**Code Example: Simple Banking System**

java

Copy code

// Interface for basic account operations

interface Account {

void deposit(double amount);

void withdraw(double amount);

}

// Interface for loan services

interface Loan {

void applyForLoan(double amount);

}

// Class that implements both interfaces

class Customer implements Account, Loan {

private double balance;

// Constructor to initialize balance

public Customer(double initialBalance) {

this.balance = initialBalance;

}

// Implement deposit method from Account

public void deposit(double amount) {

balance += amount;

System.out.println("Deposited: " + amount + ". Current Balance: " + balance);

}

// Implement withdraw method from Account

public void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

System.out.println("Withdrew: " + amount + ". Current Balance: " + balance);

} else {

System.out.println("Insufficient balance! Current Balance: " + balance);

}

}

// Implement applyForLoan method from Loan

public void applyForLoan(double amount) {

System.out.println("Loan application submitted for amount: " + amount);

}

}

// Main class to test the banking system

public class SimpleBankingExample {

public static void main(String[] args) {

// Create a customer with an initial balance

Customer customer = new Customer(1000.0);

// Perform operations

customer.deposit(500.0); // Deposit money

customer.withdraw(300.0); // Withdraw money

customer.applyForLoan(2000.0); // Apply for a loan

}

}

**Output:**

yaml

Copy code

Deposited: 500.0. Current Balance: 1500.0

Withdrew: 300.0. Current Balance: 1200.0

Loan application submitted for amount: 2000.0

**Key Features:**

1. **Account Interface**:
   * Defines methods for basic account operations (deposit, withdraw).
2. **Loan Interface**:
   * Defines a method for loan services (applyForLoan).
3. **Customer Class**:
   * Implements both interfaces, demonstrating multiple inheritance.
   * Combines functionalities of managing a bank account and applying for loans.
4. **Main Class**:
   * Demonstrates simple banking operations using the Customer class.

This example is simple, clear, and highlights the use of multiple inheritance in a real-world banking scenario.

In Java, **multiple inheritance** is not supported with classes to avoid ambiguity caused by the diamond problem. However, Java allows **multiple inheritance** using **interfaces**, as an interface only declares method signatures without providing their implementation.

Here's an example of achieving multiple inheritance in Java using interfaces:

**Example: Multiple Inheritance with Interfaces**

java

Copy code

// Interface 1

interface Animal {

void eat(); // Abstract method

}

// Interface 2

interface Pet {

void play(); // Abstract method

}

// Class that implements both interfaces

class Dog implements Animal, Pet {

// Implementing the eat() method from Animal interface

public void eat() {

System.out.println("The dog eats food.");

}

// Implementing the play() method from Pet interface

public void play() {

System.out.println("The dog plays fetch.");

}

}

// Main class to test the multiple inheritance

public class MultipleInheritanceExample {

public static void main(String[] args) {

Dog dog = new Dog();

// Call methods from both interfaces

dog.eat(); // Method from Animal interface

dog.play(); // Method from Pet interface

}

}

**Output:**

Copy code

The dog eats food.

The dog plays fetch.

**Key Points:**

1. **Interfaces for Multiple Inheritance**:
   * A class can implement multiple interfaces using the implements keyword.
   * This allows combining behaviors from different sources without ambiguity.
2. **Method Implementation**:
   * The implementing class (Dog in this case) must provide concrete implementations for all abstract methods in the interfaces.
3. **Why Not Classes?**:
   * If Java allowed multiple inheritance with classes, it could lead to ambiguity when a method with the same signature exists in both parent classes. This is avoided by allowing multiple inheritance only through interfaces.
4. **Default Methods in Interfaces**:
   * Since Java 8, interfaces can have **default methods** (methods with a body). If two interfaces provide default methods with the same name, the implementing class must resolve the conflict by overriding the method.

**Example of Conflict with Default Methods:**

java

Copy code

interface A {

default void show() {

System.out.println("Default method from A");

}

}

interface B {

default void show() {

System.out.println("Default method from B");

}

}

class C implements A, B {

// Resolving conflict by overriding the show() method

public void show() {

System.out.println("Overriding the conflicting methods");

}

}

public class ConflictResolutionExample {

public static void main(String[] args) {

C obj = new C();

obj.show(); // Calls the overridden method

}

}

**Output:**

Copy code

Overriding the conflicting methods

This demonstrates how Java handles potential conflicts in multiple inheritance scenarios.