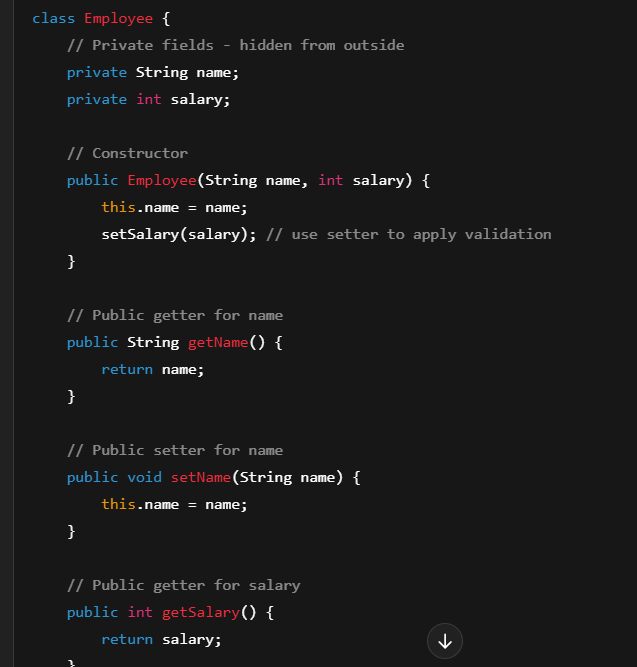
**Encapsulation**:   
Encapsulation is the process of wrapping data (variables) and the methods that operate on the data into a single unit, i.e., a class. In Java, this is achieved using **access modifiers** like private, protected, and public to control access to the fields of a class. The idea is to restrict direct access to some of the object's components, which can help prevent accidental modification of data.

\*\*\* private instance variables

Getter and setter methods

Constructors



A screen shot of a computer screen

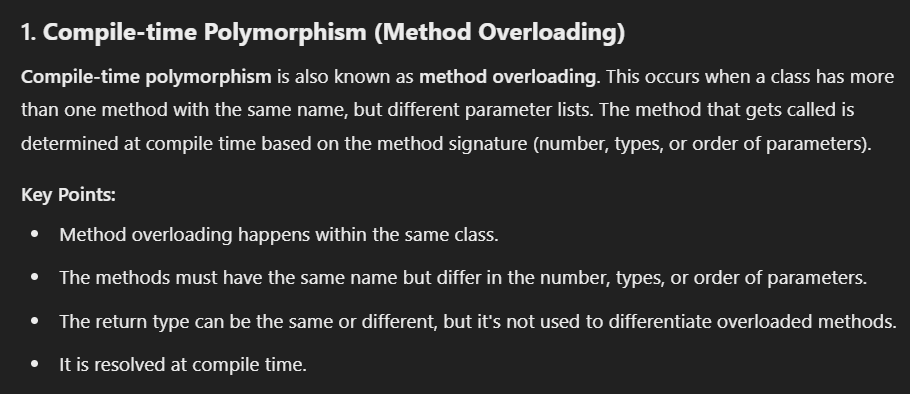
AI-generated content may be incorrect.

**Abstraction**:  
Abstraction involves hiding complex implementation details and showing only the essential features of an object. In Java, abstraction is achieved through **abstract classes** and **interfaces**. An abstract class can have abstract methods (without body), and the subclasses must implement those methods. An interface can have only method signatures (with no body), and the implementing class must provide the body of those methods.

**Inheritance**:  
Inheritance allows a new class to inherit the properties and behaviours (fields and methods) of an existing class. This promotes code reuse and creates a relationship between the parent (superclass) and child (subclass) classes. In Java, inheritance is achieved using the extends keyword.

**Polymorphism**:  
Polymorphism allows objects of different classes to be treated as objects of a common superclass. It also allows one method or operator to have different behaviours based on the object that invokes it. There are two types of polymorphism in Java:

* **Compile-time polymorphism (Method Overloading)**: This is achieved by having multiple methods with the same name but different parameter lists.
* **Runtime polymorphism (Method Overriding)**: This is achieved when a subclass provides a specific implementation of a method that is already defined in its superclass.

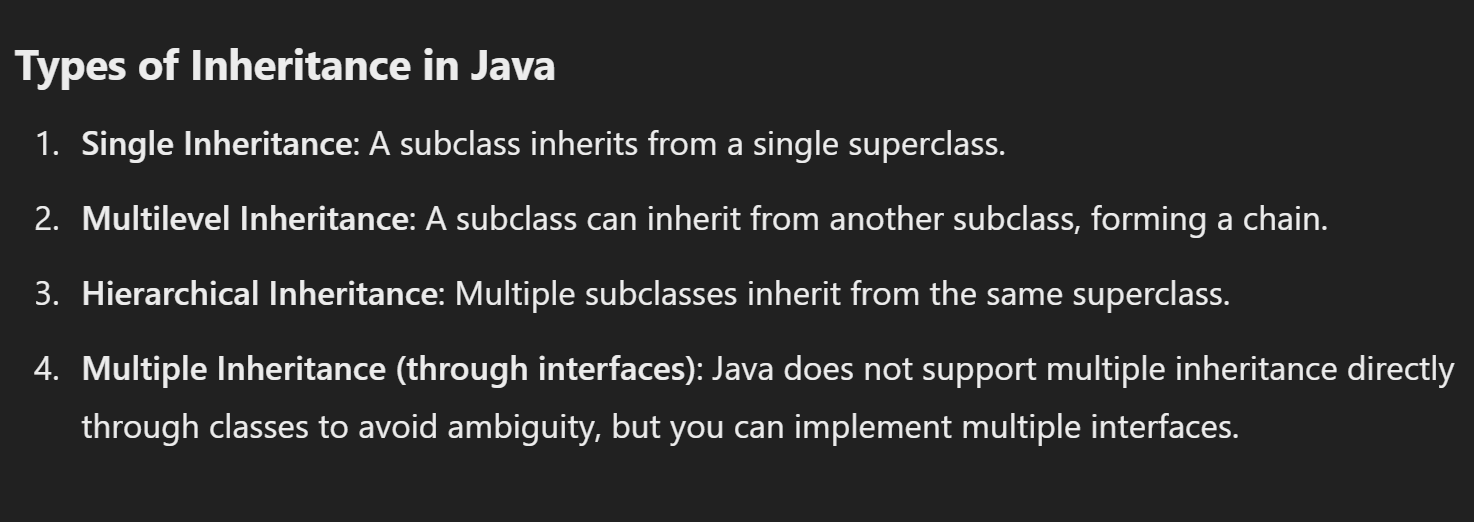


A black and white text on a black background

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.



A screenshot of a black screen

AI-generated content may be incorrect.

**Class**

* A **class** is a blueprint or template for creating objects (instances). It defines properties (also called fields or variables) and behaviors (methods or functions) that the objects created from the class will have.
* A class doesn't represent real data, but it provides the structure and functionality that objects will use.

**Key points about a class:**

* + **Fields (Variables)**: These define the attributes or properties of the class.
  + **Methods (Functions)**: These define the actions or behaviours that the class can perform.
  + A class can be thought of as a **blueprint** for objects.

An **object** is an instance of a class. Once a class is defined, you can create multiple objects from that class. Each object will have its own unique set of properties, but they will all share the same behaviours defined in the class.

In Java, method overloading is based on the method name and the parameter list — **not the return type**. You cannot overload methods by changing only the return type.

In Overriding

1. The same name
2. The same parameter list
3. And a compatible return type.