Week 10 - S10 - Advanced OOP - UML Diagram - Practice Problem

Name: Ramesh Harisabapathi Chettiar

Date of Submission: 17/10/2025

QNO1→

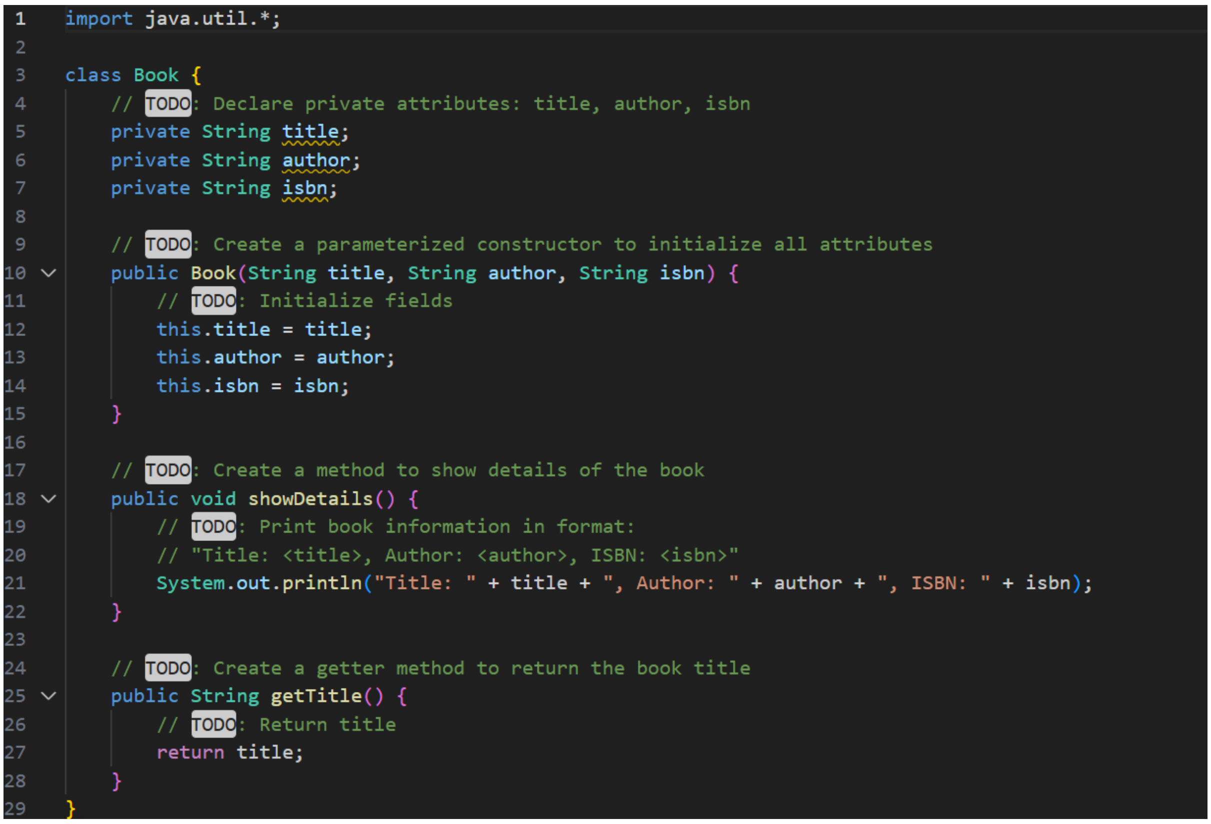
Problem Statement: Design a simple Library Management System that tracks books and Members.

● A Library contains multiple Books.

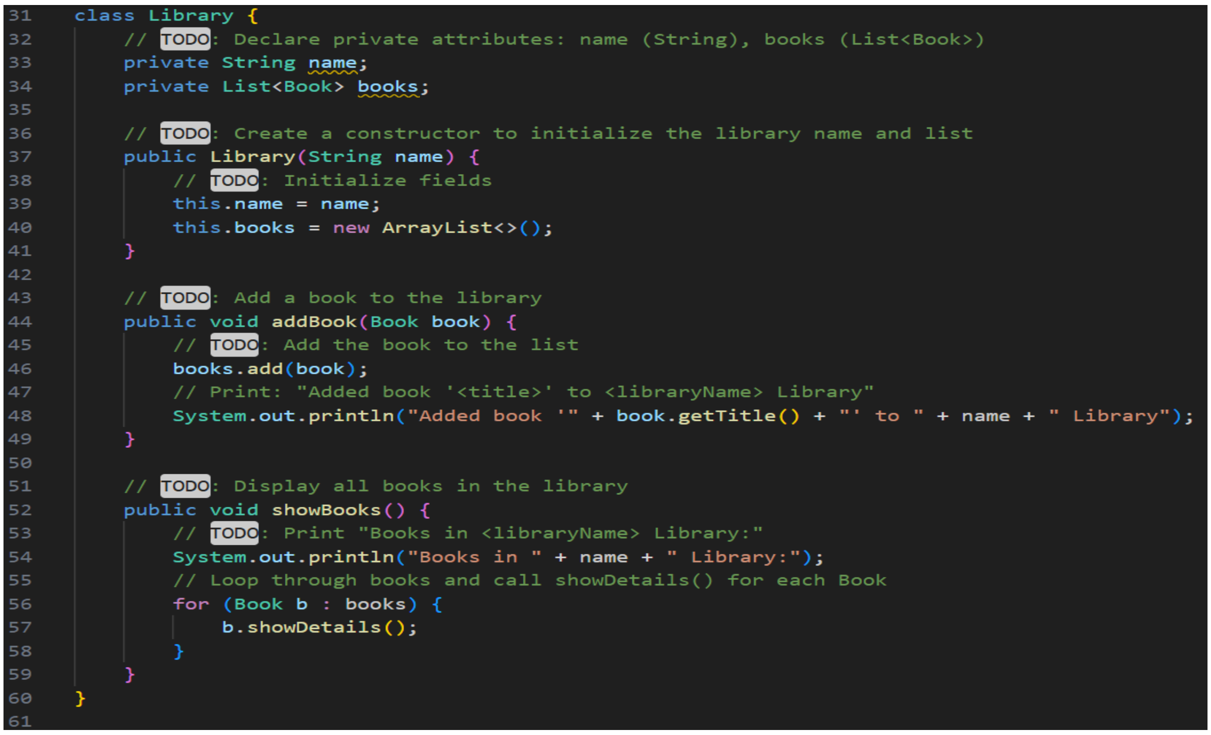
● A Member can borrow multiple Books.

● If the Library is deleted, all its Books are also removed (Composition relationship).

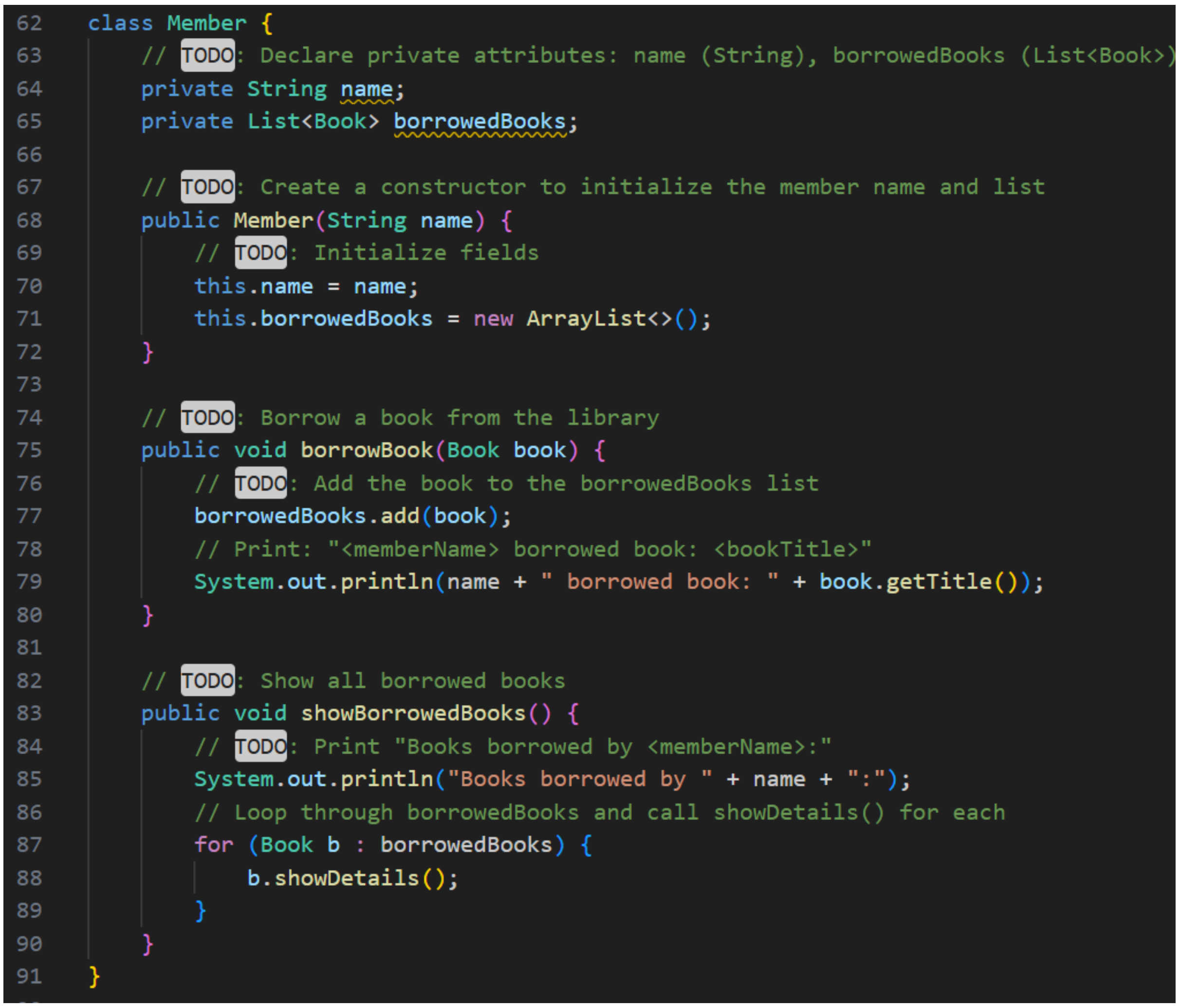
LibraryDemo.java



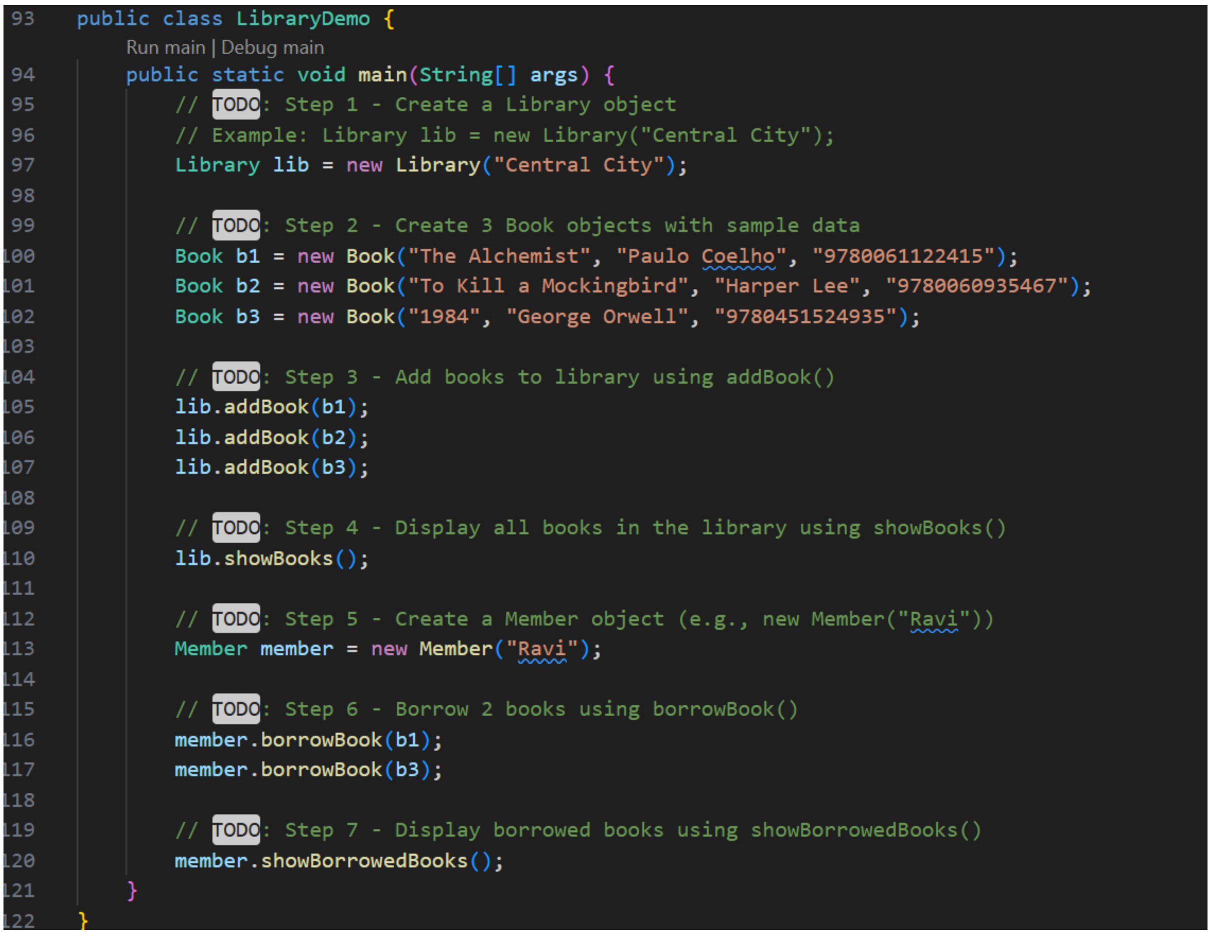
```java
import java.util.*;

class Book {
    // TODO: Declare private attributes: title, author, isbn
    private String title;
    private String author;
    private String isbn;

    // TODO: Create a parameterized constructor to initialize all attributes
    public Book(String title, String author, String isbn) {
        // TODO: Initialize fields
        this.title = title;
        this.author = author;
        this.isbn = isbn;
    }

    // TODO: Create a method to show details of the book
    public void showDetails() {
        // TODO: Print book information in format:
        // "Title: <title>, Author: <author>, ISBN: <isbn>"
        System.out.println("Title: " + title + ", Author: " + author + ", ISBN: " + isbn);
    }

    // TODO: Create a getter method to return the book title
    public String getTitle() {
        // TODO: Return title
        return title;
    }
}
```



```java
class Library {
    // TODO: Declare private attributes: name (String), books (List<Book>)
    private String name;
    private List<Book> books;

    // TODO: Create a constructor to initialize the library name and list
    public Library(String name) {
        // TODO: Initialize fields
        this.name = name;
        this.books = new ArrayList<>();
    }

    // TODO: Add a book to the library
    public void addBook(Book book) {
        // TODO: Add the book to the list
        books.add(book);
        // Print: "Added book '<title>' to <libraryName> Library"
        System.out.println("Added book '" + book.getTitle() + "' to " + name + " Library");
    }

    // TODO: Display all books in the library
    public void showBooks() {
        // TODO: Print "Books in <libraryName> Library:"
        System.out.println("Books in " + name + " Library:");
        // Loop through books and call showDetails() for each Book
        for (Book b : books) {
            b.showDetails();
        }
    }
}

```

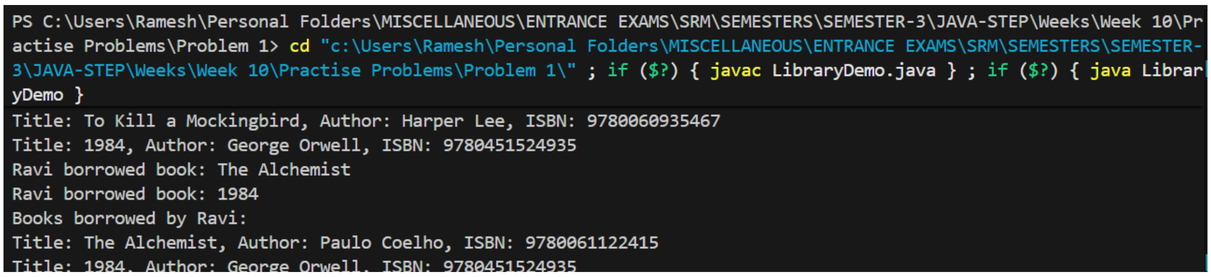


```java
class Member {
    // TODO: Declare private attributes: name (String), borrowedBooks (List<Book>)
    private String name;
    private List<Book> borrowedBooks;

    // TODO: Create a constructor to initialize the member name and list
    public Member(String name) {
        // TODO: Initialize fields
        this.name = name;
        this.borrowedBooks = new ArrayList<>();
    }

    // TODO: Borrow a book from the library
    public void borrowBook(Book book) {
        // TODO: Add the book to the borrowedBooks list
        borrowedBooks.add(book);
        // Print: "<memberName> borrowed book: <bookTitle>"
        System.out.println(name + " borrowed book: " + book.getTitle());
    }

    // TODO: Show all borrowed books
    public void showBorrowedBooks() {
        // TODO: Print "Books borrowed by <memberName>:"
        System.out.println("Books borrowed by " + name + ":");
        // Loop through borrowedBooks and call showDetails() for each
        for (Book b : borrowedBooks) {
            b.showDetails();
        }
    }
}
```



```java
public class LibraryDemo {
    public static void main(String[] args) {
        // TODO: Step 1 - Create a Library object
        // Example: Library lib = new Library("Central City");
        Library lib = new Library("Central City");

        // TODO: Step 2 - Create 3 Book objects with sample data
        Book b1 = new Book("The Alchemist", "Paulo Coelho", "9780061122415");
        Book b2 = new Book("To Kill a Mockingbird", "Harper Lee", "9780060935467");
        Book b3 = new Book("1984", "George Orwell", "9780451524935");

        // TODO: Step 3 - Add books to library using addBook()
        lib.addBook(b1);
        lib.addBook(b2);
        lib.addBook(b3);

        // TODO: Step 4 - Display all books in the library using showBooks()
        lib.showBooks();

        // TODO: Step 5 - Create a Member object (e.g., new Member("Ravi"))
        Member member = new Member("Ravi");

        // TODO: Step 6 - Borrow 2 books using borrowBook()
        member.borrowBook(b1);
        member.borrowBook(b3);

        // TODO: Step 7 - Display borrowed books using showBorrowedBooks()
        member.showBorrowedBooks();
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Pr
actise Problems\Problem 1> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-
3\JAVA-STEP\Weeks\Week 10\Practise Problems\Problem 1\" ; if ($?) { javac LibraryDemo.java } ; if ($?) { java Librar
yDemo }
Title: To Kill a Mockingbird, Author: Harper Lee, ISBN: 9780060935467
Title: 1984, Author: George Orwell, ISBN: 9780451524935
Ravi borrowed book: The Alchemist
Ravi borrowed book: 1984
Books borrowed by Ravi:
Title: The Alchemist, Author: Paulo Coelho, ISBN: 9780061122415
Title: 1984, Author: George Orwell, ISBN: 9780451524935
```

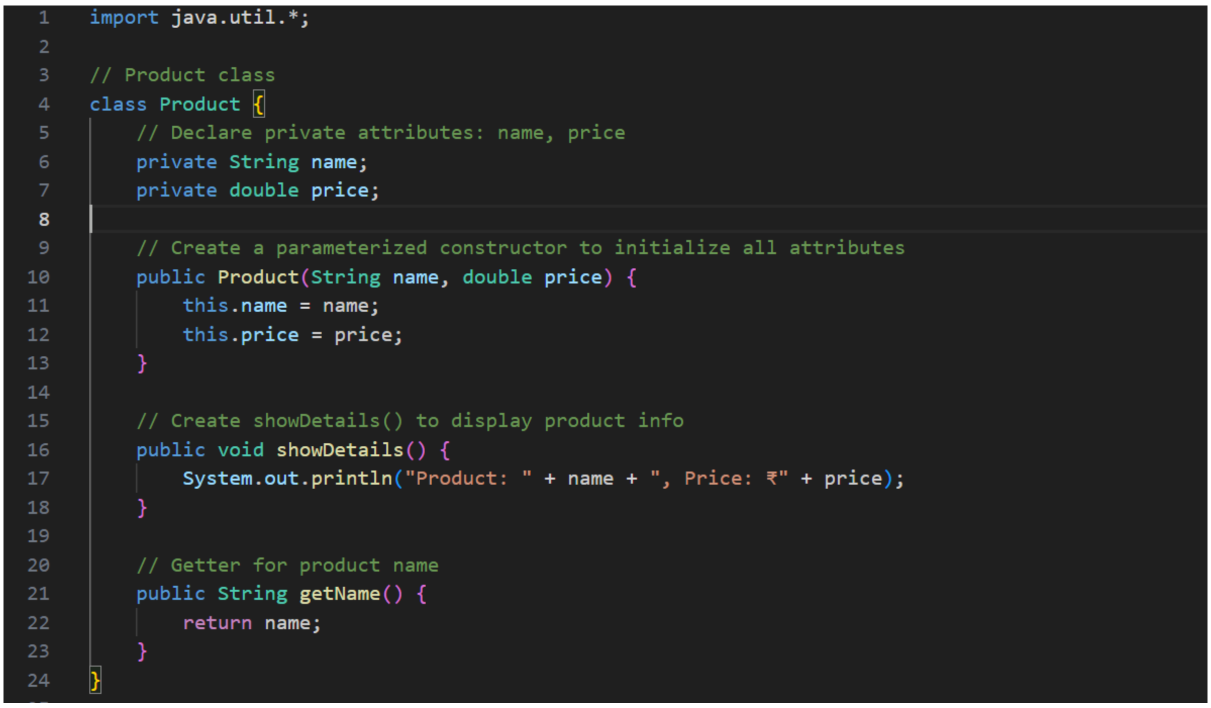
QNO2→

Design a simple Online Shopping System that represents the

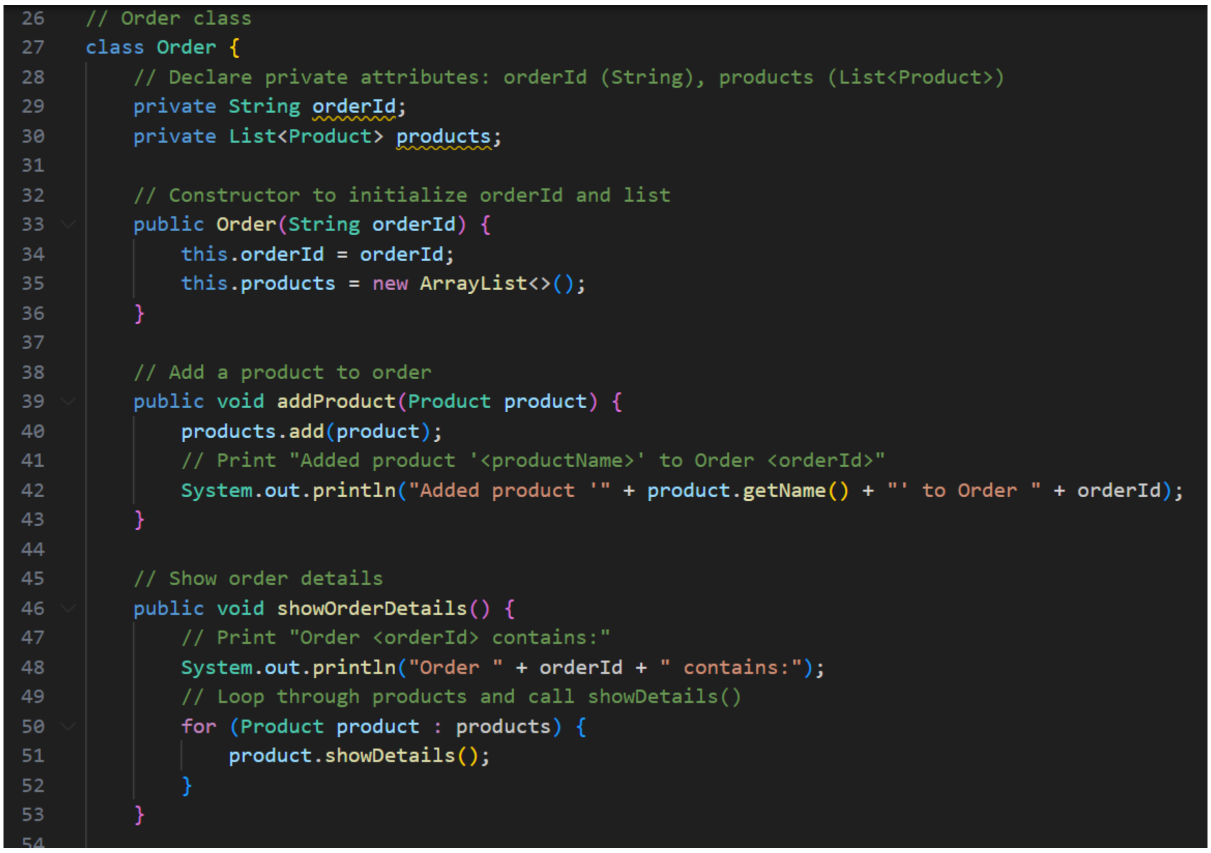
relationship between Customer, Order, and Product objects.

- A Customer can place multiple Orders.
- Each Order contains multiple Products.
- Each Product has a name and price.
- A Customer object has personal details like name and email.

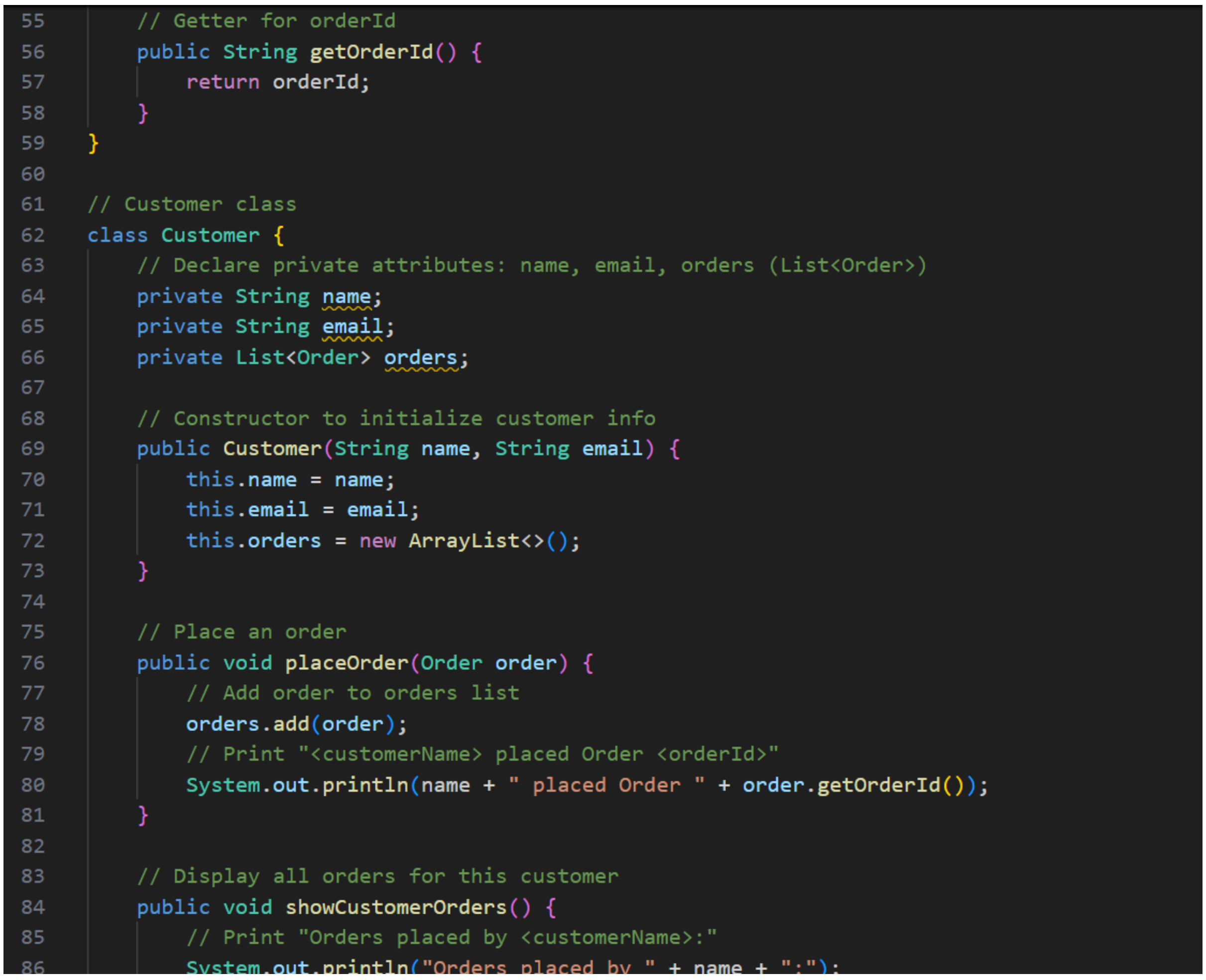
ShopingDemo.java



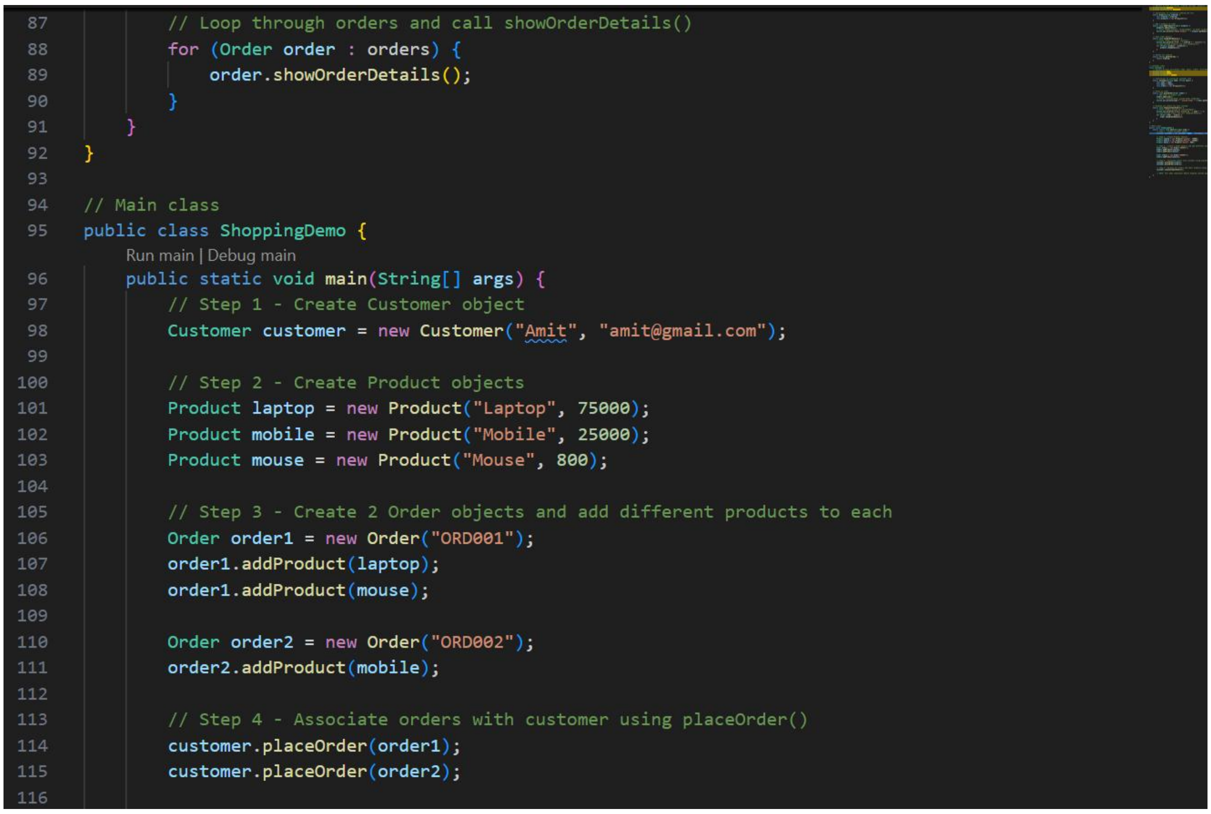
```java
import java.util.*;

// Product class
class Product {
    // Declare private attributes: name, price
    private String name;
    private double price;

    // Create a parameterized constructor to initialize all attributes
    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    // Create showDetails() to display product info
    public void showDetails() {
        System.out.println("Product: " + name + ", Price: ₹" + price);
    }

    // Getter for product name
    public String getName() {
        return name;
    }
}
```



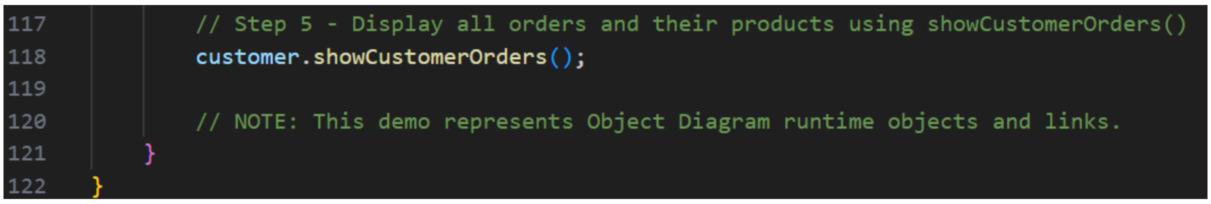
```java
// Order class
class Order {
    // Declare private attributes: orderId (String), products (List<Product>)
    private String orderId;
    private List<Product> products;

    // Constructor to initialize orderId and list
    public Order(String orderId) {
        this.orderId = orderId;
        this.products = new ArrayList<>();
    }

    // Add a product to order
    public void addProduct(Product product) {
        products.add(product);
        // Print "Added product '<productName>' to Order <orderId>"
        System.out.println("Added product '" + product.getName() + "' to Order " + orderId);
    }

    // Show order details
    public void showOrderDetails() {
        // Print "Order <orderId> contains:"
        System.out.println("Order " + orderId + " contains:");
        // Loop through products and call showDetails()
        for (Product product : products) {
            product.showDetails();
        }
    }

```



```java
    // Getter for orderId
    public String getOrderId() {
        return orderId;
    }
}

// Customer class
class Customer {
    // Declare private attributes: name, email, orders (List<Order>)
    private String name;
    private String email;
    private List<Order> orders;

    // Constructor to initialize customer info
    public Customer(String name, String email) {
        this.name = name;
        this.email = email;
        this.orders = new ArrayList<>();
    }

    // Place an order
    public void placeOrder(Order order) {
        // Add order to orders list
        orders.add(order);
        // Print "<customerName> placed Order <orderId>"
        System.out.println(name + " placed Order " + order.getOrderId());
    }

    // Display all orders for this customer
    public void showCustomerOrders() {
        // Print "Orders placed by <customerName>:"
        System.out.println("Orders placed by " + name + ":");
```

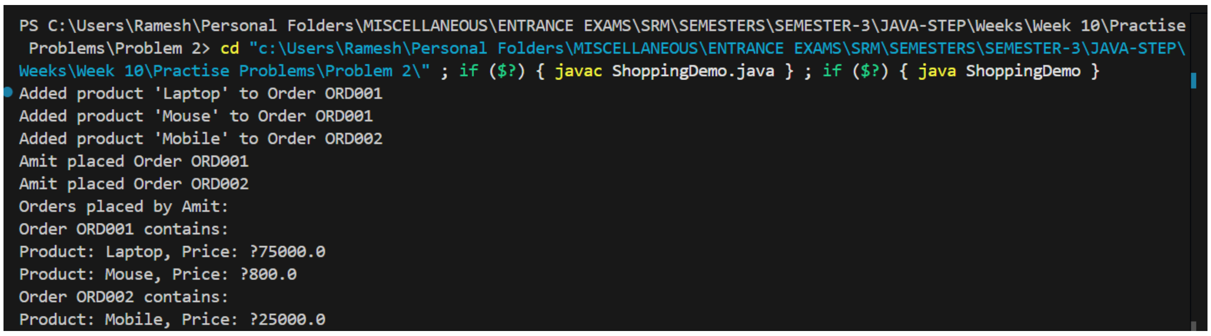


```java
            // Loop through orders and call showOrderDetails()
            for (Order order : orders) {
                order.showOrderDetails();
            }
        }
    }

    // Main class
    public class ShoppingDemo {
        public static void main(String[] args) {
            // Step 1 - Create Customer object
            Customer customer = new Customer("Amit", "amit@gmail.com");

            // Step 2 - Create Product objects
            Product laptop = new Product("Laptop", 75000);
            Product mobile = new Product("Mobile", 25000);
            Product mouse = new Product("Mouse", 800);

            // Step 3 - Create 2 Order objects and add different products to each
            Order order1 = new Order("ORD001");
            order1.addProduct(laptop);
            order1.addProduct(mouse);

            Order order2 = new Order("ORD002");
            order2.addProduct(mobile);

            // Step 4 - Associate orders with customer using placeOrder()
            customer.placeOrder(order1);
            customer.placeOrder(order2);

```



```java
            // Step 5 - Display all orders and their products using showCustomerOrders()
            customer.showCustomerOrders();

            // NOTE: This demo represents Object Diagram runtime objects and links.
        }
    }
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Practise
 Problems\Problem 2> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\
Weeks\Week 10\Practise Problems\Problem 2\" ; if ($?) { javac ShoppingDemo.java } ; if ($?) { java ShoppingDemo }
Added product 'Laptop' to Order ORD001
Added product 'Mouse' to Order ORD001
Added product 'Mobile' to Order ORD002
Amit placed Order ORD001
Amit placed Order ORD002
Orders placed by Amit:
Order ORD001 contains:
Product: Laptop, Price: ?75000.0
Product: Mouse, Price: ?800.0
Order ORD002 contains:
Product: Mobile, Price: ?25000.0
```

QNO3→

Design a Sequence Diagram that models an ATM withdrawal process between a Customer, ATM, and BankAccount.

When a customer inserts a card and requests withdrawal:

1. The Customer sends a request to the ATM.

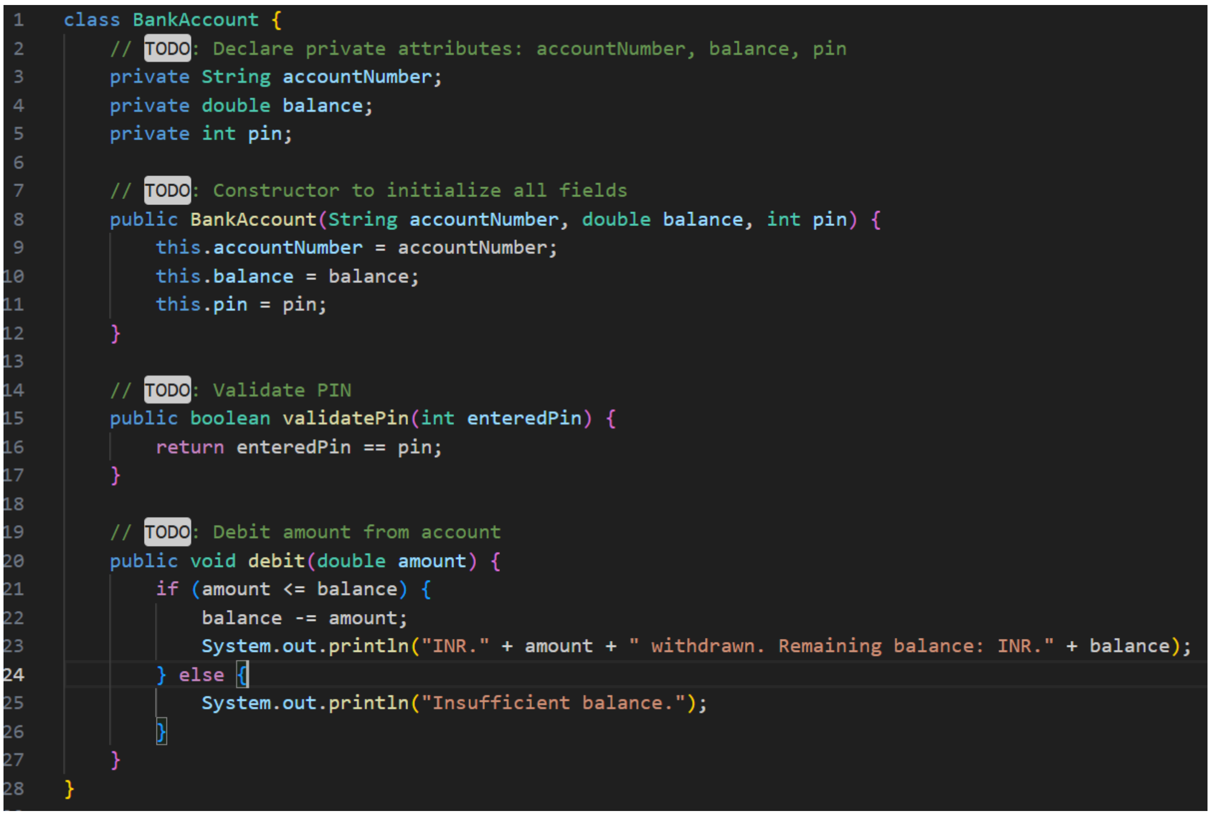
2. The ATM verifies the PIN with the BankAccount.

3. If successful, the BankAccount processes the withdrawal.

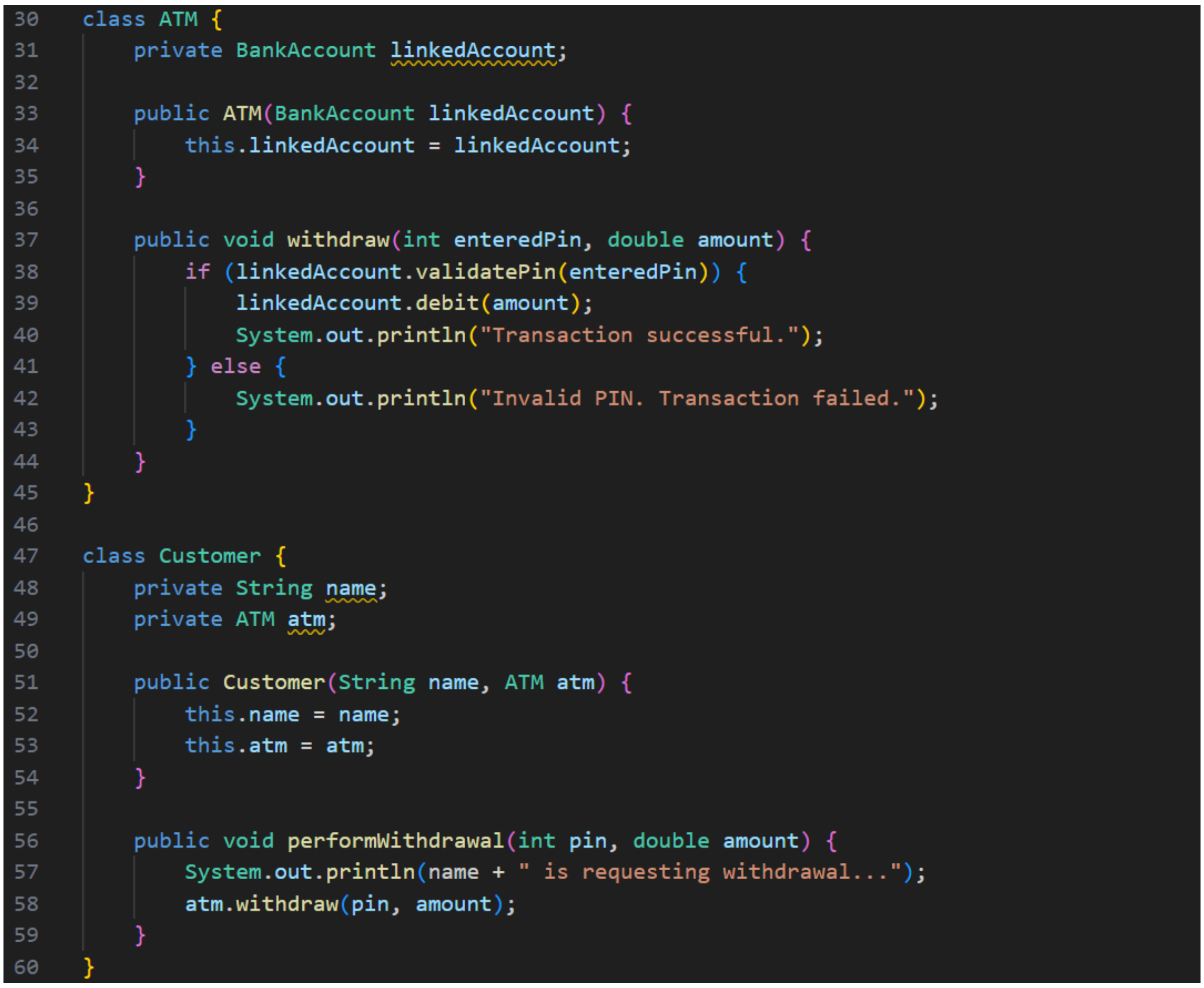
4. The ATM dispenses the cash.

5. The Customer receives confirmation.

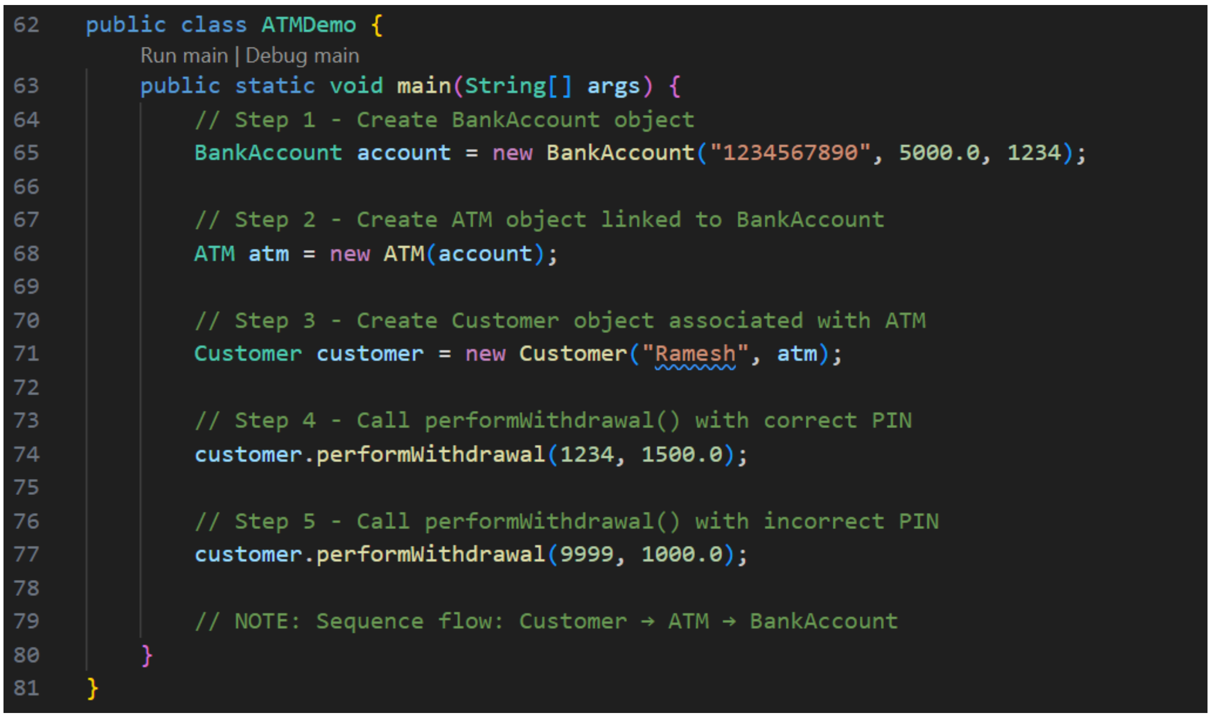
ATMDemo.java



```java
class BankAccount {
    // TODO: Declare private attributes: accountNumber, balance, pin
    private String accountNumber;
    private double balance;
    private int pin;

    // TODO: Constructor to initialize all fields
    public BankAccount(String accountNumber, double balance, int pin) {
        this.accountNumber = accountNumber;
        this.balance = balance;
        this.pin = pin;
    }

    // TODO: Validate PIN
    public boolean validatePin(int enteredPin) {
        return enteredPin == pin;
    }

    // TODO: Debit amount from account
    public void debit(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("INR." + amount + " withdrawn. Remaining balance: INR." + balance);
        } else {
            System.out.println("Insufficient balance.");
        }
    }
}
```

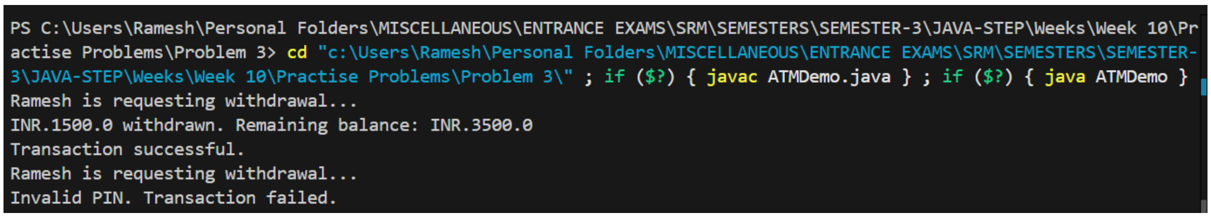


```java
class ATM {
    private BankAccount linkedAccount;

    public ATM(BankAccount linkedAccount) {
        this.linkedAccount = linkedAccount;
    }

    public void withdraw(int enteredPin, double amount) {
        if (linkedAccount.validatePin(enteredPin)) {
            linkedAccount.debit(amount);
            System.out.println("Transaction successful.");
        } else {
            System.out.println("Invalid PIN. Transaction failed.");
        }
    }
}

class Customer {
    private String name;
    private ATM atm;

    public Customer(String name, ATM atm) {
        this.name = name;
        this.atm = atm;
    }

    public void performWithdrawal(int pin, double amount) {
        System.out.println(name + " is requesting withdrawal...");
        atm.withdraw(pin, amount);
    }
}
```



```java
public class ATMDemo {
    public static void main(String[] args) {
        // Step 1 - Create BankAccount object
        BankAccount account = new BankAccount("1234567890", 5000.0, 1234);

        // Step 2 - Create ATM object linked to BankAccount
        ATM atm = new ATM(account);

        // Step 3 - Create Customer object associated with ATM
        Customer customer = new Customer("Ramesh", atm);

        // Step 4 - Call performWithdrawal() with correct PIN
        customer.performWithdrawal(1234, 1500.0);

        // Step 5 - Call performWithdrawal() with incorrect PIN
        customer.performWithdrawal(9999, 1000.0);

        // NOTE: Sequence flow: Customer → ATM → BankAccount
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Pr
actise Problems\Problem 3> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-
3\JAVA-STEP\Weeks\Week 10\Practise Problems\Problem 3\" ; if ($?) { javac ATMDemo.java } ; if ($?) { java ATMDemo }
Ramesh is requesting withdrawal...
INR.1500.0 withdrawn. Remaining balance: INR.3500.0
Transaction successful.
Ramesh is requesting withdrawal...
Invalid PIN. Transaction failed.
```