**Static**

**Static Variable:**

A static variable is a variable that is associated with the class itself rather than with instances (objects) of the class. There is only one copy of the static variable shared among all instances of the class. It retains its value across different instances.

**Static Method:**

A static method is a method that belongs to the class itself, not to any specific instance of the class. It can be called using the class name without creating an instance of the class.

**Static Block:**

A static block is a block of code that is executed when the class is loaded by the JVM. It is used to initialize static variables or perform other static tasks before any instance of the class is created.

**OOPs Interview Questions**

**Object-oriented programming** (OOPs) is a programming paradigm **that is based on** **the concept of objects rather than just functions and procedures**. It is the most popular methodology among developers.

### **What is the purpose of using OOPs concepts?**

The aim of OOP is to implement real-world entities like inheritance, hiding, polymorphism in programming**. The main purpose of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.**

# Constructors in Java

In [Java](https://www.javatpoint.com/java-tutorial), a constructor is a block of codes similar to the method. It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java) is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

**Note:** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

## **Types of Java constructors**

1. Default constructor (no-arg constructor)
2. Parameterized constructor

### **What is the purpose of a default constructor?**

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

### **Why use the parameterized constructor?**

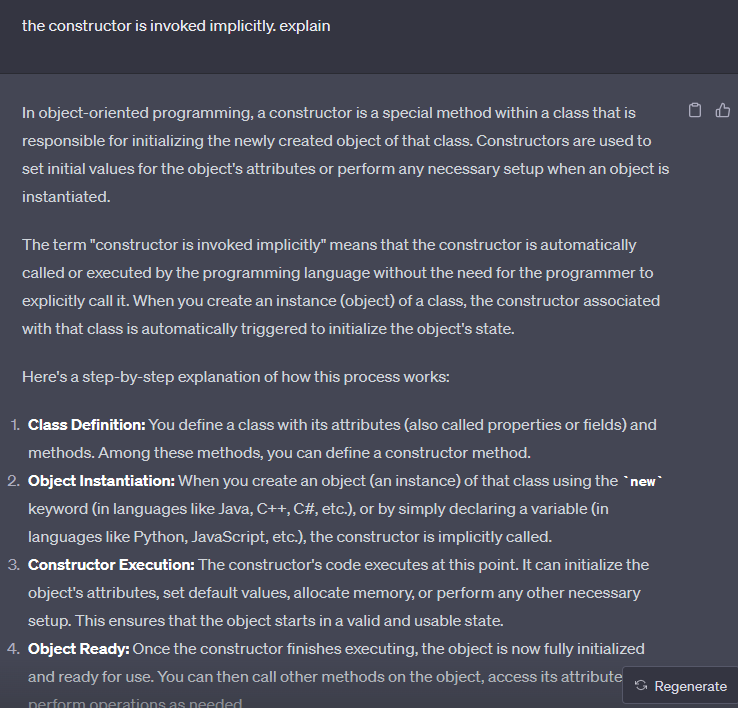
The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

## **Java Copy Constructor**

**There is no copy constructor** **in Java**. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

* By constructor
* By assigning the values of one object into another
* By clone() method of Object class



**why we need constructor**

Constructors serve several important purposes in object-oriented programming, providing essential functionality that contributes to the integrity, initialization, and usability of objects within a class. Here are some reasons why constructors are needed:

1. **Object Initialization:** Constructors ensure that newly created objects are properly initialized with valid initial values for their attributes. This prevents objects from being in an undefined or inconsistent state when they are first created.
2. **Setting Default Values:** Constructors can be used to set default values for attributes that may not be explicitly provided during object creation. This ensures that objects have sensible values even if all attributes are not explicitly initialized.
3. **Dependency Injection:** Constructors allow you to inject dependencies into an object. If an object relies on other objects or services to function properly, you can pass those dependencies through the constructor, promoting better encapsulation and testability.
4. **Validation and Error Handling:** Constructors can include validation logic to ensure that the provided parameters are valid. This helps catch errors or incorrect inputs early in the object's lifecycle.
5. **Memory Allocation and Resource Management:** Constructors can allocate memory or acquire resources that the object needs to function. This includes tasks like opening files, establishing network connections, or allocating memory on the heap.
6. **Code Reusability:** Constructors facilitate code reuse by allowing you to create objects with specific behaviors and attributes without duplicating code. You can define different constructors with varying parameter lists to offer different ways of initializing objects.
7. **Encapsulation and Abstraction:** Constructors contribute to encapsulation, a fundamental principle of object-oriented programming. By encapsulating the initialization logic within the constructor, you hide the internal implementation details of the object, providing a clear interface for interacting with it.
8. **Consistency:** Constructors ensure that all objects of a class are initialized in a consistent manner. This helps maintain a uniform structure and behavior across instances.
9. **Inheritance:** Constructors are called in a hierarchical manner when objects are created from subclasses. This ensures that the initialization logic of the superclass is also executed when a subclass object is instantiated.

**Final**

Certainly! In Java, the **final keyword is used to indicate that an entity, such as a variable, method, or class, cannot be modified or extended once it has been defined**.

**1. final Variable:**

**A final variable is a constant whose value cannot be changed once it is assigned. This is often used to represent values that should remain constant throughout the execution of a program.**

**2. final Method:**

**A final method is a method that cannot be overridden by a subclass. This is used to prevent subclasses from altering the behavior of the method defined in the superclass.**

**3. final Class:**

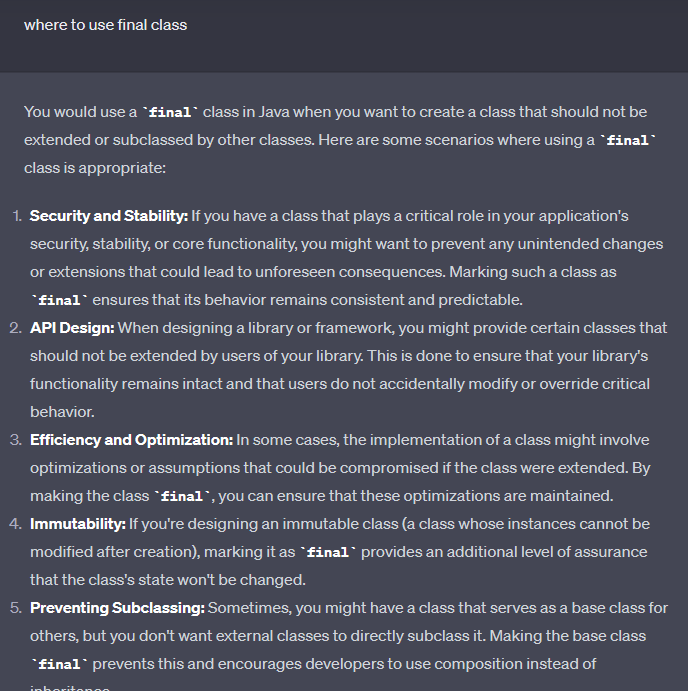
**A final class is a class that cannot be subclassed. It is used to ensure that the class's behavior and structure remain unchanged, preventing any further extension.**

To summarize:

final variable: A constant value that cannot be changed after initialization.

final method: A method that cannot be overridden by subclasses.

final class: A class that cannot be subclassed.



Remember that using **final** classes is a design choice that should be made based on the specific requirements and goals of your application. In some cases, you might want to provide extension points for developers, which would require not using the **final**

### **What are the four main features of OOPs?**

The OOP has the following four features:

* Inheritance
* Encapsulation
* Polymorphism
* Data Abstraction

# Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).

**Abstraction** *is a process of hiding the implementation details and showing only functionality to the user.*

*it shows only essential things to the user and hides the internal details,*

### **Ways to achieve Abstraction**

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

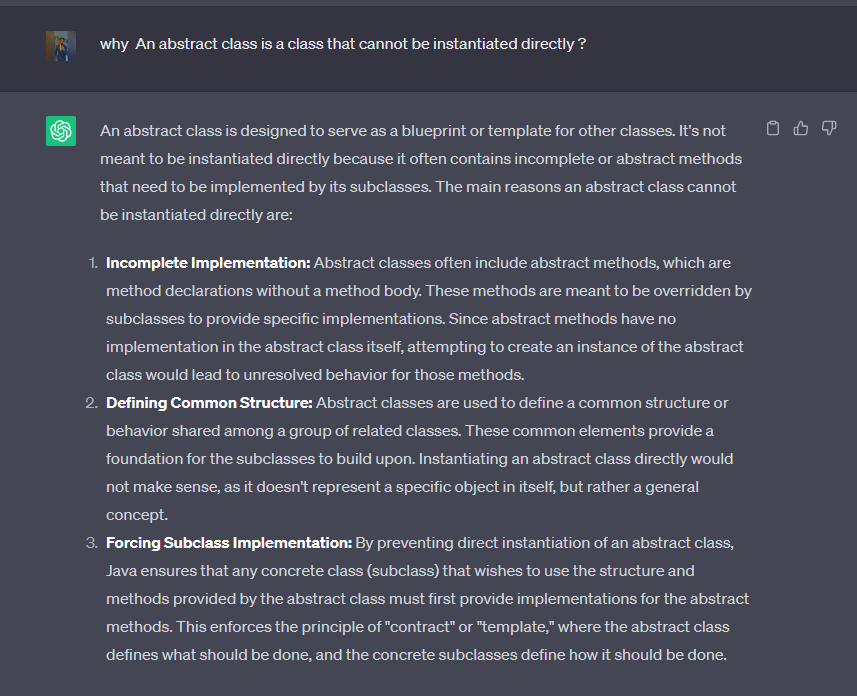
### **Abstract class in Java**

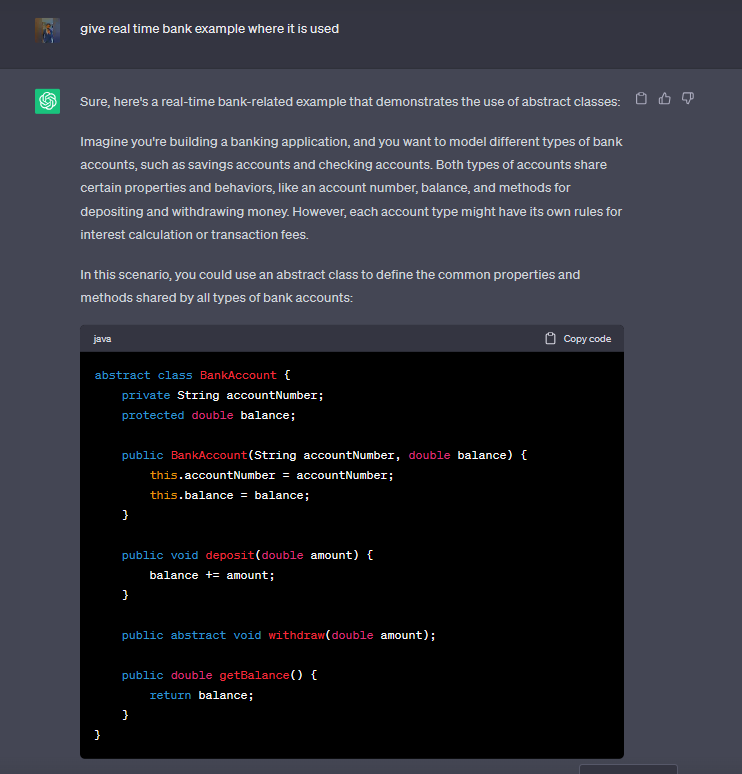
A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

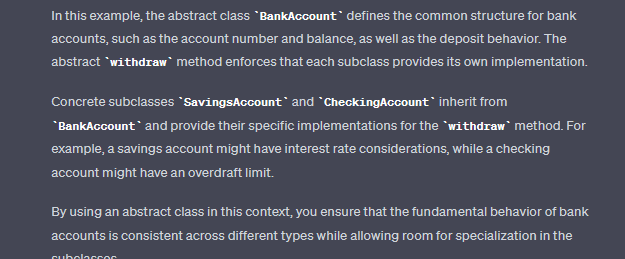
#### **Points to Remember**

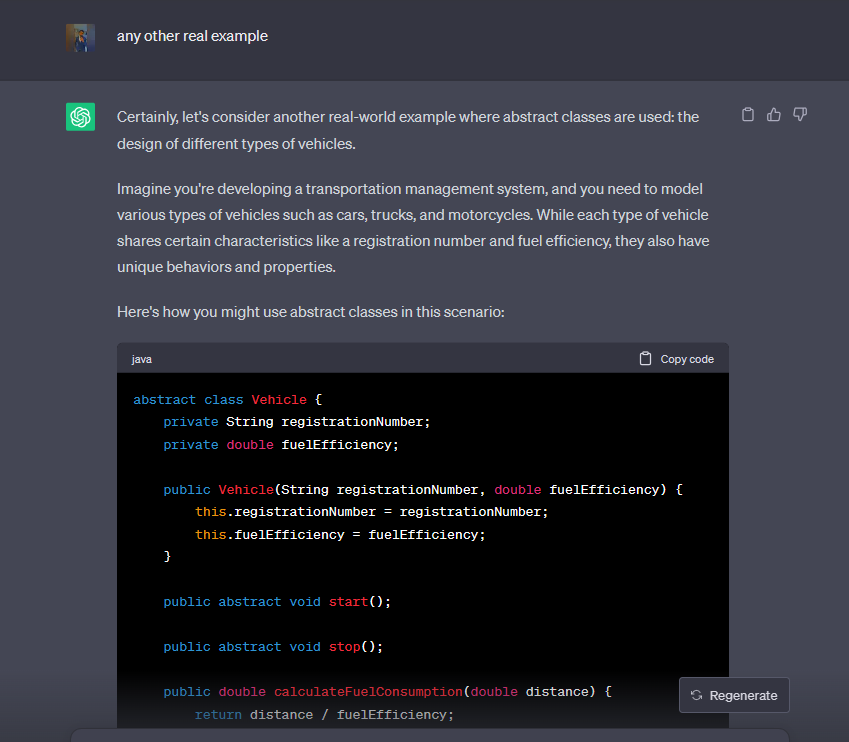
* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It **cannot be instantiated**.
* It can ***have***[***constructors***](https://www.javatpoint.com/java-constructor)***and static methods*** also.
* **It can have final methods which will force the subclass not to change the body of the method.**

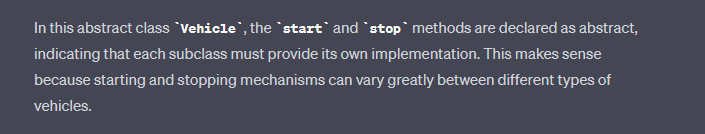
Cannot be Instantiated: You cannot create an instance (object) of an abstract class directly using the new keyword. It exists solely to be subclassed by other classes.

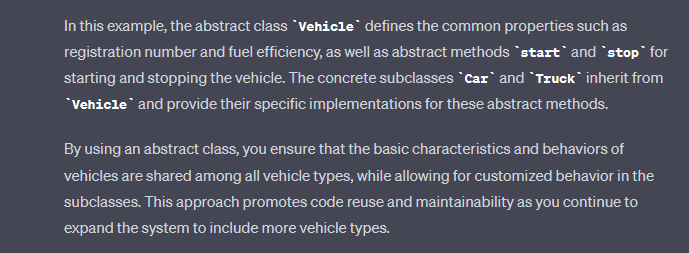


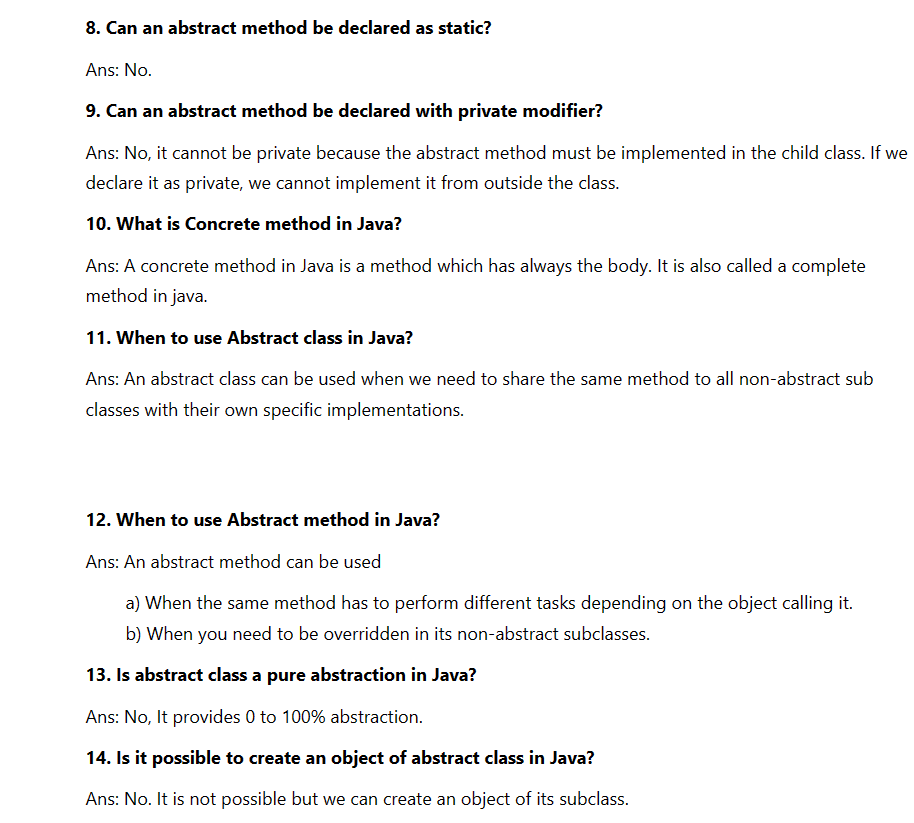


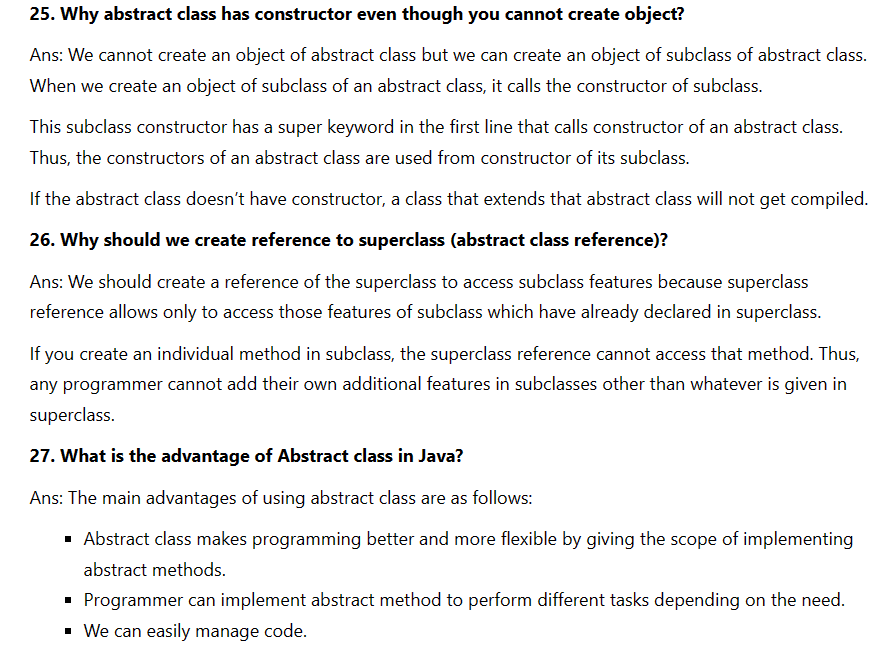


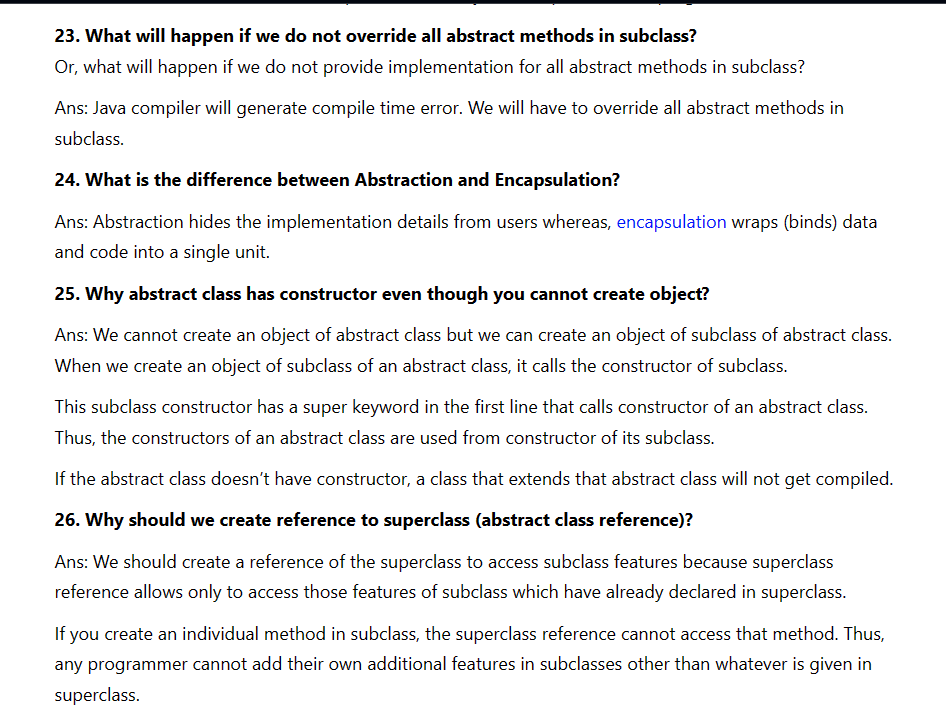












# Inheritance in Java

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

## **Single Inheritance Example**

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

## **Hierarchical Inheritance Example**

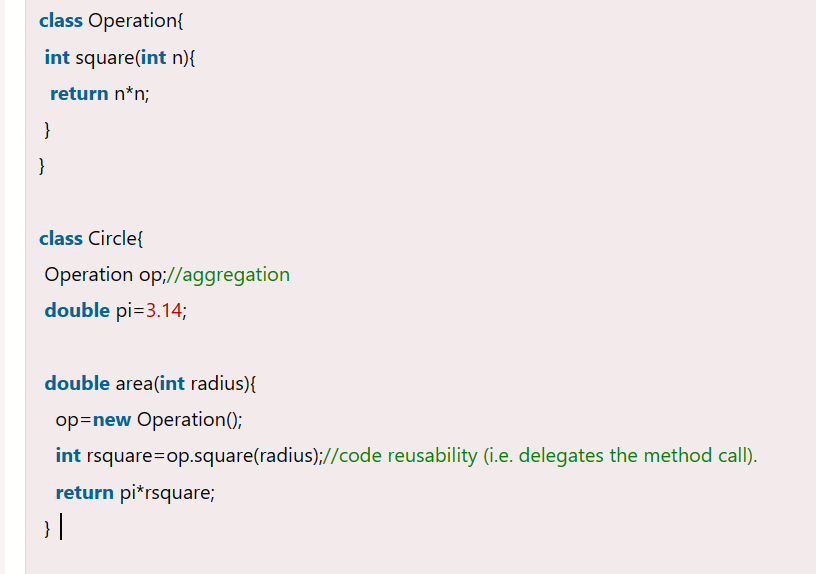
When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

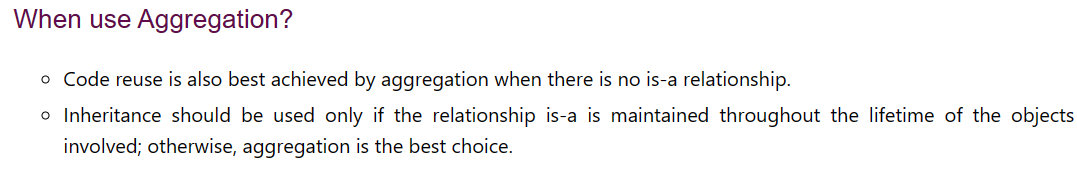
## **Multilevel Inheritance Example**

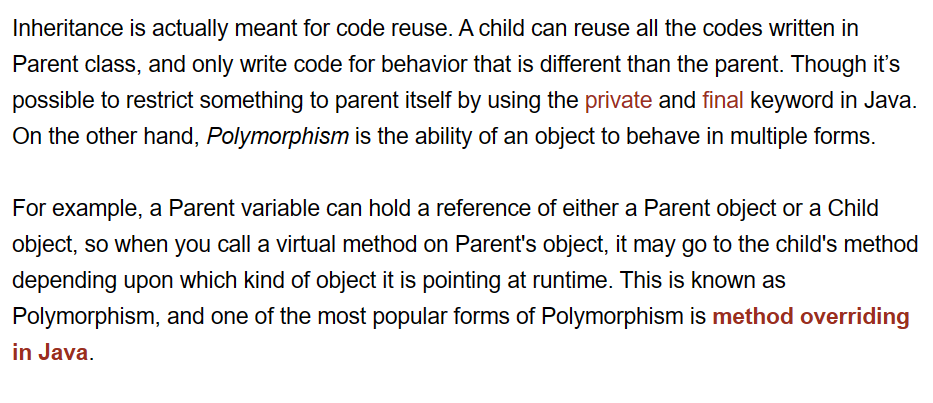
When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

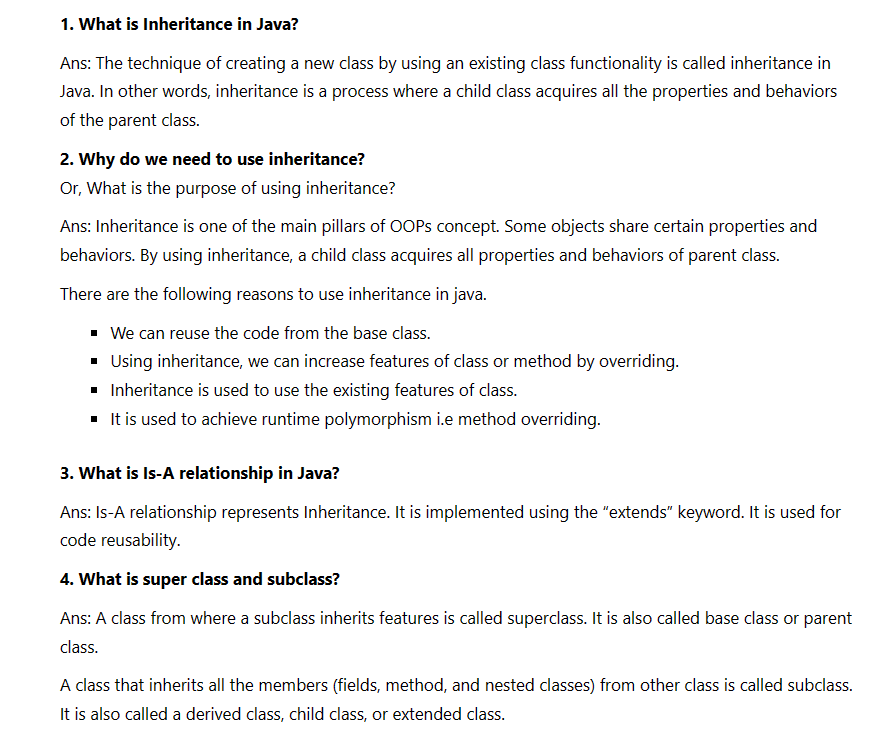
# Aggregation in Java

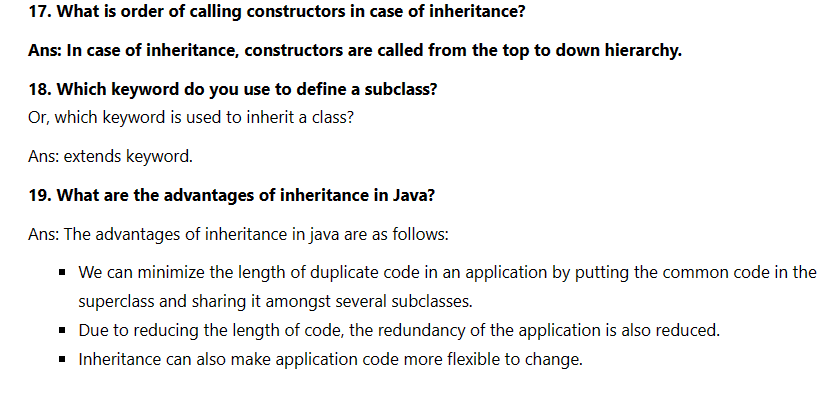
If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

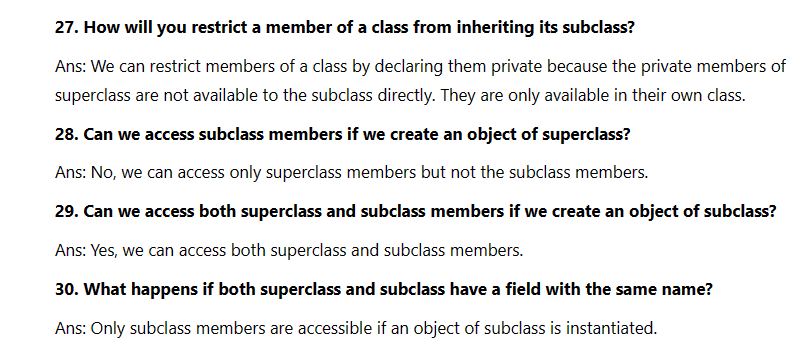


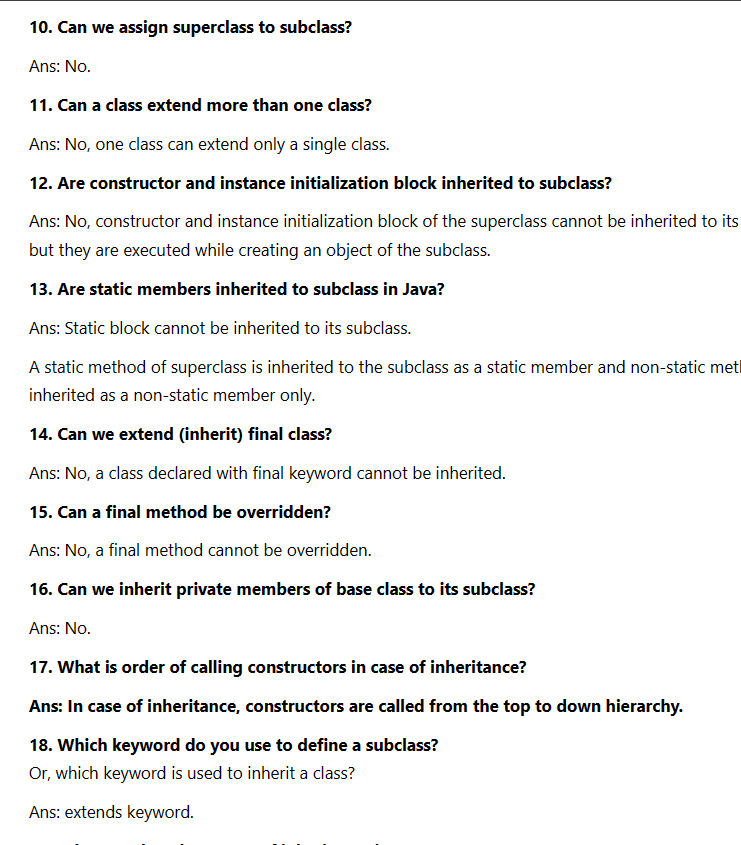


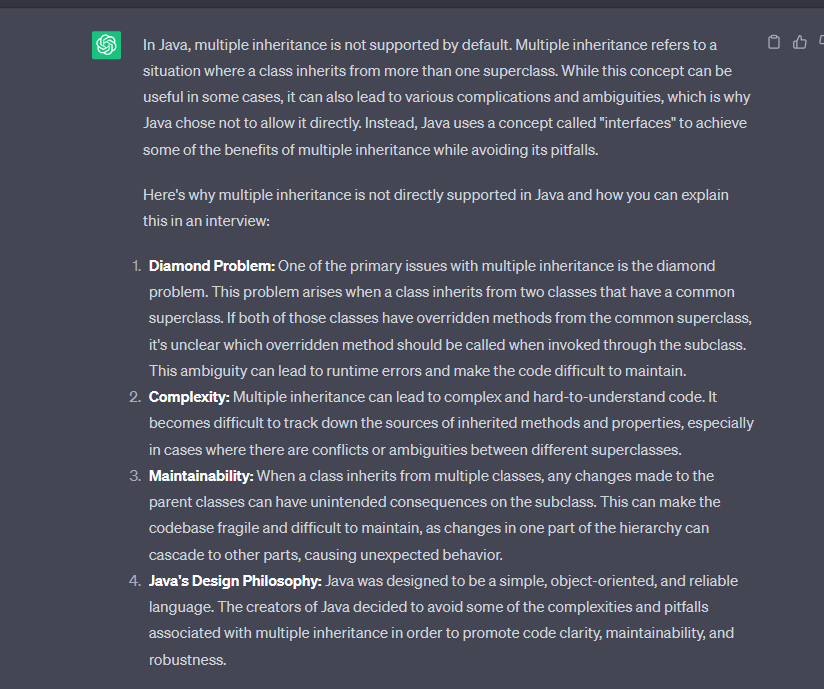


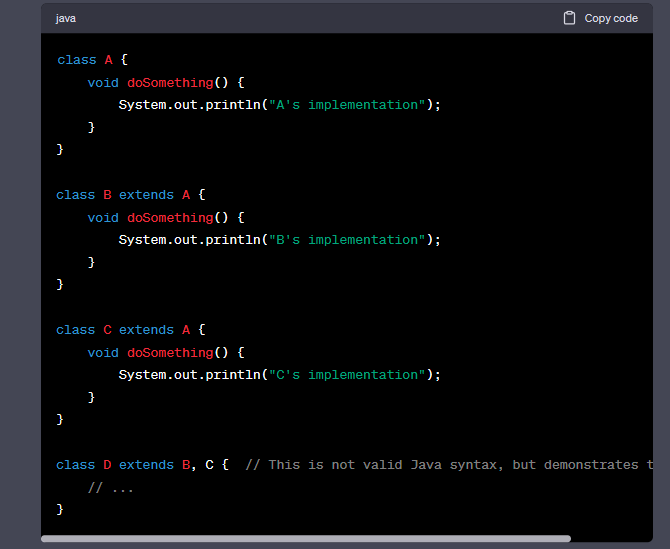


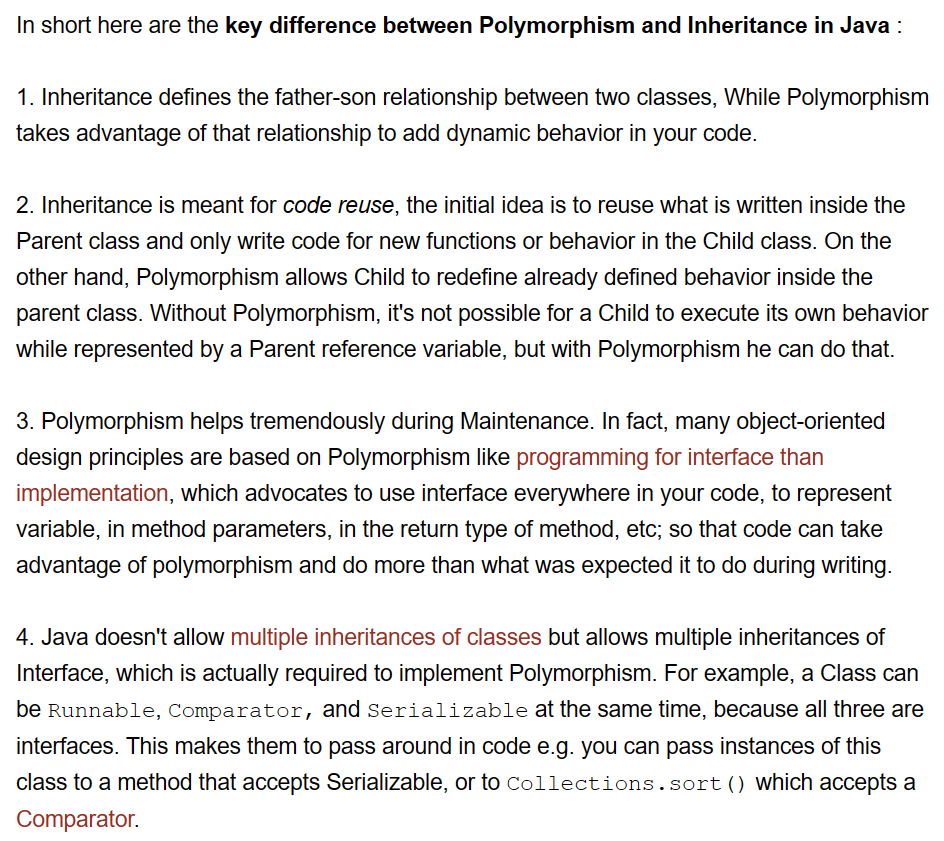
`











# Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

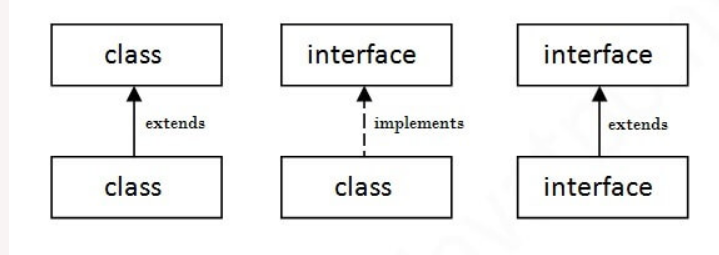
In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

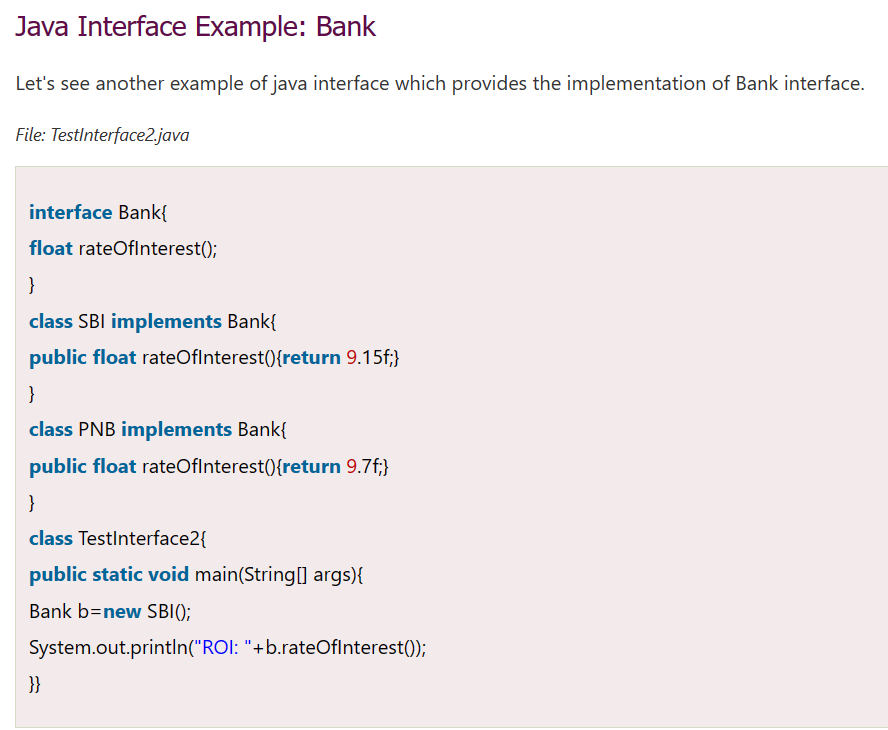
Java Interface also **represents the IS-A relationship**.

## **Why use Java interface?**

There are mainly three reasons to use interface. They are given below.

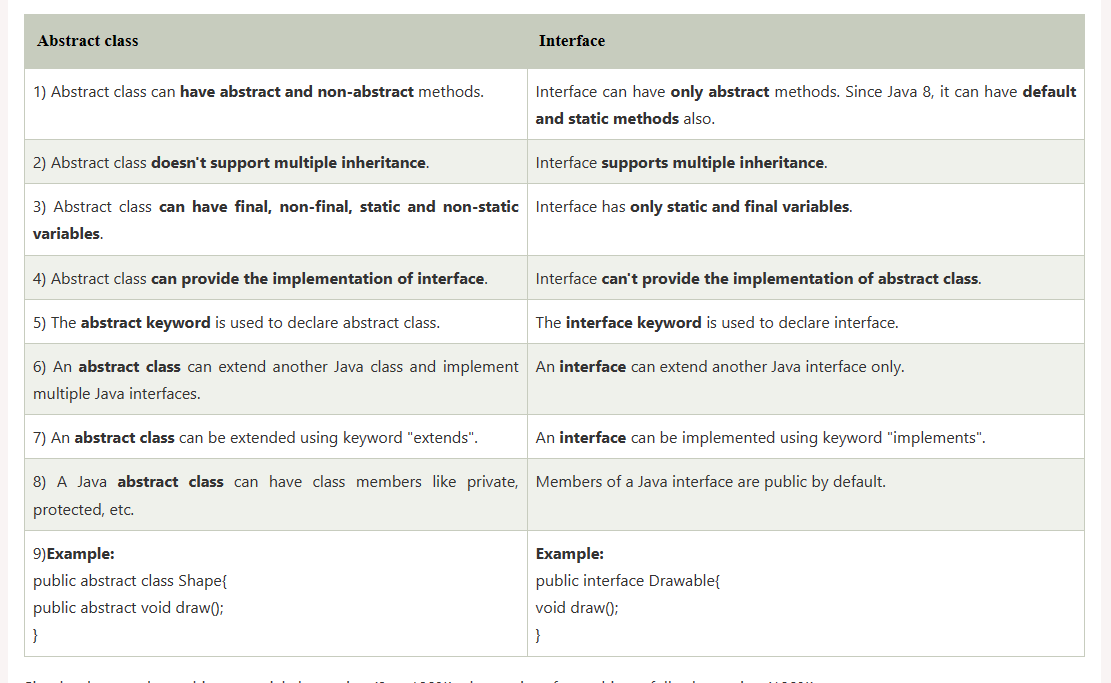
* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.

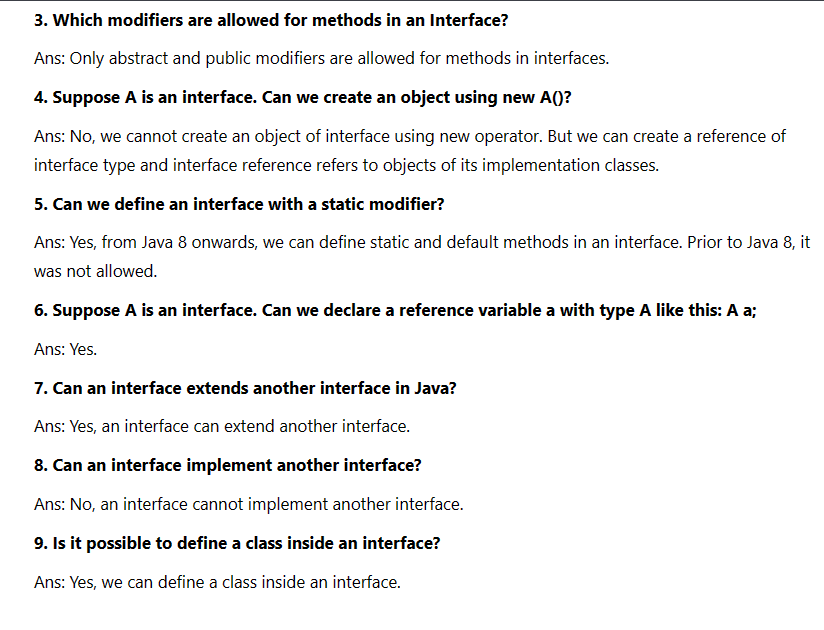


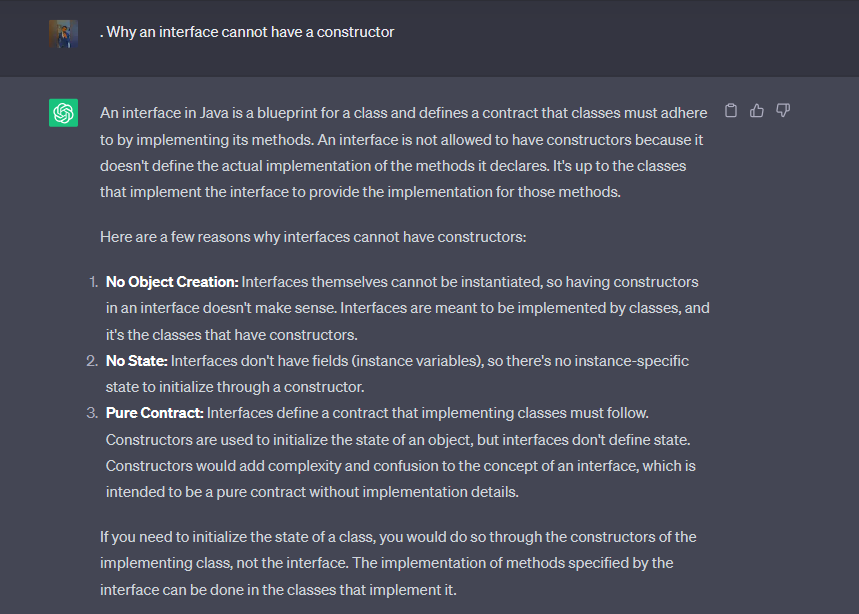


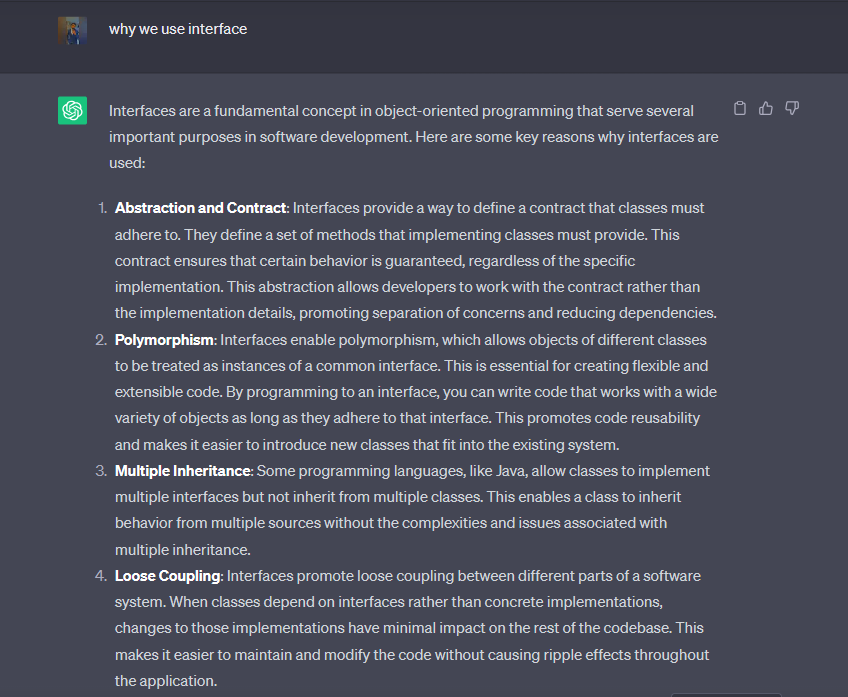
***Using Interface***

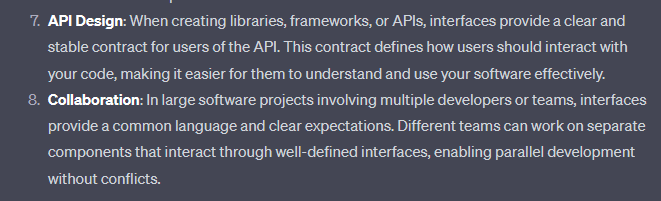












# Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

**Package** In Java, a package is a group of related classes, interfaces, and sub-packages. Packages help developers organize their code and make it easier to maintain and reuse in multiple projects. As such, Packages Interview Questions are a critical part of Java developer interviews.

## **Advantage of Java Package**

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

## **How to access package from another package?**

There are three ways to access the package from outside the package.

1. import package.\*;
2. import package.classname;
3. fully qualified name.

#### **1) Using packagename.\***

If you use package.\* then all the classes and interfaces of this package will be accessible but not subpackages.

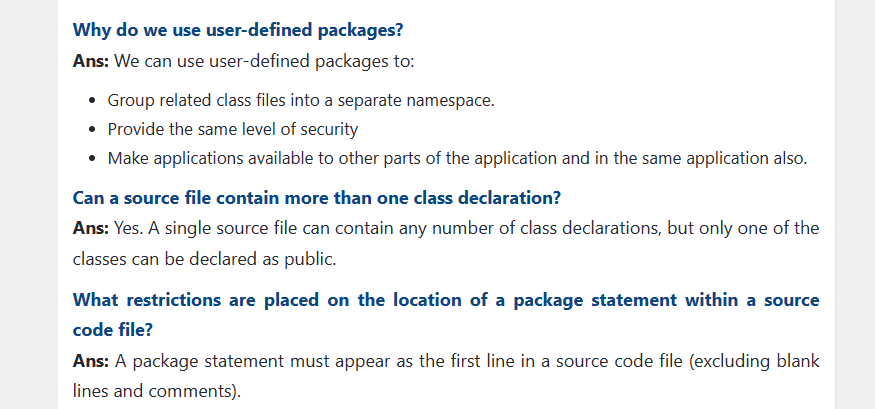
#### **2)Using packagename.classname**

If you import package.classname then only declared class of this package will be accessible.

## **Subpackage in java**

Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.





# Polymorphism in Java

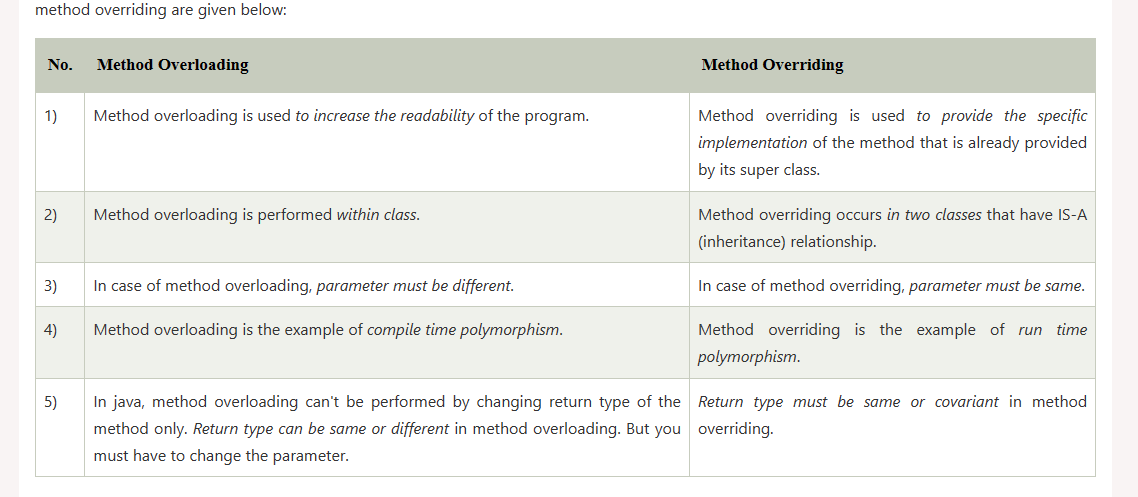
**Polymorphism in Java** is a concept by ***which we can perform a*single action in different ways*.*** Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. ***So polymorphism means many forms.***

***There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.*** We can perform polymorphism in java by method overloading and method overriding.

***If you overload a static method in Java, it is the example of compile time polymorphism***

### **Why can we not override static method?**

**It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.**



We can achieve polymorphism in Java using the following ways:

1. [Method Overriding](https://www.programiz.com/java-programming/method-overriding)
2. [Method Overloading](https://www.programiz.com/java-programming/method-overloading)
3. Operator Overloading (in c++)

## Java Method Overriding

During [inheritance in Java](https://www.programiz.com/java-programming/inheritance), if the same method is present in both the superclass and the subclass. Then, the method in the subclass overrides the same method in the superclass. This is called method overriding.

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

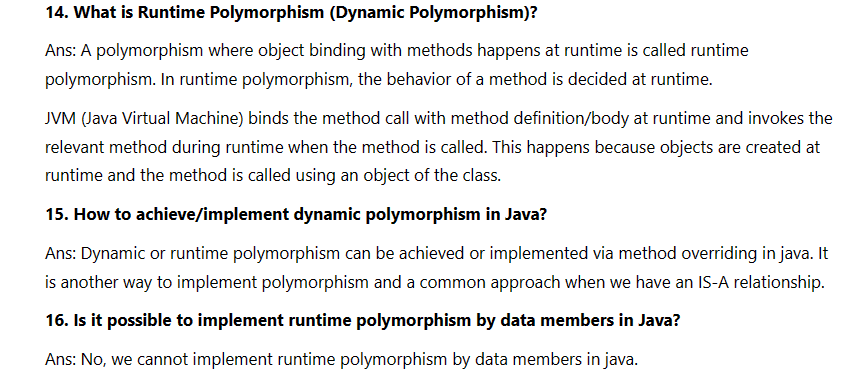
In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

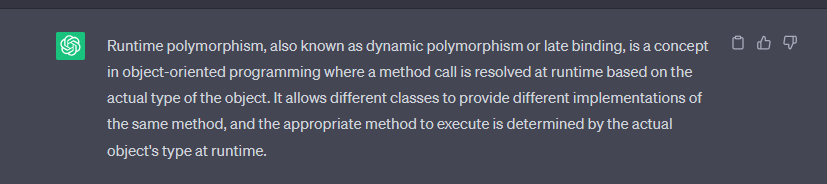
* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* **Method overriding is used for runtime polymorphism**

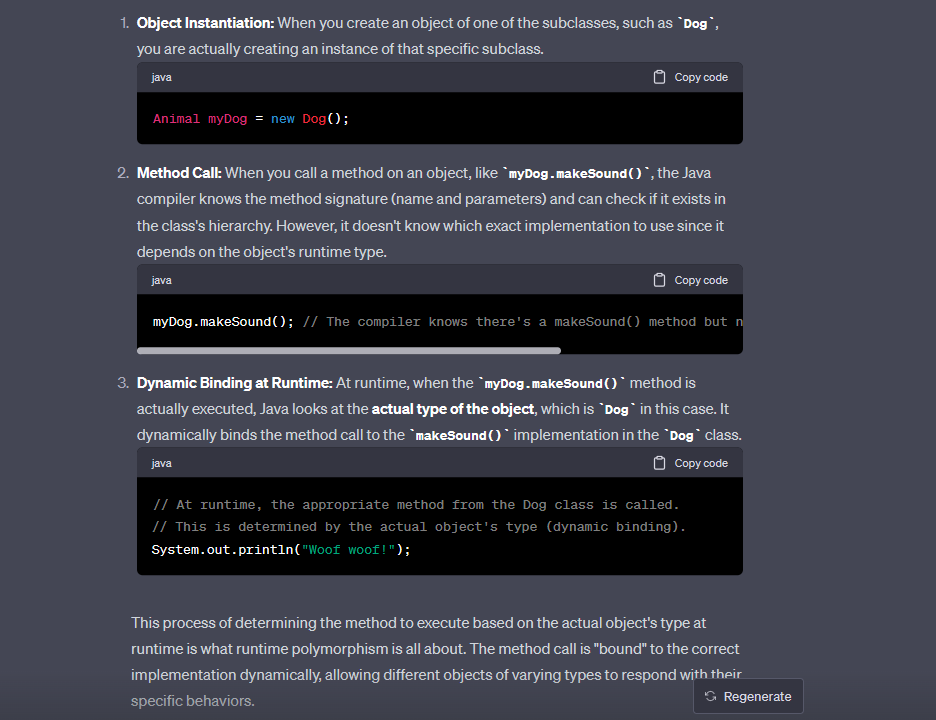
#### **Rules for Java Method Overriding**

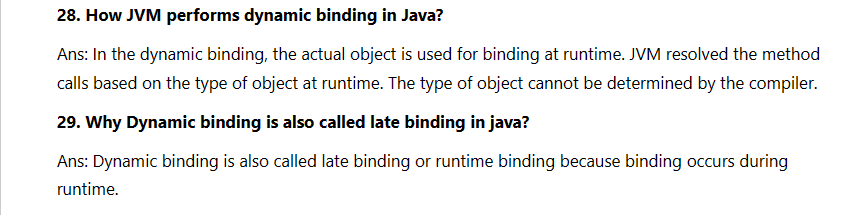
1. The method must have the **same name as in the parent class**
2. The method must have **the same parameter as in the parent class.**
3. **There must be an IS-A relationship (inheritance).**

**Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.



****

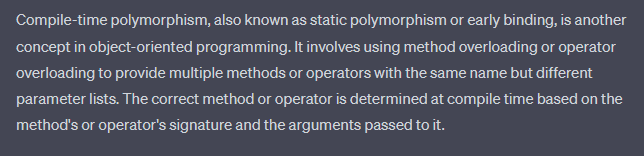
****

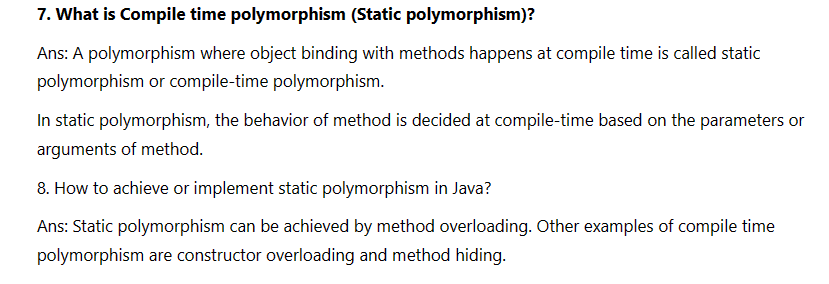
****

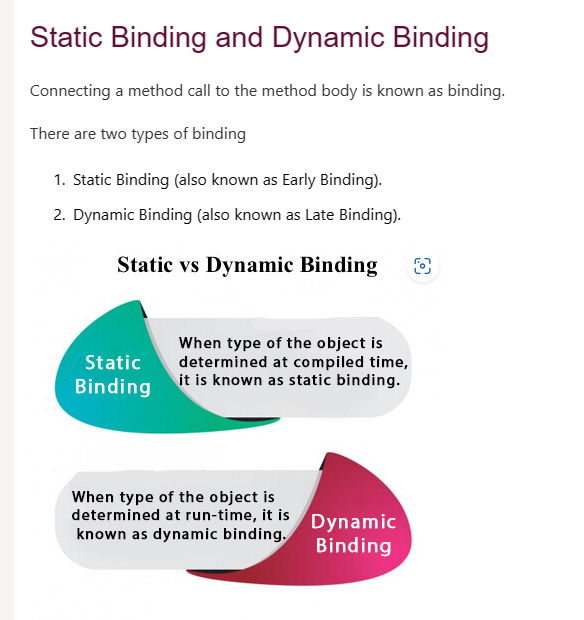
**Real World Example :**

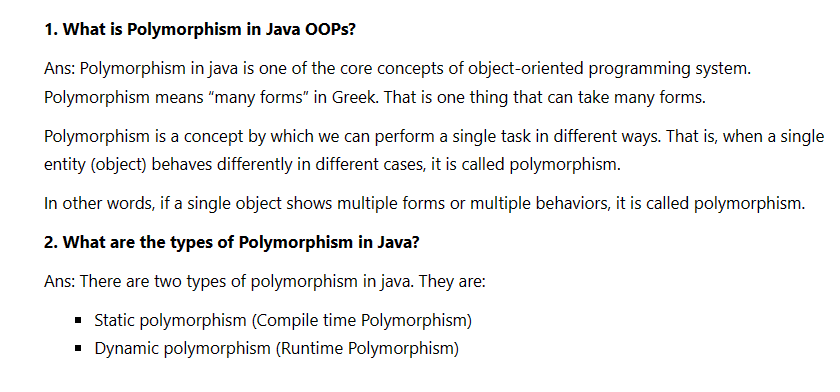


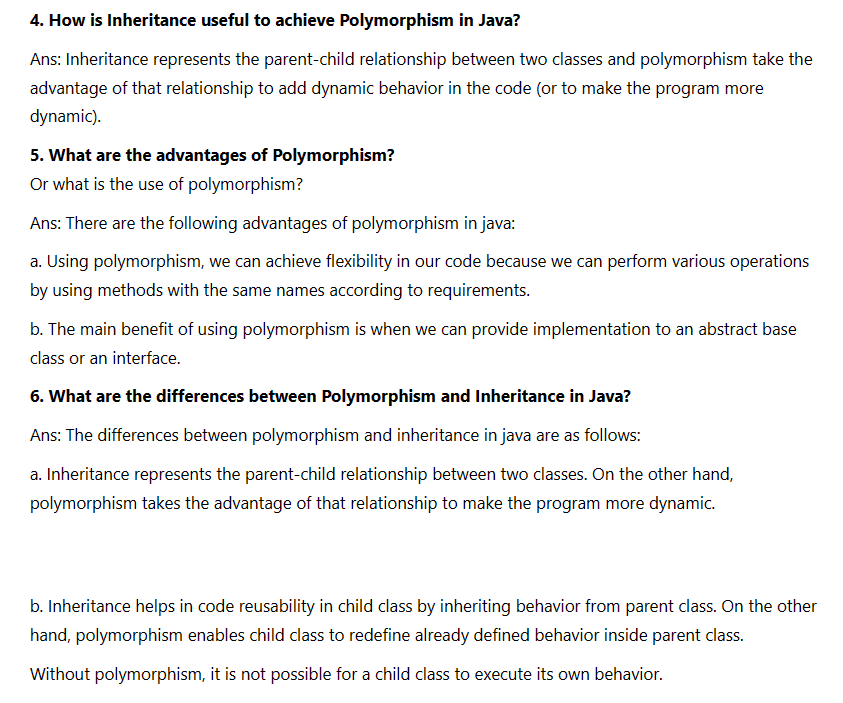
**Compile Time :**

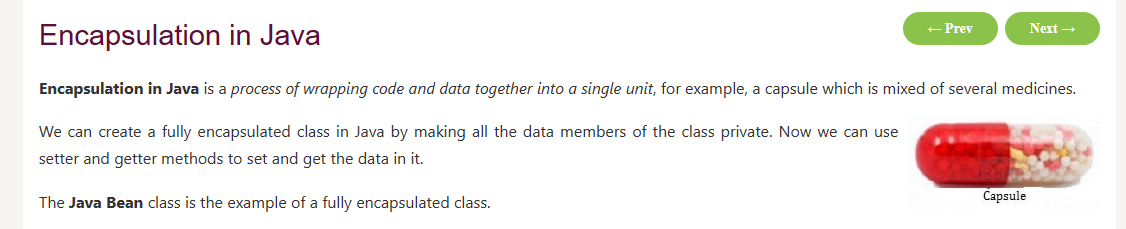












### **Advantage of Encapsulation in Java**

By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.

It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

The encapsulate class is **easy to test**. So, it is better for unit testing.

The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

