Lec 24 Polymorphism:

This lesson covers the concepts of inheritance and polymorphism in object-oriented programming, focusing on how method overloading and overriding help reduce code duplication and make code more flexible. It explains the difference between compile-time and run-time polymorphism and demonstrates how these concepts can simplify and organize code, especially when working with related classes.

Car class →

- BMW extends Car class
- Thar extends Car class
- Alto extends Car class
- Safari extends Car class

If all these 4 car doesn't extends the Car class then we have to make an object of every car (4 objects → 4 lines) and will have to call (suppose 5 methods → 4 x 5 = 20 lines) 20 times.

Now if all class inherits 5 methods from Car class then, we will not call 5 methods every time.

We will make a separate class containing all 5 methods which will be called upon passing object of the intended class.

In this separate class we will accept Car Class’s object but we will send object of (BMW/Thar/Alto/Safari).

Here, Upcasting will work. Parent class reference can hold child class’s reference.

Line of code will be reduced from 20 to 8 (4 obj creation and 4 for method calling).

Inheritance Basics and Constructors

Inheritance allows one class (child) to use properties and methods of another class (parent). For inheritance to work properly, the constructors of parent and child classes must match. When a child object is created, the parent class is instantiated first, followed by the child.

The Final Keyword in Java

The final keyword can be used with classes, methods, and variables. A final class cannot be extended, a final method cannot be overridden, and a final variable's value cannot be changed after assignment, except through the constructor.

Introduction to Polymorphism: Overloading vs. Overriding

Polymorphism means "many forms." Method overloading is when multiple methods in the same class have the same name but different parameters. Method overriding is when a child class rewrites a method from its parent class to provide its own implementation.

Method Overriding Explained

Method overriding allows a child class to provide a new version of a method inherited from the parent class. This enables the child class to specialize or change inherited behavior.

Method Overloading Explained

Method overloading lets a class have multiple methods with the same name but different parameter lists. This is useful for performing similar actions with different types or numbers of inputs.

Parameter Types and Method Overloading

Overloaded methods must differ in their parameter types, order, or number. Changing only the return type or access modifier does not count as overloading.

What is Polymorphism and Why Use It?

Polymorphism allows a single statement or method call to perform different tasks depending on the object it operates on. This reduces the number of lines of code and increases code flexibility.

Types of Polymorphism: Compile-Time vs. Run-Time

Compile-time polymorphism (method overloading) is resolved during code compilation, while run-time polymorphism (method overriding) is determined when the program runs. Compile-time polymorphism uses method overloading, and run-time uses method overriding.

Demonstrating Compile-Time Polymorphism

By overloading methods, different tasks can be performed by the same method name depending on the input parameters. The correct method is chosen at compile time based on the arguments provided.

Demonstrating Run-Time Polymorphism

Run-time polymorphism is achieved through method overriding. By using a parent class reference to hold

child class objects, the overridden method of the actual object type is called at run time, not the parent's version.

Non-Polymorphic vs. Polymorphic Code

Without polymorphism, you must create separate objects and call methods for each class, leading to a lot of repeated code. With run-time polymorphism, a single parent class reference can be used to call methods on different child class objects, reducing code length.

Upcasting and Its Role in Polymorphism

Upcasting is storing a child class object in a parent class reference. This allows the same parent reference to point to different child objects, enabling polymorphic behavior.

Reducing Code with Helper Methods

Repetitive code can be further reduced by placing common code in a helper method (such as a "garage" class), which accepts a parent type parameter. This method can then operate on any child object, making code even more concise.

Identifying Run-Time Polymorphism in Practice

Run-time polymorphism can be confirmed by passing different child objects to a method expecting a parent type, and observing that the correct child method executes. Errors related to object types will only appear during program execution, not compilation.

Benefits and Practice of Polymorphism

Polymorphism, especially run-time polymorphism, greatly reduces lines of code, supports code organization, and allows a single reference to operate on many object types. This approach is essential for scalable and maintainable code in large projects.