

**Practice Problem - UML with Java**

**Problem 1: Library Management System**

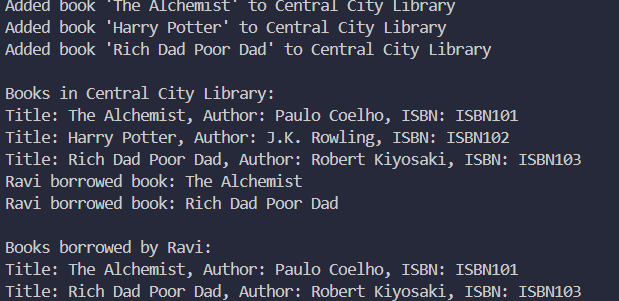
**Problem Statement:** Design a simple Library Management System that tracks books and Members.

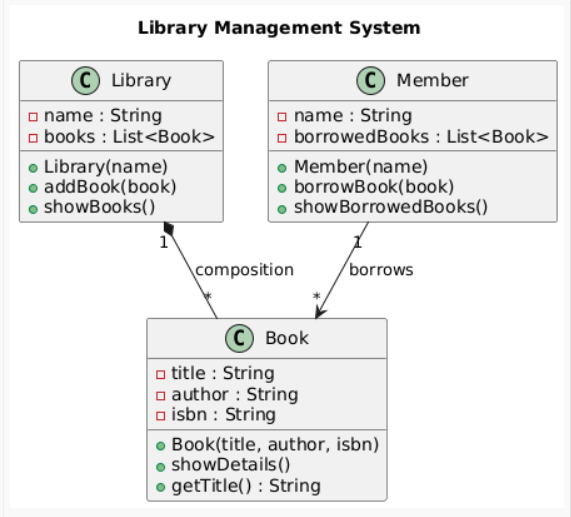
● A Library contains multiple Books.

● A Member can borrow multiple Books.

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

| import java.util.\*;  // ==================== CLASS: Book ==================== class Book {  private String title;  private String author;  private String isbn;   public Book(String title, String author, String isbn) {  this.title = title;  this.author = author;  this.isbn = isbn;  }   public void showDetails() {  System.out.println("Title: " + title + ", Author: " + author + ", ISBN: " + isbn);  }   public String getTitle() {  return title;  } }  // ==================== CLASS: Library ==================== class Library {  private String name;  private List<Book> books;   public Library(String name) {  this.name = name;  this.books = new ArrayList<>();  }   public void addBook(Book book) {  books.add(book);  System.out.println("Added book '" + book.getTitle() + "' to " + name + " Library");  }   public void showBooks() {  System.out.println("\nBooks in " + name + " Library:");  for (Book b : books) {  b.showDetails();  }  } }  // ==================== CLASS: Member ==================== class Member {  private String name;  private List<Book> borrowedBooks;   public Member(String name) {  this.name = name;  this.borrowedBooks = new ArrayList<>();  }   public void borrowBook(Book book) {  borrowedBooks.add(book);  System.out.println(name + " borrowed book: " + book.getTitle());  }   public void showBorrowedBooks() {  System.out.println("\nBooks borrowed by " + name + ":");  for (Book b : borrowedBooks) {  b.showDetails();  }  } }  // ==================== CLASS: LibraryDemo (Main) ==================== public class LibraryManagementSystem {  public static void main(String[] args) {  // Step 1 - Create a Library  Library lib = new Library("Central City");   // Step 2 - Create 3 Books  Book b1 = new Book("The Alchemist", "Paulo Coelho", "ISBN101");  Book b2 = new Book("Harry Potter", "J.K. Rowling", "ISBN102");  Book b3 = new Book("Rich Dad Poor Dad", "Robert Kiyosaki", "ISBN103");   // Step 3 - Add books to Library  lib.addBook(b1);  lib.addBook(b2);  lib.addBook(b3);   // Step 4 - Display all books  lib.showBooks();   // Step 5 - Create a Member  Member m1 = new Member("Ravi");   // Step 6 - Borrow 2 books  m1.borrowBook(b1);  m1.borrowBook(b3);   // Step 7 - Show borrowed books  m1.showBorrowedBooks();  } } |
| --- |





**Problem 2: Object Diagram – Online Shopping System**

**Problem Statement:** 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**.

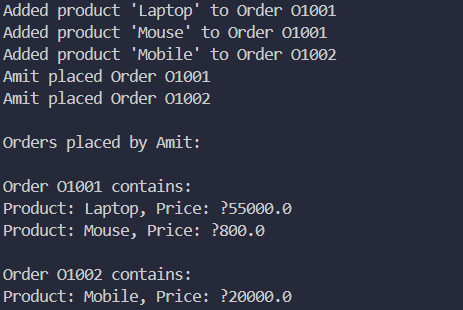
● Each **Product** has a name and price.

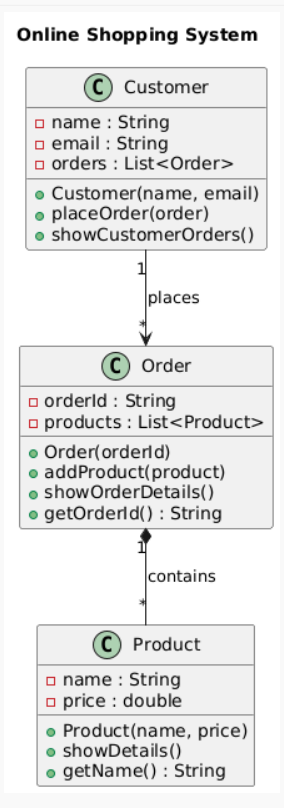
● A **Customer** object has personal details like name and email.

The goal is to visualize **specific object instances** and their **links** at runtime using an **Object Diagram**.

import java.util.\*;

| // ==================== CLASS: Product ==================== class Product {  private String name;  private double price;   public Product(String name, double price) {  this.name = name;  this.price = price;  }   public void showDetails() {  System.out.println("Product: " + name + ", Price: ₹" + price);  }   public String getName() {  return name;  } }  // ==================== CLASS: Order ==================== class Order {  private String orderId;  private List<Product> products;   public Order(String orderId) {  this.orderId = orderId;  this.products = new ArrayList<>();  }   public void addProduct(Product product) {  products.add(product);  System.out.println("Added product '" + product.getName() + "' to Order " + orderId);  }   public void showOrderDetails() {  System.out.println("\nOrder " + orderId + " contains:");  for (Product p : products) {  p.showDetails();  }  }   public String getOrderId() {  return orderId;  } }  // ==================== CLASS: Customer ==================== class Customer {  private String name;  private String email;  private List<Order> orders;   public Customer(String name, String email) {  this.name = name;  this.email = email;  this.orders = new ArrayList<>();  }   public void placeOrder(Order order) {  orders.add(order);  System.out.println(name + " placed Order " + order.getOrderId());  }   public void showCustomerOrders() {  System.out.println("\nOrders placed by " + name + ":");  for (Order o : orders) {  o.showOrderDetails();  }  } }  // ==================== CLASS: ShoppingDemo (Main) ==================== public class OnlineShoppingSystem {  public static void main(String[] args) {  // Step 1 - Create Customer  Customer customer1 = new Customer("Amit", "amit@gmail.com");   // Step 2 - Create Product objects  Product laptop = new Product("Laptop", 55000);  Product mobile = new Product("Mobile", 20000);  Product mouse = new Product("Mouse", 800);   // Step 3 - Create 2 Orders and add Products  Order order1 = new Order("O1001");  order1.addProduct(laptop);  order1.addProduct(mouse);   Order order2 = new Order("O1002");  order2.addProduct(mobile);   // Step 4 - Associate Orders with Customer  customer1.placeOrder(order1);  customer1.placeOrder(order2);   // Step 5 - Display Orders and their Products  customer1.showCustomerOrders();  } } |
| --- |





**Problem 3: Sequence Diagram – ATM Transaction System** �� **Problem Statement:**

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**.

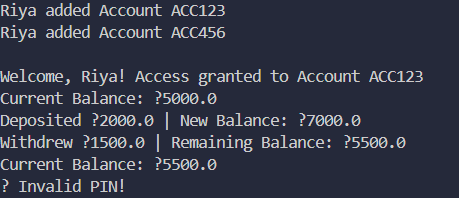
2. The **ATM** verifies the **PIN** with the **BankAccount**.

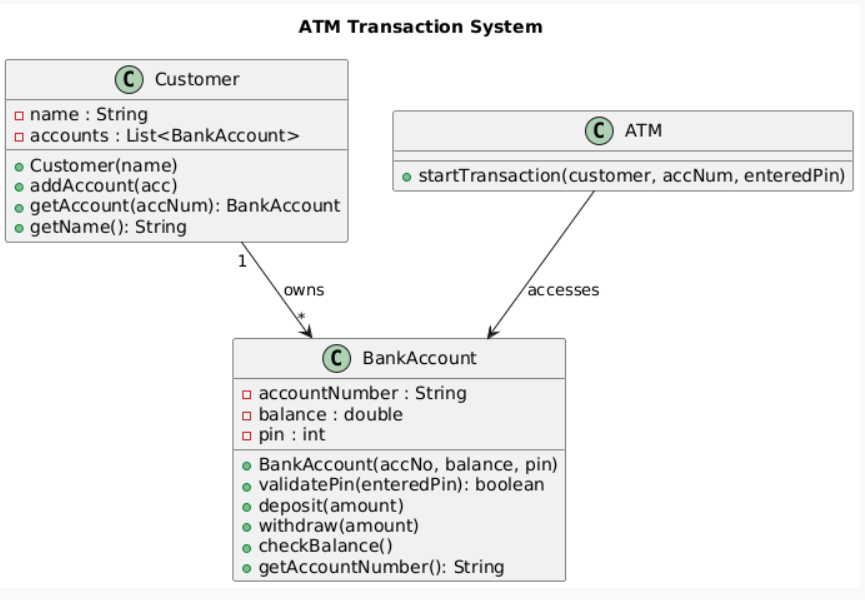
3. If successful, the **BankAccount** processes the withdrawal.

4. The **ATM** dispenses the cash.

5. The **Customer** receives confirmation.

| import java.util.\*;  // ==================== CLASS: BankAccount ==================== class BankAccount {  private String accountNumber;  private double balance;  private int pin;   public BankAccount(String accountNumber, double balance, int pin) {  this.accountNumber = accountNumber;  this.balance = balance;  this.pin = pin;  }   public boolean validatePin(int enteredPin) {  return this.pin == enteredPin;  }   public void deposit(double amount) {  balance += amount;  System.out.println("Deposited ₹" + amount + " | New Balance: ₹" + balance);  }   public void withdraw(double amount) {  if (amount > balance) {  System.out.println("❌ Insufficient balance!");  } else {  balance -= amount;  System.out.println("Withdrew ₹" + amount + " | Remaining Balance: ₹" + balance);  }  }   public void checkBalance() {  System.out.println("Current Balance: ₹" + balance);  }   public String getAccountNumber() {  return accountNumber;  } }  // ==================== CLASS: Customer ==================== class Customer {  private String name;  private List<BankAccount> accounts;   public Customer(String name) {  this.name = name;  this.accounts = new ArrayList<>();  }   public void addAccount(BankAccount acc) {  accounts.add(acc);  System.out.println(name + " added Account " + acc.getAccountNumber());  }   public BankAccount getAccount(String accNum) {  for (BankAccount acc : accounts) {  if (acc.getAccountNumber().equals(accNum))  return acc;  }  return null;  }   public String getName() {  return name;  } }  // ==================== CLASS: ATM ==================== class ATM {  public void startTransaction(Customer customer, String accNum, int enteredPin) {  BankAccount account = customer.getAccount(accNum);  if (account == null) {  System.out.println("Account not found!");  return;  }   if (!account.validatePin(enteredPin)) {  System.out.println("❌ Invalid PIN!");  return;  }   System.out.println("\nWelcome, " + customer.getName() + "! Access granted to Account " + accNum);  account.checkBalance();   // Example transactions  account.deposit(2000);  account.withdraw(1500);  account.checkBalance();  } }  // ==================== MAIN CLASS ==================== public class ATMSystem {  public static void main(String[] args) {  Customer c1 = new Customer("Riya");   BankAccount acc1 = new BankAccount("ACC123", 5000, 1234);  BankAccount acc2 = new BankAccount("ACC456", 10000, 5678);   c1.addAccount(acc1);  c1.addAccount(acc2);   ATM atm = new ATM();  atm.startTransaction(c1, "ACC123", 1234); // correct PIN  atm.startTransaction(c1, "ACC456", 1111); // wrong PIN  } } |
| --- |





10