Object Oriented Programming

Interview Questions

----------**********----------

1. Inheritance

----------**********----------

- what is inheritance?
    - Inheritance is one of the oops concepts in java.inheritance is concept of
    - Getting properties of one class object to another class object.
    - Inheritance represents the IS-A relationship, also known as parent-child relationship.
- What are the types of inheritance?
    - Multiple inheritance (java doesn't support multiple inheritance).
    - Multilevel inheritance.
- How Inheritance can be implemented in java?
    - Inheritance can be implemented in JAVA using below two keywords:
    - extends
    - Implements
        - Extends is used for developing inheritance between two classes and two interfaces.
        - Implements keyword is used to developed inheritance between interface and class.
- Why we need to use Inheritance?
    - For Code Re usability.
    - For Method Overriding.
- What is syntax of inheritance?
    - public class subclass extends superclass{
    - //all methods and variables declare here
    - }
    - What is multilevel inheritance?

- Getting the properties from one class object to another class object level wise with different priorities.

- What is multiple inheritance? Why Java Doesn't Support multiple Inheritance.
  - The concept of Getting the properties from multiple class objects to sub class
  - Object with same priorities is known as multiple inheritance.
  - In multiple inheritance there is every chance of multiple properties of multiple
  - Objects with the same name available to the sub class object with same priorities leads for the ambiguity.
  - Also known as diamond problem. One class extending two super classes.
  - Because of multiple inheritance there is chance of the root object getting created more than once.
  - Always the root object i.e. object of object class hast to be created only once.
  - Because of above mentioned reasons multiple inheritance would not be supported by java.
  - Thus in java a class cannot extend more than one class simultaneously. At most a class can extend only one class.

- How do you implement multiple inheritance in java?
  - Using interfaces java can support multiple inheritance concept in java.
  - in java cannot extend more than one classes, but a class can implement more than one interfaces.
  - Program:

  - interface A{

  - }
  - interface B{
  - }
  - class C implements A,B{
  - }
- Can a class extend itself?
    - No, A class can't extend itself.
- What happens if super class and sub class having same field name?
    - Super class field will be hidden in the sub class. You can access hidden super class field in sub class using super keyword.

- xi What are the types of inheritance.?

- There are 5 types of inheritance.

- Single Inheritance: One class is extended by only one class.

- Multilevel Inheritance: One class is extended by a class and that class in turn is extended by another class thus forming a chain of inheritance.

- Hierarchical Inheritance: One class is extended by many classes.
- Hybrid Inheritance: It is a combination of above types of inheritance.
- Multiple Inheritance: One class extends more than one classes. (Java does not support multiple inheritance.)

2. Polymorphism

----------********----------**

- Q1) what is polymorphism?

    - Ans) The ability to identify a function to run is called Polymorphism. In java, c++ there are two types of polymorphism: compile time polymorphism (overloading) and runtime polymorphism (overriding).

    - Mehtod overriding: Overriding occurs when a class method has the same name and signature as a method in parent class. When you override methods, JVM determines the proper method to call at the program's run time, not at the compile time.

    - Overloading: Overloading is determined at the compile time. It occurs when several methods have same names with:

- Different method signature and different number or type of parameters.
- Same method signature but different number of parameters.
- Same method signature and same number of parameters but of different type

- Example of Overloading
- int add(int a,int b)
    - float add(float a,int b)
    - float add(int a ,float b)
    - void add(float a)
    - int add(int a)
    - void add(int a) //error conflict with the method int add(int a)
- class BookDetails {
    - String title;
    - setBook(String title){}
- }
- class ScienceBook extends BookDetails {
- setBook(String title){} //overriding
- setBook(String title, String publisher,float price){} //overloading
- }

➢ What is Polymorphism?
- Polymorphism is the ability of an object to take on many forms.
- The most common use of polymorphism in OOP occurs when a parent
- class reference is used to refer to a child class object.

- What is polymorphism in Java
- Polymorphism is an Oops concept which advice use of common interface instead of concrete implementation while writing code.
- When we program for interface our code is capable of handling any new requirement or enhancement arise in near future
- due to new implementation of our common interface. If we don't use common interface and rely on concrete implementation,
- we always need to change and duplicate most of our code to support new implementation.
- Its not only Java but other object oriented language like C++ also supports polymorphism and it
- comes as fundamental along with other OOPS concepts like Encapsulation , Abstraction and Inheritance.

- What is function overloading?
    - If a class has multiple functions by same name but different parameters, it is known as Method Overloading.

- Difference between Overloading and Overriding?
    - Method overloading increases the readability of the program.
    - Method overriding provides the specific implementation of the method that is already provided
    - By its super class parameter must be different in case of overloading, parameter must be same in case of overriding.

- What is Function Overriding and Overloading in Java?
    - overloading in Java occurs when two or more methods in the same class have the exact same name,
    - But different parameters. On the other hand, method overriding is defined as the case when a child
    - Class redefines the same method as a parent class. Overridden methods must have the same name, argument list,
    - And return type. The overriding method may not limit the access of the method it overrides.

3. Encapsulation

----------********----------**

- What is Encapsulation?
    - It is the technique of making the fields in a class private and providing access to the fields via public methods.
    - If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class.
    - Therefore encapsulation is also referred to as data hiding.

- What is the primary benefit of Encapsulation?

- The main benefit of encapsulation is the ability to modify our implemented code without breaking
- The code of others who use our code. With this Encapsulation gives maintainability, flexibility and extensibility to our code.

- What is an Interface?
  - An interface is a collection of abstract methods. A class implements an interface,
  - Thereby inheriting the abstract methods of the interface..

  - Give some features of Interface?
    - It includes -
- Interface cannot be instantiated
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.

- What is difference between Encapsulation and Abstraction?
  - Abstraction solves the problem at design level while encapsulation solves the problem at implementation level
  - Abstraction is used for hiding the unwanted data and giving relevant data.
    - While Encapsulation means hiding the code and data into a single unit to protect the data from outside world.
  - Abstraction lets you focus on what the object does instead of how it does it while Encapsulation means hiding
    - The internal details or mechanics of how an object does something.
  - For example: Outer Look of a Television, like it has a display screen and channel buttons to change channel it
    - explains Abstraction but Inner Implementation detail of a Television how CRT and Display Screen are connect with
    - each other using different circuits , it explains Encapsulation.

- What are the features of encapsulation?
  - Combine the data of our application and its manipulation at one place. Encapsulation Allow the state of an object
  - To be accessed and modified through behaviour. Reduce the coupling of modules and increase the cohesion inside them.

- Explain in detail what is Encapsulation in Java?
  - Encapsulation is nothing but protecting anything which is prone to change. rational behind encapsulation is

- that if any functionality which is well encapsulated in code i.e maintained in just one place and not scattered
- Around code is easy to change. This can be better explained with a simple example of encapsulation in Java.
- We all know that constructor is used to create object in Java and constructor can accept argument.
- Suppose we have a class Loan has a constructor and then in various classes you have created instance of loan by using this constructor.
- Now requirements change and you need to include age of borrower as well while taking loan.
- Since this code is not well encapsulated i.e. not confined in one place you need to change
- everywhere you are calling this constructor i.e. for one change you need to modify several file instead
- of just one file which is more error prone and tedious, though it can be done with refactoring feature of
- Advanced IDE wouldn't it be better if you only need to make change at one place? Yes that is possible if we
- encapsulate Loan creation logic in one method say create Loan() and client code call this method and this method internally
- Crate Loan object. In this case you only need to modify this method instead of all client code.

- Example of Encapsulation in Java

- class Loan{
    - private int duration; //private variables examples of encapsulation
    - private String loan;
    - private String borrower;
    - private String salary;

    - //public constructor can break encapsulation instead use factory method
    - private Loan(int duration, String loan, String borrower, String salary){
        - this.duration = duration;
        - this.loan = loan;
        - this.borrower = borrower;
        - this.salary = salary;

        - }

        - //no argument consustructor omitted here
        - // create loan can encapsulate loan creation logic
        - public Loan createLoan(String loanType){
            - //processing based on loan type and then returning loan object
            - return loan;
        - }
- }

- Advantage of Encapsulation in Java and OOPS?
    - Here are few advantages of using Encapsulation while writing code in Java or any Object oriented programming language:

- Encapsulated Code is more flexible and easy to change with new requirements.
- Encapsulation in Java makes unit testing easy.
- Encapsulation in Java allows you to control who can access what.
- Encapsulation also helps to write immutable class in Java which are a good choice in multi-threading environment.
- Encapsulation reduce coupling of modules and increase cohesion inside a module because all piece of one thing are encapsulated in one place.
- Encapsulation allows you to change one part of code without affecting other part of code.

    - What should you encapsulate in code
        - Anything which can be change and more likely to change in near future is candidate of Encapsulation.
        - This also helps to write more specific and cohesive code. Example of this is object creation code,
        - Code which can be improved in future like sorting and searching logic.

    - Important point's about encapsulation in Java.
- "Whatever changes encapsulate it" is a famous design principle.
- Encapsulation helps in loose coupling and high cohesion of code.
- Encapsulation in Java is achieved using access modifier private, protected and public.
- Factory pattern, Singleton pattern in Java makes good use of Encapsulation.

- Design Pattern based on Encapsulation in Java
- Many design pattern in Java uses encapsulation concept, one of them is Factory pattern which is
- Used to create objects. Factory pattern is better choice than new operator for creating object of
- Those classes whose creation logic can vary and also for creating different implementation of same interface.
- Border Factory class of JDK is a good example of encapsulation in Java which creates different types of Border
- And encapsulate creation logic of Border. Singleton pattern in Java also encapsulate how you create instance by
- Providing getInstance() method. since object is created inside one class and not from any other place in code
- You can easily change how you create object without affect other part of code.

- What is the Benefits of Encapsulation?
  - The fields of a class can be made read-only or write-only.
  - A class can have total control over what is stored in its fields.
  - The users of a class do not know how the class stores its data.
    - A class can change the data type of a field and users of the class do not need to change any of their code..

- Example of how to achieve Encapsulation in Java?
  - To achieve encapsulation in Java
  - Declare the variables of a class as private.
  - Provide public setter and getter methods to modify and view the variables values.
    - Below given is an example that demonstrates how to achieve Encapsulation in Java:
    - /* File name : EncapTest.java */

    - public class EncapTest{
      - private String name;
      - private String idNum;
      - private int age;

      - public int getAge(){
        - return age;
      - }

- public String getName(){
    - return name;
- }
- public String getIdNum(){
    - return idNum;
- }
- public void setAge(int newAge){
    - age = newAge;
- }
- public void setName(String newName){
    - name = newName;
- }
- public void setIdNum(String newId){
    - idNum = newId;
- }

- }

- How to variables of the EncapTest class can be accessed?
    - The public setXXX() and getXXX() methods are the access points of the instance variables of
    - The EncapTest class. Normally, these methods are referred as getters and setters.
    - Therefore any class that wants to access the variables should access them through these getters and setters.
    - The variables of the EncapTest class can be accessed as below::

    - /* File name : RunEncap.java */
    - public class RunEncap{

        - public static void main(String args[]){
            - EncapTest encap = new EncapTest();
            - encap.setName("James");
            - encap.setAge(20);
            - encap.setIdNum("12343ms");
            - System.out.print("Name : " + encap.getName() + " Age : " + encap.getAge());
        - }
    - }
    - This would produce the following result:

4. Abstraction

----------**********----------

- What is Abstraction?
    - It refers to the ability to make a class abstract in OOP.
    - It helps to reduce the complexity and also improves the maintainability of the system

- What is Abstract class?
    - These classes cannot be instantiated and are either partially implemented or not at all implemented.
    - This class contains one or more abstract methods which are simply method declarations without a body.

- When Abstract methods are used?
    - If you want a class to contain a particular method but you want the actual
    - implementation of that method to be determined by child classes, you can
    - Declare the method in the parent class as abstract.

- Can an interface extend another interface in Java?
    - Yes, an interface can extend another interface in Java.
    - This is what the code for something like that would look like:
    - // this interface extends from the Body interface:

    - public interface FourLegs extends Body {
        - public void walkWithFourLegs( );
    - }
- When would we want an interface to extend another interface?
    - Remember that any class that implements an interface must implement
    - The method headings that are declared in that interface.
    - And, if that interface extends from other interfaces, then the implementing

- Class must also implement the methods in the interfaces that are being extended or derived from.
- So, in the example above, if we have a class that implements the FourLegs interface,
- Then that class must have definitions for any method headings in both the FourLegs interface and the Body interface.

- Could you differentiate an Interface and an Abstract class?
    - An abstract class may have instance methods, which can implement a default behavior.
    - On the other hand, an interface can't implement any default behavior.
    - However, it can declare different constants and instance methods.
    - While an interface has all the public members, an abstract class contains only class members like private, protected and so on.

- What is Marker Interface?
    - Marker interface in Java is interfaces with no field or methods.
    - Uses of Mark Interfaces are following:
    - We use Marker interface to tell java compiler to add special behavior to the class implementing it.
    - Java marker interface has no members in it.
    - It is implemented by classes in get some functionality.

    - Example: when we want to save the state of an object then we implement serializable interface.

- Can an inner class be built in an Interface?
    - Yes, an inner class may be built an Interface

    - Example :
    - public interface xyz{
        - static int p=0;
        - void m();
        - class c{
            - c(){
                - int q;
                - System.out.println("inside");
            - };
            - public static void main(String c[]){

        - System.out.println("inside ");
    - }
- }
- }

- How to define an Abstract class?
    - A class containing abstract method is called Abstract class. An Abstract class can't be instantiated.

    - Example of Abstract class :
    - abstract class testAbstractClass{
    - protected String myString;
    - public String getMyString(){
        - return myString;
    - }

    - public abstract string anyAbstractFunction();

    - }

- How to define an Interface?
    - In Java Interface defines the methods but does not implement them. Interface can include constants.
    - A class that implements the interfaces is bound to implement all the methods defined in Interface.

    - Example of Interface :
    - public interface sampleInterface {

        - public void functionOne();
        - public long CONSTANT_ONE = 1000;

    - }

- Why will you use Comparator and Comparable interfaces?
  - java.util.Comparator
  - java.util.Comparator compares some other class's instances,
  - java.lang.Comparable
  - java.lang.Comparable compares another object with itself .

- What's the difference between an interface and an abstract class? Also discuss the similarities?

  - Abstract class is a class which contain one or more abstract methods, which has to be implemented by sub classes.
  - Interface is a Java Object containing method declaration and doesn't contain implementation.
  - The classes which have implementing the Interfaces must provide the method definition for all
  - The methods Abstract class is a Class prefix with a abstract keyword followed by Class definition.
  - Interface is an Interface which starts with interface keyword. Abstract class contains one or more abstract methods.
  - where Interface contains all abstract methods and final declarations
  - Abstract classes are useful in a situation that Some general methods should be
  - Implemented and specialization behavior should be implemented by child classes.
  - Interfaces are useful in a situation that all properties should be implemented.

  - ---Differences are as follows ::

    - Interfaces provide a form of multiple inheritance. A class can extend only one other class.
    - Interfaces are limited to public methods and constants with no implementation. Abstract classes
  - Can have a partial implementation, protected parts, static methods, etc.
    - A Class may implement several interfaces. But in case of abstract class, a class may extend only one abstract class.
    - Interfaces are slow as it requires extra indirection to to find corresponding method in in the actual class. Abstract classes are fast.
  - ---Similarities ::
    - Either Abstract classes or Interface can be instantiated.