In Java, **method hiding** occurs when a **static method** in a subclass has the same name and parameter list as a static method in its parent class. Method hiding is different from method overriding because static methods belong to the class, not the instance, and are resolved at **compile-time**, not runtime.

Here’s a **real-time example** of **method hiding** in Java:

**Example: Banking System with Method Hiding**

class Bank {

// Static method in the parent class

public static void displayInterestRate() {

System.out.println("The interest rate for a regular bank is 4%.");

}

}

class OnlineBank extends Bank {

// Static method in the child class (method hiding)

public static void displayInterestRate() {

System.out.println("The interest rate for an online bank is 5%.");

}

}

public class MethodHidingExample {

public static void main(String[] args) {

// Call the static method using the parent class reference

Bank.displayInterestRate(); // Calls the method in Bank class

// Call the static method using the child class reference

OnlineBank.displayInterestRate(); // Calls the method in OnlineBank class

// Using a parent reference to a child object

Bank bankRef = new OnlineBank();

bankRef.displayInterestRate(); // Still calls the method in Bank class

}

}

**Output:**

The interest rate for a regular bank is 4%.

The interest rate for an online bank is 5%.

The interest rate for a regular bank is 4%.

**Explanation:**

1. **Static Method Resolution**:
   * Static methods are resolved at **compile-time** based on the reference type, not the actual object.
   * Bank.displayInterestRate() calls the method in the Bank class.
   * OnlineBank.displayInterestRate() calls the method in the OnlineBank class.
2. **Parent Reference to Child Object**:
   * Even though bankRef refers to a OnlineBank object, the static method from the Bank class is invoked because the reference type is Bank.
3. **Key Difference from Overriding**:
   * If the methods were not static, the method in the subclass would override the one in the superclass, and runtime polymorphism would determine which method is called.

**Real-Time Use Case:**

In a banking system:

* A **regular bank** and an **online bank** might have different static configurations (like interest rates).
* Using method hiding, the relevant configuration can be displayed based on the class being accessed, even though they share the same method name.

This example clearly demonstrates method hiding in a real-world context.