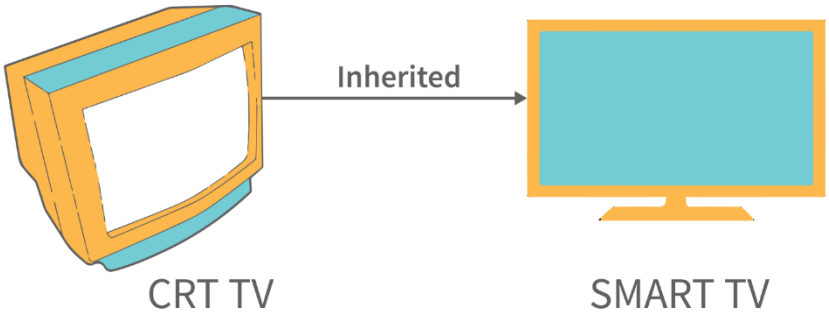
IMPORTANT INTERVIEW QUESTIONS

1. **What is the 'IS-A' relationship in OOPs java?**

Ans: 'IS-A' relationship is another name for inheritance. When we inherit the base class from the derived class, then it forms a relationship between the classes. So that relationship is termed an 'IS-A' Relationship.

Example - Consider a Television (Typical CRT TV). Now another Smart TV that is inherited from television class. So, we can say that the Smart TV is also a TV. Because CRT TV things can also be done in the Smart TV.



2. **What benefits does inheritance offer in Java?**

Ans: The following are some benefits of inheritance in Java:

- By placing the common code in the superclass and distributing it among various subclasses, we can reduce the amount of duplicate code in an application.
- The application's redundancy is decreased as a result of shorter code.
- Application code may become more adaptable through inheritance.

3. **Why is inheritance necessary?**

Ans: One of the fundamental elements of the OOPs paradigm is inheritance. The usage of inheritance in Java is justified for the reasons listed below.

- The basic class's code is reusable.
- By overriding, we can extend the functionality of a class or method via inheritance.
- The features of a class that already exist are utilized through inheritance.
- It is used to implement method overriding, often known as runtime polymorphism.

4. **In Java, are static members passed down to subclasses?**

Ans: Static variables are inherited as long as they're are not hidden by another static variable with the same identifier.

A static method from the superclass is inherited as a static member by the subclass, while non-static methods are only inherited as non-static members.

Consider the following code:

```
class Parent {
     static String id = "Parent";
     static String name = "Parent";
     static void display() {
          System.out.println("From Parent class");
     }
}

class Child extends Parent {
     static String name = "Child";
     public static void main(String [] args)
     {
         System.out.println(Child.id);
         System.out.println(Child.name);
         Child.display();
     }
}
```

Output:

Parent
Child
From Parent class

5. What does "co-variant Method Overriding" means?

Ans: One of the rules of method overriding is that the return type of the overriding method must be the same as the overridden method; however, starting with Java 1.5, this requirement has been slightly eased and the overridden method can now return a subclass of the return type of the original method.

Casting at the client end can be eliminated thanks to a relaxation technique called co-variant method overriding.

The clone() method overriding is one of the best examples. The Object.clone() method returns an Object that needs to be cast, but by overriding the co-variant method, you can return the appropriate type immediately.

6. What does Java's method hiding mean?

Ans: Because Java's static methods' method calls are resolved at compile time, they cannot be modified. However, this did not stop you from declaring a method with the same name in a subclass.

In this instance, we say that a static method in the parent class was hidden by a method in the subclass.

Because overloading is handled at compile time, if a variable in the parent class points to an object in the child class, the parent class's static method will also be called.

7. Is it possible to block method overriding without the final modifier?

Ans: Yes, it is possible to block method overriding without using the final modifier in Java. This can be achieved by using private methods or by implementing the method in a way that makes it non-overridable.

Here are a couple of ways to block method overriding:

- **Private Methods:** Private methods are not inherited by subclasses, so they cannot be overridden.
- **Static Methods:** Static methods belong to the class rather than the instance of the class. Hence, they cannot be overridden (but they can be hidden).

8. What distinguishes inheritance from encapsulation?

Ans: A parent-child relationship is created via the object-oriented idea of inheritance. Although it serves as the foundation for polymorphism, it is one approach to reuse code created for parent classes.

On the other hand, encapsulation is an object-oriented notion used to conceal a class's internal details, such as How to store elements and generate hash values are both covered by HashMap.

9. What distinguishes overriding from overloading methods?

Ans: The primary distinction between overloading and overriding is that the former occurred during compile time while the latter occurred during run time. This is the cause.

Only virtual methods in Java can be overloaded. Methods resolved at compile time, such as private, static, and final methods in Java, cannot be overridden.

Additionally, the conditions for overloading and overriding methods differ. For instance, a method must have a different method signature to be considered overloading, whereas it must have the same method signature to be considered overriding.

10. What is an interface in Java and how is it different from a class?

Ans: An interface is a collection of abstract methods, which means they do not have a body. You can only implement them in a class. A class that implements an interface requires concrete implementation for all the interface's methods.

A class can have both abstract and concrete methods and can also have fields and constructors. It can also inherit from a single superclass; in which case it can implement multiple interfaces.

11. Can an interface have a constructor?

Ans: An interface in Java cannot have a constructor, as the primary function of constructors is to initialise the state of an object after its creation, and interfaces do not have a state. Interfaces typically define a contract for a set of methods that a class implements. You cannot initialise them. An interface can have only abstract methods and constant variables but not constructors or instance variables.

12. What is polymorphism in Java?

Ans: In Java, polymorphism is the ability of an object to take on many forms. One way to achieve this in Java is by using interfaces. An interface defines a contract for a set of methods that a class implements. When a class implements an interface, it can provide its own implementation for the methods in the interface. This means that multiple classes can implement the same interface and provide their own unique implementation for the interface's methods.

13. How can you prevent a class from implementing an interface?

Ans: There are various ways in which you can prevent a class from implementing an interface.

- You can declare a class as final, which prevents that class from extending. This inhibits it from implementing an interface.
- Another way is to use a sealed class (Java 17 and later). A sealed class can specify which classes can extend or implement that sealed class.

```
public sealed interface A permits B {
    void myMethod();
}
```

14. Using relevant properties highlight the differences between interfaces and abstract classes.

Ans: The differentiating properties of an abstract class and an interface are:

- **Availability of methods:** Only abstract methods are available in interfaces, whereas non-abstract methods can be present along with abstract methods in abstract classes.

- **Variable types:** Static and final variables can only be declared in the case of interfaces, whereas abstract classes can also have non-static and non-final variables.
- **Inheritance:** Multiple inheritances are facilitated by interfaces, whereas abstract classes do not promote multiple inheritances.
- **Data member accessibility:** By default, the class data members of interfaces are of the public- type. Conversely, the class members for an abstract class can be protected or private also.
- **Implementation:** With the help of an abstract class, the implementation of an interface is easily possible. However, the converse is not true

15. What do you understand by marker interfaces in Java?

Ans: Marker interfaces, also known as tagging interfaces are those interfaces that have no methods and constants defined in them. They are there for helping the compiler and JVM to get run time-related information regarding the objects.

16. In case a package has sub packages, will it suffice to import only the main package? e.g. Does importing of com.myMainPackage.* also import com.myMainPackage.mySubPackage.*?

Ans: This is a big NO. We need to understand that the importing of the sub-packages of a package needs to be done explicitly. Importing the parent package only results in the import of the classes within it and not the contents of its child/sub-packages.

17. Can we import same package/class twice? Will the JVM load the package twice at runtime?

Ans: One can import the same package or same class multiple times. Neither compiler nor JVM complains anything about it. And the JVM will internally load the class only once no matter how many times you import the same class.

Reference:

- https://www.interviewbit.com/java-interview-questions/
- https://www.naukri.com/code360/library/java-inheritance-interview-questions
- https://in.indeed.com/career-advice/interviewing/interface-interview-questions-in-java

**********END**********