

**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).

**UML Diagram Instructions:**

1. Draw the Class Diagram showing:

**○** Classes: Library, Book, Member

2. Relationships:

**○** Library → Book → Composition (Library owns Books)

**○** Member → Book → Association (Member borrows Books)

3. Show attributes and methods with proper visibility:

○ Private fields

○ Public methods

4. Indicate multiplicity:

**○** One Library → Many Books

**○** One Member → Many Books

**Java Implementation Instructions**

| public 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;  } }  import java.util.\*;  public class Library {  private String name;  private List<Book> books;   public Library(String name) {  this.name = name;  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("Books in " + name + " Library:");  for (Book b : books) b.showDetails();  } }  import java.util.\*;  public class Member {  private String name;  private List<Book> borrowedBooks;   public Member(String name) {  this.name = name;  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("Books borrowed by " + name + ":");  for (Book b : borrowedBooks) b.showDetails();  } }  public class LibraryDemo {  public static void main(String[] args) {  Library lib = new Library("Central City");  Book b1 = new Book("Java Basics", "Herbert Schildt", "J101");  Book b2 = new Book("OOP Concepts", "Grady Booch", "O202");  Book b3 = new Book("Data Structures", "Cormen", "D303");  lib.addBook(b1);  lib.addBook(b2);  lib.addBook(b3);  lib.showBooks();  Member m = new Member("Ravi");  m.borrowBook(b1);  m.borrowBook(b2);  m.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**.

**UML Diagram Instructions:**

1. **Draw an Object Diagram** showing *sample object instances*:

○ customer1 : Customer

○ order1, order2 : Order

○ product1, product2, product3 : Product

2. **Show links**:

○ customer1 → order1, order2 (association)

○ order1 → product1, product2 (composition)

3. **Represent values:**

○ Example:

■ customer1 : Customer → name = "Amit", email =

"amit@gmail.com"

■ product1 : Product → name = "Laptop", price = 55000

4



| public 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;  } } import java.util.\*;  public class Order {  private String orderId;  private List<Product> products;   public Order(String orderId) {  this.orderId = orderId;  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("Order " + orderId + " contains:");  for (Product p : products) p.showDetails();  } }  import java.util.\*;  public class Customer {  private String name;  private String email;  private List<Order> orders;   public Customer(String name, String email) {  this.name = name;  this.email = email;  orders = new ArrayList<>();  }   public void placeOrder(Order order) {  orders.add(order);  System.out.println(name + " placed Order " + orderId);  }   public void showCustomerOrders() {  System.out.println("Orders placed by " + name + ":");  for (Order o : orders) o.showOrderDetails();  } }  public class ShoppingDemo {  public static void main(String[] args) {  Customer c = new Customer("Amit", "amit@gmail.com");  Product p1 = new Product("Laptop", 55000);  Product p2 = new Product("Mobile", 20000);  Product p3 = new Product("Mouse", 800);  Order o1 = new Order("O101");  Order o2 = new Order("O102");  o1.addProduct(p1);  o1.addProduct(p2);  o2.addProduct(p3);  c.placeOrder(o1);  c.placeOrder(o2);  c.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.

**UML Diagram Instructions:**

7



1. Draw a **Sequence Diagram** with three lifelines:

○ Customer

○ ATM

○ BankAccount

2. Show message flow:

○ insertCard()

○ enterPIN()

○ requestWithdrawal(amount)

○ validatePIN()

○ debit(amount)

○ dispenseCash()

○ showConfirmation()

3. Use arrows to indicate the direction of method calls and returns. **Java Implementation Instructions :**

| public 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 enteredPin == pin;  }   public void debit(double amount) {  balance -= amount;  System.out.println("₹" + amount + " withdrawn. Remaining balance: ₹" + balance);  } }  public 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.");  }  } }  public class CustomerATM {  private String name;  private ATM atm;   public CustomerATM(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);  } }  public class ATMDemo {  public static void main(String[] args) {  BankAccount acc = new BankAccount("12345", 10000, 1234);  ATM atm = new ATM(acc);  CustomerATM cust = new CustomerATM("Rohit", atm);  cust.performWithdrawal(1234, 2000);  cust.performWithdrawal(1111, 1000);  } } |
| --- |

10