Lec 26 Interface PART - 1:

This lesson explains the concept of interfaces in Java. An interface is a blueprint for a class, specifying what methods a class should have without providing their actual code. Interfaces help standardize code, allow multiple inheritance, and promote loose coupling, making software easier to maintain and extend.

Static Concepts Recap

Static blocks, methods, and variables are related to the class itself, not to objects. A static variable defined outside a method is a class variable, not an instance variable.

Introduction to Interfaces

An interface is a special type of code block in Java, similar to a class but used for a different purpose. It acts as a blueprint that defines what methods a class should have, without implementing them.

Interface as a Blueprint

Just as a class is a blueprint for creating objects, an interface is a blueprint for creating classes. It guides what methods must be present in classes that implement it, but does not specify how they work.

Real-World Use of Interfaces

In advanced Java applications, interfaces are used more frequently than classes, especially in frameworks like JDBC. Examples include Connection, PreparedStatement, and Statement, which are all interfaces used to interact with databases.

Key Features of Interfaces

Interfaces help achieve standardization, ensuring that different classes follow the same structure. They also allow multiple inheritance (a class can implement multiple interfaces) and promote loose coupling, making code modules independent from each other.

Loose Coupling Explained

Loose coupling means that changes in one part of the code do not cause errors in other parts. Interfaces help achieve this by separating the definition of methods from their implementation, making the code more flexible and easier to modify.

Creating and Using Interfaces

An interface can be created in Java using the interface keyword. It can only contain abstract methods, which are method declarations without a body. In Java 8 and later, interfaces can also have default and private methods, but traditionally they only have abstract methods.

Abstract Methods in Interfaces

Abstract methods are methods without a body, meaning they only have a name and signature. In interfaces, the abstract keyword is optional because all methods are abstract by default. Interfaces cannot contain methods with actual code (method bodies).

Interfaces Are Incomplete by Design

Since interfaces only have method declarations, they are incomplete and cannot be instantiated (no objects can be created from them). They serve only as a template for classes to implement.

Real-Time Example: Database Connectivity

Interfaces play a key role in connecting Java applications to different databases such as MySQL, Oracle, and DB2. Each database provides its own connector, usually written in Java, that implements standard interfaces to allow Java programs to interact with them.

Developer Exercise: Implementing Connectors

Developers for different databases (like MySQL, Oracle, DB2) create their own classes and methods to connect to their databases, following the structure defined by interfaces. Each connector class implements the required methods for connectivity, insert, delete, update, and read operations.--- This structure provides a clear learning path, focusing on the essential concepts and practical applications of interfaces in Java.

This lesson explains why and how Java interfaces are used to create a common standard for different database connectors. By using interfaces, developers ensure that all connectors have the same method names, making it easier to switch databases and maintain code.

Setting Up Java Projects and Sharing Code

Developers often need to share their Java projects as JAR files to allow others to use their code. Unlike simple file zipping, Java projects are exported as JARs, which package compiled classes and resources together for reuse and distribution.

Using JAR Files for Database Connectivity

To connect Java applications to different databases (like MySQL, Oracle, or DB2), developers use specific connector JARs provided by each database vendor. These JARs contain classes and methods needed for database communication.

Integrating Database Connectors in Java Projects

Adding a connector JAR to a Java project involves updating the build path and referencing the classes inside the JAR. This allows the project to use the connector’s methods for operations like connecting, reading, and deleting data.

The Problem of Inconsistent Method Names

Each database connector can have different class and method names for similar operations (e.g., connecting, reading, deleting). This inconsistency makes it hard for developers to switch databases or maintain code, as they must remember many different names.

JDBC and the Power of Standardized Methods

JDBC (Java Database Connectivity) solves the inconsistency problem by providing a set of standard method names (like getConnection, executeQuery, executeUpdate) that all database connectors must implement. This allows developers to use any database with the same method names.

Why Do Different Connectors Have Different Methods?

The reason for different method names is that companies develop connectors independently, without coordination or shared standards. This lack of communication leads to each company inventing its own method names, causing confusion for developers.

Interfaces as a Solution for Standardization

An interface in Java acts as a blueprint that specifies method names and signatures but not their implementation. When all connectors implement the same interface, they must use the same method names, ensuring consistency across different connectors.

Creating and Distributing a Standard Interface

The Java team can create a standard interface (e.g., with getConnection, insert, read, delete methods) and distribute it as a JAR. All connector developers then implement this interface, ensuring their classes have the required methods.

Implementing the Standard Interface in Connectors

To use the standard interface, connector classes use the implements keyword and provide concrete code for each method defined in the interface. This guarantees that all connectors have the same method signatures, even if the internal code is different.

Interface as a Blueprint and the Rule of Implementation

An interface only defines method names (no bodies). Any class implementing the interface must provide the actual code for these methods. This rule ensures that the interface acts as a contract, and all implementing classes fulfill it.

The Impact of Standardization and Interface Use

Standardization through interfaces reduces confusion, simplifies maintenance, and makes it easy to switch between different connectors. Developers only need to remember a few standard method names, regardless of the database used.

Key Points and Limitations of Interfaces

Interfaces can only have abstract methods (methods without bodies) and cannot be instantiated directly. Objects are created from the implementation classes, not the interfaces themselves.

Applying the Standard Interface in Practice

After implementing the standard interface, each developer creates their own connector, but all connectors now have consistent method names. This solves the initial problem of inconsistent naming and makes integration seamless.--- This summary covers the key concepts and practical applications of interfaces and standardization in Java, especially for database connectivity.