make abstract method as static ?

- static method cannot be Overridden but abstract method has be overridden. so hence we cannot make abstract method as static.

▼ Is it necessary to write minimum one abstract method inside the abstract class ?

- With respect to syntax abstract class can contains zero abstract method no syntax error but abstract class without abstract method will not make any sense. so at least we should add one abstract method.

▼ Can we create object or instance for an abstract class ?

- abstract class is an incomplete class so hence it is not possible to create an object for abstract class and moreover abstract class may contains abstract method.

▼ Can we write constructor for an abstract class ?

- Yes, It is possible to write constructor inside an abstract class.

▼ Program

```
abstract class test
{
  int a;

  public test(int a)
  {
    this.a=a;
  }
  abstract int display(int val);
}
public class Demo extends test
{
  public Demo()
  {
    super(2);
  }
  @Override
  public int display(int val)
  {
    return this.a*val;
  }

  public static void main(String[] args)
  {
    Demo d1=new Demo();
    int cp=d1.display(3);
    System.out.println(cp);
  }
}
```

▼ Will abstract class inherit object class ?

- Yes, abstract will inherit object class.

▼ Can we overload abstract method ?

- It is possible to overload abstract method.

▼ Interface

- Interface is an intermediate between service and consumer.
- Interface is an intermediate between any a two entities.
- Interface is a blue print for a class.
- Interface variable by default public static final
- Interface method by default public abstract

▼ Program for Interface

```
//Multiple Inheritance  //Solution for Dimond Problem using Interfece.
//-------------------------------------------------------------------------
public interface Father
{
  abstract void care();
}
public interface Mother
{
  abstract void care();
}
public class Son implements Father,Mother
{
  public Son()
  {
    System.out.println("Son");
  }
  @Override
  public void care()
  {
    System.out.println("Father-Mother care");
  }
  public static void main(String[] args)
  {
    Son s1=new Son();
    s1.care();
  }
}

OUTPUT:-
Son
Father-Mother care
```

▼ Types of Interface

▼ Regular Interface

- If an interface contains zero or more then abstract method then it is called as Regular Interface.
- Regular Interface can be pre-defined as well as user defined.

▼ Marker Interface

- If an interface contains zero abstract method then it is called Marker activities.
- Marker Interface can be Serializable, cloneable and random access.
- We use marker interface to create duplicate object in heap memory by using clone method.

▼ Serializable

- The **Serializable** interface is present in **java.io** package. It is a marker interface. serialization allows us to convert the object into a byte stream

▼ cloneable

The **Java.lang** interface is a marker interface. It was introduced in JDK 1.0. this interface to create duplicate object in heap memory by using clone() method

▼ random access

▼ Functional Interface

- If an interface contains exactly one abstract method then it is called Functional Interface.
- Functional Interface can be pre-defined or user defined.
- Comparable is a type Functional Interface

▼ Where we use Comparable and Comparator

- Both Comparable and Comparator is useful while sorting data in case of Collection.

▼ Solution for Diamond problem

```
//Multiple Inheritance  //Solution for Dimond Problem using Interfece.
//-------------------------------------------------------------------------
public interface Father
{
  abstract void care();
}
public interface Mother
{
  abstract void care();
}
public class Son implements Father,Mother
{
  public Son()
  {
    System.out.println("Son");
  }
  @Override
  public void care()
  {
    System.out.println("Father-Mother care");
  }
  public static void main(String[] args)
  {
    Son s1=new Son();
    s1.care();
  }
}

OUTPUT:-
Son
Father-Mother care
```

▼ Difference between abstract class and interface

Abstract	Interface
1). “abstract” keyword is used to create abstract	1). “interface” keyword is used to create abstract
2). It can have both concrete and abstract method	2). It has abstract method only util JDK1.7 but JDK1.8 it can have both concrete and abstract method.
3). Constructor is allowed	3). Constructor is not allowed
4). GV can be static as well as non-static	4). GV are by default public static final.
5). Not possible to achieve multiple inheritance	5) possible to achieve multiple inheritance

▼ Multiple Inheritance using “implements”

▼ Multiple Inheritance using “extends”

▼ Multiple Inheritance using “implements” and “extends”

▼ Encapsulation vs Abstraction

- Encapsulation is data hiding (information hiding).

- Abstraction is details hiding (implementation hiding).

31. What is data hiding ?

▼ **Ans**

- Data hiding means **hiding the internal data within the class to prevent the direct access from outside the class is called data hiding.**

▼ How we can implement data hiding ?

- By declaring variable as a private, we ca achieve data hiding i.e. outside person unable to access to data

▼ Advantage of data hiding

- Security for the application.

▼ Example Program

```
public class Emp
{
  public static void main(String[] args)
  {
    Bank emp = new Bank();
    emp.bankid = 12345;
    emp.name = "Manu";

    emp.setbalance(10000, 12345);

    long empbalance = emp.getbalance(12345);

    System.out.println("User Name"+ "  " + emp.name);
    System.out.println("Bank ID"+ "  " + emp.bankid);
    System.out.println("Current Balance"+ "  " + empbalance);
  }
}
class Bank
{
  private long CurBalance = 0;
  long bankid;
  String name;

  public long getbalance(long Id)
  {
    if (this.bankid == Id)
    {
      return CurBalance;
    }
    return -1;
  }
  public void setbalance(long balance, long Id)
```

```
  {
    if (this.bankid == Id)
    {
      CurBalance = CurBalance + balance;
    }
  }
}
-----------------OUTPUT---------------
User Name  Manu
Bank ID  12345
Current Balance  10000
```

4. What are the types of Inheritance ?

▼ Ans

- Single Inheritance.
- Multiple Inheritance.
- Hierarchical Inheritance.
- Multi-Level Inheritance.
- Hybrid Inheritance.

5. Why Multiple and Hybrid Inheritance is not supported in java ?

▼ Ans

- Multiple inheritance means one class extending more then one parent class. If we do this by using class we will get 2 problems that is Constructor chaining ambiguity and Method Binding ambiguity. so to solve this problem we go with interface. In interface there is no concept of constructor and method binding ambiguity solved by using abstract method.

4. Difference between Abstract Class and Interface ?

▼ Ans

Abstract Class: Abstract is a keyword it does not contain method implementation and providing method declaration and if abstract method is in the super class then it mandatary to provide method implementation with different formal arguments by subclass

Interface: Interface is a component like abstract class which is used to achieve only 100% of abstraction

- In the interface all the non-static method are by default public and abstract.
- Subclass of interface is responsible to give implementation for the abstract method.
- object creation for the interface is not possible.

5. How to make class read only and write only ?

▼ Ans

We can make a class read-only and write only **by making all of the data members as private.**

If we make a class read-only, then we can't modify the properties or data members value of the class. If we made a class write-only then we can modify the properties or data member value of the class.(by using getter and setter method).

6. Can we define concrete method in Interface ?

▼ Ans

We cannot define concrete method in interface. Because all the methods in interface are abstract

7. How to invoke abstract method of a abstract class ?

▼ Ans

- We need to inherit it by extending its class and provide implementation to it.

8. Explain types of Interface ?

▼ Ans

9. How java is object oriented?

▼ Ans

- Because Java **is based on the concept of objects and classes** Without the creation of objects and classes, it is not possible to write any code in Java.

10. Explain why java is not pure object oriented?

▼ Ans

Because java uses primitive data types and static methods.

11. What is Method Binding ?

▼ Ans

=Attaching the method call with respect to method body is called Method Binding.

12. What is Dynamic Binding ?

▼ Ans

- In method overriding change in the object will lead to change in method binding so hence the name dynamic binding.
- The method binding decision is made based on object.

13. What is Static Binding ?

▼ Ans

- In method overriding change in the object will not lead to change in method binding so hence the name Static binding.
- The method binding decision is made based on reference not object.

14. What is setter and getter method ?

▼ Ans

setter method : is used to write the data and setter method must be in void type and must contains one parameter.

getter method : is used to read the data and getter method must be in return type method and should not contain any parameter.

15. What is loose coupling and Tight coupling ?

▼ And

16. Real time Example of Hybrid Inheritance ?

▼ Ans

17. Abstract class real time example ?

▼ Ans

18. Why we use Interface

▼ Ans

by using interface we can achieve 1.100% abstraction 2.generalization in collection 3.multiple inheritance

19. Why we go with Class, Abstract class and Interface ?

▼ Ans

class : class used for permanent implementation. If don't want modify implementation. we go with class

abstract class: we want to modify or change implementation after some period we go abstract class.

ex: dictionary class present class library.

Interface : for collection

20. Can we use abstract class and make 100% abstraction ? if yes then Why we go with Interface ?

▼ Ans

- Yes, It is possible to make 0 to 100% Abstraction by using abstract class and abstract method. but It is not possible to achieve multiple Inheritance so that is why we go with Interface. In Interface we can achieve multiple Inheritance(solution for dimond problem.

21. Can we achieve multiple inheritance using abstract class? If no why ?

▼ Ans

22. Method overloading is Example of compile time polymorphism then why cannot we say constructor overloading is also example of Compile time polymorphism ?

▼ Ans

- yes, we can say.

23. What is java beans ?

▼ Ans

24. When Interface was introduced means from which JDK ?

▼ Ans

25. How do you mandate or impose Inheritance ?

▼ Ans

26. Explain all the object oriented principle with real-time examples ?

▼ Ans

27. Explain java bean specification ?

▼ Ans

28. Explain steps to achieve Encapsulation ?

▼ Ans

29. How to invoke the abstract methods of abstract class or interface ?

▼ Ans

30. Can you define concrete method in Interface ?

▼ Ans

31. What is marker interface ? What is use of Marker Interface ? Can we define our own marker interface ? Name few marker interface available in java ?

▼ Ans

32. Can we define protected abstract method in Interface ?

▼ Ans

33. What types of method cannot be abstract ?

▼ Ans

34. How many ways we can an object ?

▼ Ans

35. What is Pointers ?

▼ Ans

36. Why should we prefer Java instead of C or C++ ?

▼ Ans