

Java OOPs Concepts

Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

- **Object**

Any entity that has state and behaviour is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

- **Class**

Collection of objects is called class. It is a logical entity.

1. Inheritance

When one object acquires all the properties and behaviours of a parent object, it is known as inheritance. It provides code reusability.

2. Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

3. Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

4. Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other.

Cohesion

Cohesion refers to the level of a component which performs a single well- defined task. A single well-defined task is done by a highly cohesive method.

Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects.

Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state.

Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state.

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

- **Constructor Overloading in Java**

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

- **Private Constructor in Java**

Java allows us to declare a constructor as private. We can declare a constructor private by using the **private** access specifier. Note that if a constructor is declared private, we are not able to create an object of the class. Instead, we can use this private constructor in **Singleton Design Pattern**.

Rules for Private Constructor

The following rules keep in mind while dealing with private constructors.

- It does not allow a class to be sub-classed.
- It does not allow to create an object outside the class.
- If a class has a private constructor and when we try to extend the class, a compile-time error occurs.
- We cannot access a private constructor from any other class.
- If all the constant methods are there in our class, we can use a private constructor.

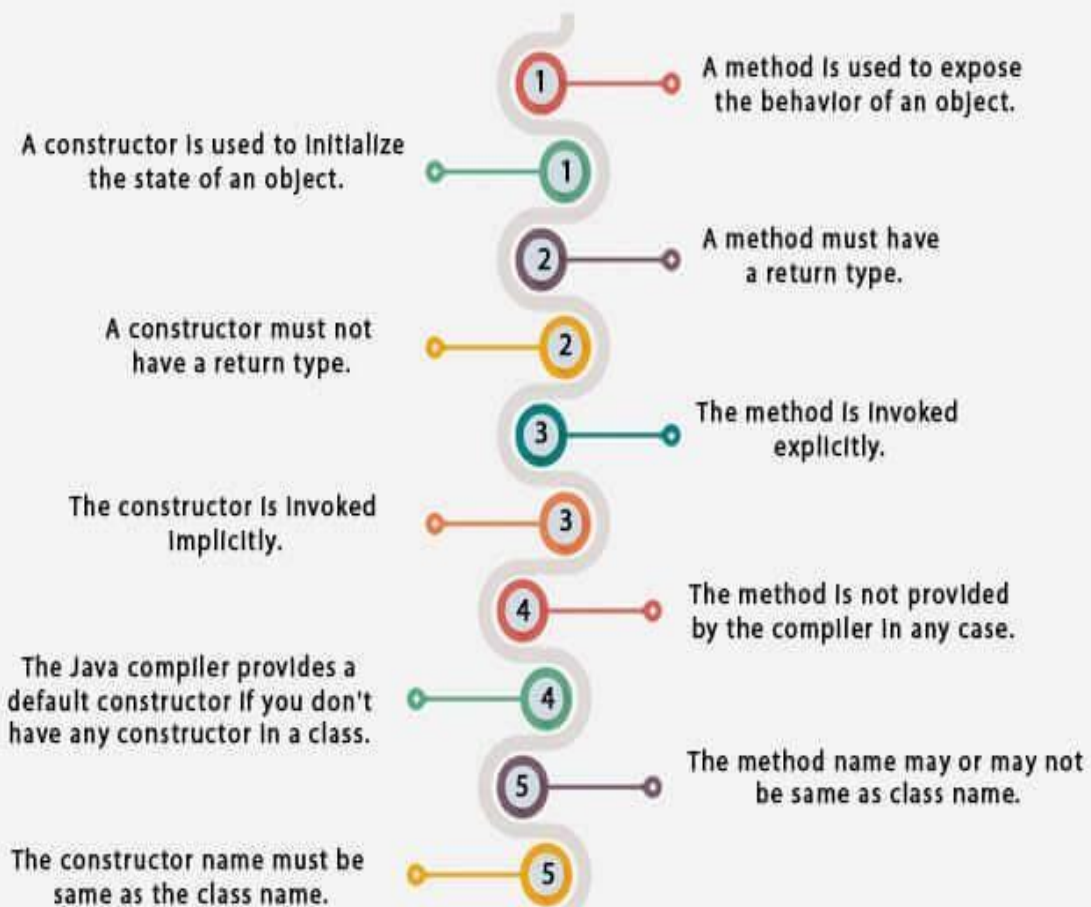
Use Cases of Private Constructor

The main purpose of using a private constructor is **to restrict object creation**. We also use private constructors to implement the singleton design pattern. The use-cases of the private constructor are as follows:

- It can be used with static members-only classes.
- It can be used with static utility or constant classes.
- It can also be used to create singleton classes.
- It can be used to assign a name, for instance, creation by utilizing factory methods.
- It is also used to avoid sub-classing.

Different b/w Constructor and Method

Difference between constructor and method in Java



Java Inheritance

Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object.

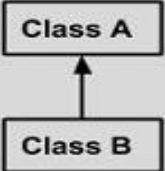
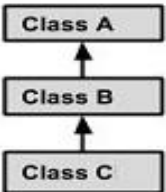
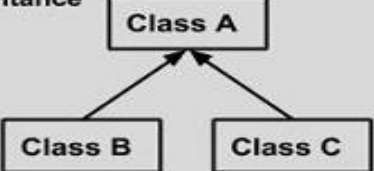
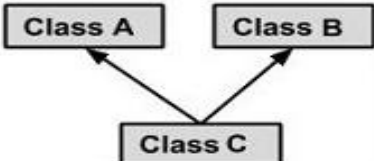
Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

| | |
|---|--|
| Single Inheritance  <pre>graph BT; B[Class B] --> A[Class A]</pre> | <pre>public class A { } public class B extends A { }</pre> |
| Multi Level Inheritance  <pre>graph BT; C[Class C] --> B[Class B]; B --> A[Class A]</pre> | <pre>public class A {} public class B extends A {.....} public class C extends B {.....}</pre> |
| Hierarchical Inheritance  <pre>graph BT; B[Class B] --> A[Class A]; C[Class C] --> A</pre> | <pre>public class A {} public class B extends A {.....} public class C extends A {.....}</pre> |
| Multiple Inheritance  <pre>graph BT; A[Class A] --> C[Class C]; B[Class B] --> C</pre> | <pre>public class A {} public class B {.....} public class C extends A,B { } // Java does not support multiple Inheritance</pre> |

1) Single Inheritance

When a class inherits another class, it is known as a single inheritance.

2) Multilevel Inheritance Example

When there is a chain of inheritance, it is known as multilevel inheritance.

3) Hierarchical Inheritance Example

When two or more classes inherit a single class, it is known as hierarchical inheritance.

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

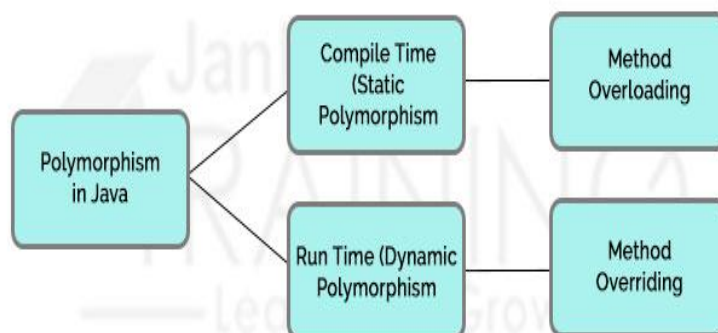
Aggregation in Java

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

Method Overloading in Java

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.

Java Polymorphism



Method Overriding in Java

- 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.

Polymorphism in Java

- Polymorphism in Java is a concept by which we can perform a single action in different ways.
- 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.

Runtime Polymorphism in Java

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.

Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding. There are two types of binding

1. Static Binding (also known as Early Binding).
2. Dynamic Binding (also known as Late Binding).

static binding

When type of the object is determined at compiled time (by the compiler), it is known as static binding.

Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

Java instanceof

- The **Java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).
- The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false.

Java Abstraction

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Abstraction in Java

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

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

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

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

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. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

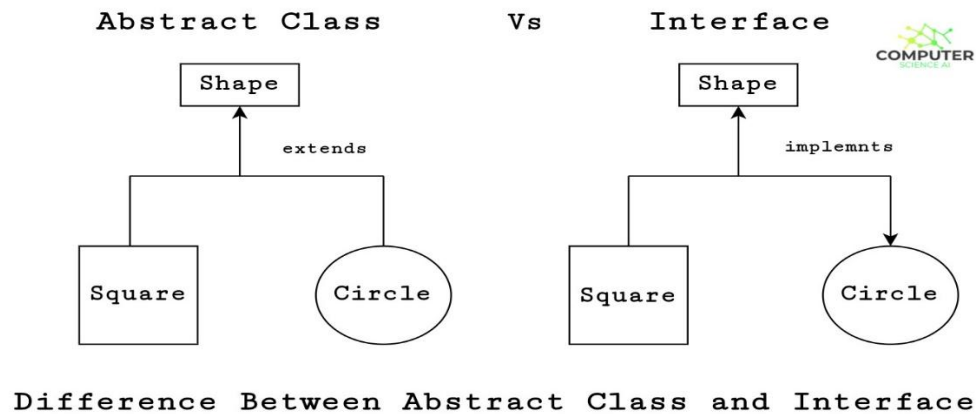
There are mainly three reasons to use interface. They are given below.

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

Syntax:

```
interface <interface_name> {
```

```
// declare constant fields
// declare methods that abstract
// by default.
}
```



Difference between abstract class and interface

| Abstract class | Interface |
|--|--|
| 1) Abstract class can have abstract and non-abstract methods. | Interface can have only abstract methods. Since Java 8, it can have default and static methods also. |
| 2) Abstract class doesn't support multiple inheritance. | Interface supports multiple inheritance. |
| 3) Abstract class can have final, non-final, static and non-static variables. | Interface has only static and final variables. |
| 4) Abstract class can provide the implementation of interface. | Interface can't provide the implementation of abstract class. |
| 5) The abstract keyword is used to declare abstract class. | The interface keyword is used to declare interface. |
| 6) An abstract class can extend another Java class and implement multiple Java interfaces. | An interface can extend another Java interface only. |

| | |
|---|--|
| 7) An abstract class can be extended using keyword "extends". | An interface can be implemented using keyword "implements". |
| 8) A Java abstract class can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)Example: <pre>public abstract class Shape{ abstract void draw(); }</pre> | Example: <pre>public interface Drawable{ void draw(); }</pre> |

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Java Encapsulation

Encapsulation in Java

Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

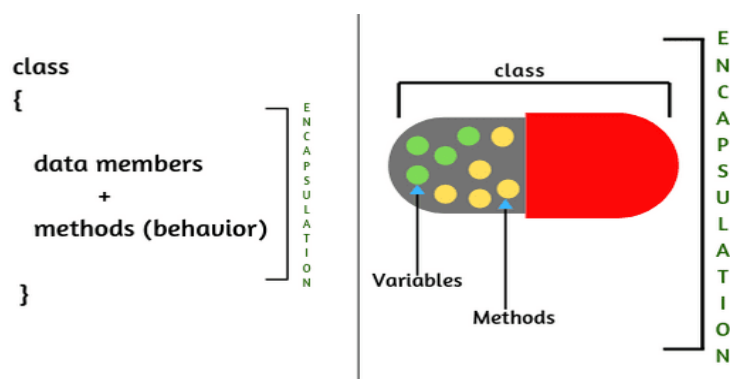


Fig: Encapsulation

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.
- 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.

Difference between object and class

| No. | Object | Class |
|-----|---|--|
| 1) | Object is an instance of a class. | Class is a blueprint or template from which objects are created. |
| 2) | Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc. | Class is a group of similar objects. |
| 3) | Object is a physical entity. | Class is a logical entity. |
| 4) | Object is created through new keyword mainly e.g. Student s1=new Student(); | Class is declared using class keyword e.g. class Student{} |
| 5) | Object is created many times as per requirement. | Class is declared once. |
| 6) | Object allocates memory when it is created. | Class doesn't allocate memory when it is created. |
| 7) | There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization. | There is only one way to define class in java using class keyword. |

Difference between method overloading and method overriding in java

| No. | Method Overloading | Method Overriding |
|-----|--|---|
| 1) | Method overloading is used to increase the readability of the program. | Method overriding is used to provide the specific implementation of the method that is already provided by its super class. |

| | | |
|----|---|--|
| 2) | Method overloading is performed within class. | Method overriding occurs in two classes that have IS-A (inheritance) relationship. |
| 3) | In case of method overloading, parameter must be different. | In case of method overriding, parameter must be same. |
| 4) | Method overloading is the example of compile time polymorphism. | Method overriding is the example of run time polymorphism. |
| 5) | In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter. | Return type must be same or covariant in method overriding. |





