
Java, Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

Why Is Java Inheritance Important?

- Implements is-a relationship: It represents a hierarchical relationship between classes. For example, a SalesEmp is-a Employee.
- Code Reusability: The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
- Polymorphism via Method Overriding: Inheritance enables runtime polymorphism, where subclasses can override methods from the superclass to provide specific behavior.
- Abstract Classes and Interfaces: Inheritance facilitates abstraction. Using abstract classes and interfaces, you can define behavior in a superclass without specifying the implementation, leaving it to the subclasses.

Terminologies Used in Java Inheritance

- Class: Class is a set of objects which shares common characteristics/ behavior and common properties/ attributes. Class is not a real-world entity. It is just a

template or blueprint or prototype from which objects are created.

- Super Class/Parent Class: The class whose features are inherited is known as a superclass(or a base class or a parent class).
- Sub Class/Child Class: The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- Reusability: Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

How to Use Inheritance in Java?

The extends keyword is used for inheritance in Java. Using the extends keyword indicates you are derived from an existing class. In other words, "extends" refers to increased functionality.

Syntax :

```
class derived-class extends base-class  {
//methods and fields
}
```

Example of Inheritance in Java

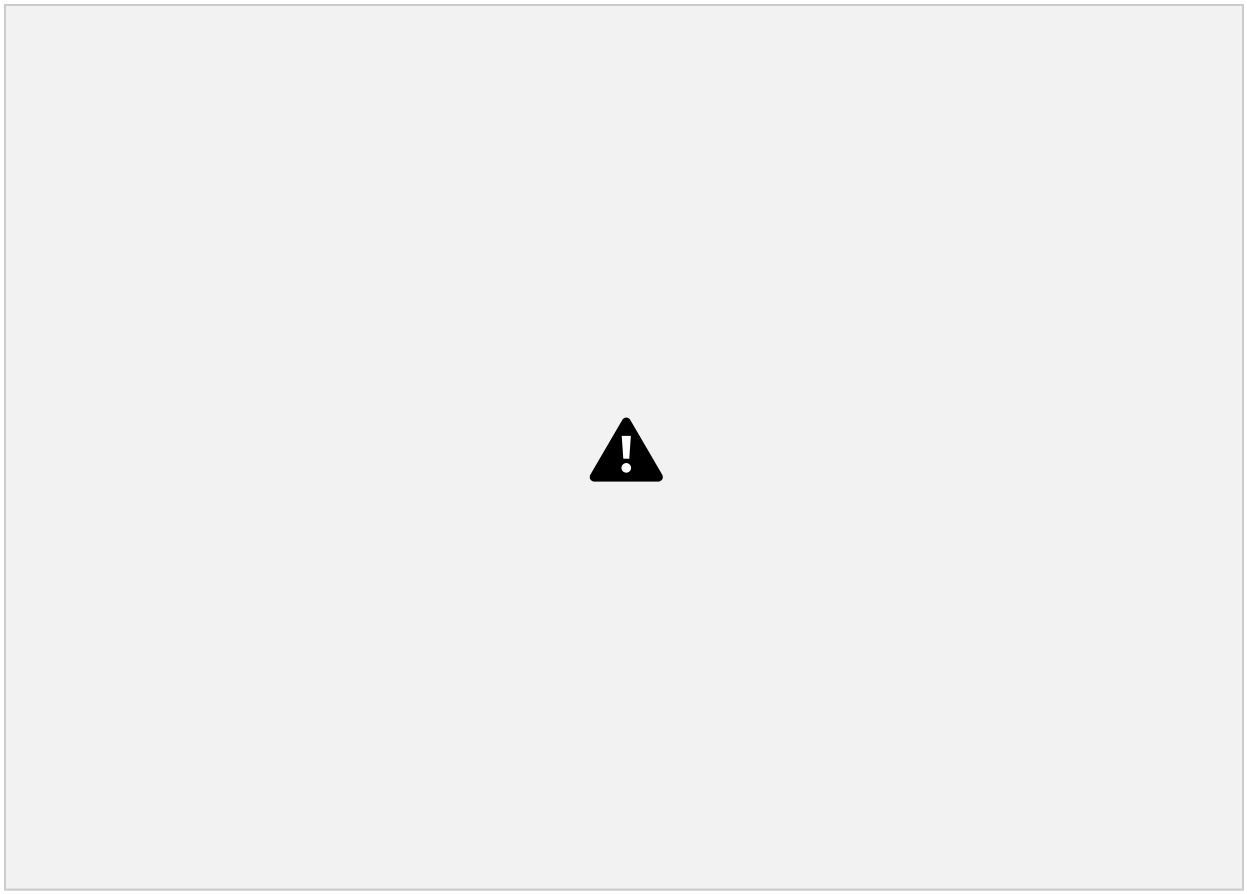
```
// Parent Class
class Employee {
    int id;
```

```
    int salary;

    Employee(int i, int s) {
        id = i;
        salary = s;
    }
}

// Child Class
class SalesEmp extends Employee {
    int salesIncentive;

    SalesEmp(int i, int s, int si) {
        super(i, s);
        salesIncentive = si;
    }
}


public class Test {
    public static void main(String[] args) {
        SalesEmp se = new SalesEmp(101, 30000, 10000);

        System.out.println("Salary: " + se.salary);
        System.out.println("ID: " + se.id);
        System.out.println("Sales Incentive: " + se.salesIncentive);
    }
}
```

Output:

```
Salary: 30000
ID: 101
Sales Incentive: 10000
```

Examples of Inheritance in Real Scenarios



1. Employee Inheritance

Employee

- SalesEmp (Handles sales-related tasks and incentives)
- HREmp (Handles human resource activities)
- MarketingEmp (Handles marketing responsibilities)
- SDEEmp (Software Development Engineer managing coding tasks)

2. Educational Inheritance:

- Person
    - Student
        - EngineeringStudent (Specializes in engineering disciplines)

        - ManagementStudent (Specializes in management studies)
    - Faculty
        - Professor (Senior faculty responsible for lectures and research)
        - AssistantProfessor (Junior faculty assisting in academic activities)
        - AssociateProfessor (Mid-level faculty with teaching and research roles)

Inheritance Hierarchy in Java

In Java, the `Object` class is the root of all classes. Even if a class doesn't explicitly extend another class, it implicitly extends `Object`. The `Object` class provides fundamental methods that are inherited by all classes.

1. `clone()`:

- Creates a copy of the current object.
- Useful for duplicating objects while maintaining their state.

2. `equals()`:

- Compares two objects for equality.
- By default, it compares memory addresses, but it can be overridden to compare object content.

3. `hashCode()`:

- Returns a unique integer representing the object.

- Typically used in collections like HashMap and HashSet for efficient lookup.

4. toString():

- Returns a string representation of the object.
- By default, it includes the class name and memory address, but it can be overridden for meaningful output.

Inheritance in Java is a powerful tool that enhances code maintainability and structure, enabling developers to model real-world hierarchies efficiently.