Lec 23 Inheritance:

Inheritance in Java is a way for one class to use the properties and methods of another class without rewriting code. This helps save time, reduces errors, and makes programs easier to manage by organizing code into parent and child relationships.

If a class has no code(empty) and we make its object in main method. We still have 9 inherited methods from OBJECT class

- equals(Object obj) → boolean
- getClass() → Class<?
- hashCode() → int
- notify() → void
- notifyAll() → void
- toString() → String
- wait() → void
- wait(long timeout) → void
- wait(long timeout, int nanos) → void

Every class inherits OBJECT class

Introduction to OOP and Inheritance

Object-Oriented Programming (OOP) includes four main ideas: encapsulation, inheritance, polymorphism, and abstraction. Inheritance allows a class (child) to acquire properties and methods from another class (parent), similar to how traits are passed from parents to children in real life.

Real-World Analogy of Inheritance

Inheritance is compared to how children inherit features from their parents, like property or traits. In programming, this means a child class gets the attributes and behaviors of a parent class automatically.

Code Example: Car and BMW Classes

If a class Car has methods like mileage, speed, and engine RPM, a new class BMW can inherit these features without rewriting them. The BMW class can also add its own features, like an AI system, by extending the Car class.

Parent-Child Relationship in Inheritance

The parent (superclass) contains shared code, while the child (subclass) inherits and can add or change features. This relationship is essential for inheritance to work; only child classes can inherit from parent classes, not unrelated classes.

Benefits of Inheritance

Inheritance reduces code repetition, saves development time, and increases productivity by allowing new classes to use existing code. This makes the process more efficient and profitable for companies.

Is-A Relationship Explained

Inheritance creates an "is-a" relationship, meaning the child class is a type of the parent class (e.g., BMW is a Car). This allows the child to be treated as an instance of the parent.

Using the `extends` Keyword

In Java, inheritance is implemented using the extends keyword. For example, class BMW extends Car means BMW inherits all public and protected members of Car.

Inheriting and Overriding Methods

Methods in the parent class become available in the child class. If the child class needs a method to work differently, it can override the method by redefining it. The @Override annotation helps the compiler catch mistakes in overriding.

Specialized Methods in Child Classes

Child classes can also have specialized methods that are not present in the parent class. These methods are unique to the child and are not inherited.

Types of Inheritance

Java supports several types of inheritance: - Single Inheritance: One child inherits from one parent.

Multilevel Inheritance: A chain of inheritance (A → B → C).

Hierarchical Inheritance: Multiple children inherit from one parent.

Hybrid Inheritance: A mix of two or more types (not directly supported with classes, but possible with interfaces).

Why Multiple Inheritance is Not Allowed in Java Classes

Java does not allow one class to inherit from more than one class (multiple inheritance) because it can cause confusion if both parents have the same method names. This leads to the "diamond problem," where the compiler cannot decide which method to use.

Suppose 2 class (A and B) have same name method then the c class inheriting both will get confused for whom to call (ambiguity).

Diamond Problem Explained

The diamond problem occurs when two parent classes have a method with the same name, and a child class inherits from both. The compiler can't decide which method to use, so Java avoids this by not allowing multiple inheritance with classes.

Object Class and Universal Inheritance

Every class in Java implicitly inherits from the Object class, which provides basic methods like toString, equals, and hashCode. This allows all Java objects to have these fundamental behaviors.

Practice and Interview Tips

Practicing different types of inheritance and understanding their differences is important for interviews. Keeping good notes and practicing code examples, especially hybrid inheritance, helps in mastering the topic.

In java if B class is inheriting A class (which in default inherits OBJECT class).

If we create B class’s object then first the object creation starts from top level hierarchy(Object → class A → class B).

Super() → calls parents class constructor to ready the parent

class so that it can be used by child class.

Child class se parent class ka constructor call hona chahiye.

This lesson explains how inheritance works in Java, focusing on how classes, methods, and constructors interact when one class extends another. It highlights rules for constructor chaining, method overriding, and the use of the final and super keywords, making it clear how inheritance helps reuse code and enforce behaviors in object-oriented programming.

What is Inheritance and Why Use It?

Inheritance allows a new class to use the properties and methods of an existing class, reducing code repetition and making it easier to build on existing functionality. If a class already exists with similar features, you can extend it instead of starting from scratch.

Calling Methods With and Without Inheritance

To call a non-static method in Java, you need to create an object of the class. However, if you create an object of a subclass, you also get access to the parent class's methods due to inheritance, even if you didn't explicitly create a parent object.

The Instantiation Order in Inheritance

When you create an object of a subclass, Java first creates objects of all parent classes in the hierarchy, starting from the top (like the Object class) down to your subclass. This ensures all inherited features are ready before the child's features are initialized.

Constructor Chaining and the `super` Keyword

The super keyword is used in a subclass constructor to call the constructor of its parent class. If you don't explicitly call super, Java inserts it automatically, calling the parent's no-argument constructor. Constructor chaining ensures that all necessary initializations happen in the correct order.

DNA Analogy and Constructor Matching

For inheritance to work smoothly, the child class's constructor must match a constructor in the parent class, similar to how DNA matches between parent and child. If the parent lacks a suitable constructor, inheritance will fail and cause errors.

Rules for Using `super` and `this` in Constructors

Both super (for parent class) and this (for current class) must be the first statement in a constructor. You cannot use both together in the same constructor. super is used to ensure the parent's constructor runs before initializing the child.

Calling Methods from Constructors and Inheritance

Methods inherited from a parent can be called directly from the child class, including from within constructors. However, super is only for calling constructors, not methods.

Method Overriding and Access Modifiers

Public and default (package-private) methods in the parent class can be overridden in the child class.

Private methods are not inherited and cannot be overridden in child class. The final keyword prevents a method from being overridden, allowing the child to use it but not change its behavior.

Final Classes and Methods

Marking a class as final prevents any other class from extending it. Marking a method as final prevents it from being overridden. This is used to enforce security, stability, or to prevent further changes to critical logic.

The `final` Keyword with Variables

A final variable cannot be changed after it is assigned. For local variables, the value must be set once and cannot change. For instance variables, the value can only be set in the constructor and not through regular methods, ensuring the variable stays constant throughout the object's life.

- Final keyword on Class:

  This class will not be extended by any other class

- Final keyword on Local Variable:

  This variable will not be re-initialised (pi).

- Final keyword on Instance Variable:

This variable should only be initialised with constructor as it is going to be final. If any method of this same class will try to change it → it will give compile time error. That's why it should be initialised with a constructor.

- public methods of class A can be inherited and overridden in class B(extends class A).
- private methods of class A will not be inherited and overridden in class B(extends class A).
- final methods of class A can be inherited but can't be overridden in class B(extends class A).

Key Takeaways on Constructor Inheritance

For inheritance to be valid, the child's constructor must be able to call a suitable parent constructor (matching "DNA"). Parent classes are always initialized before child classes. Understanding this prepares you for more complex topics like polymorphism, method overloading, and overriding.