Week 10 - S10 - Advanced OOP - UML Diagram - Assignment Problem (HW)

Name: Ramesh Harisabapathi Chettiar

Date of Submission: 22/10/25

QNO1→

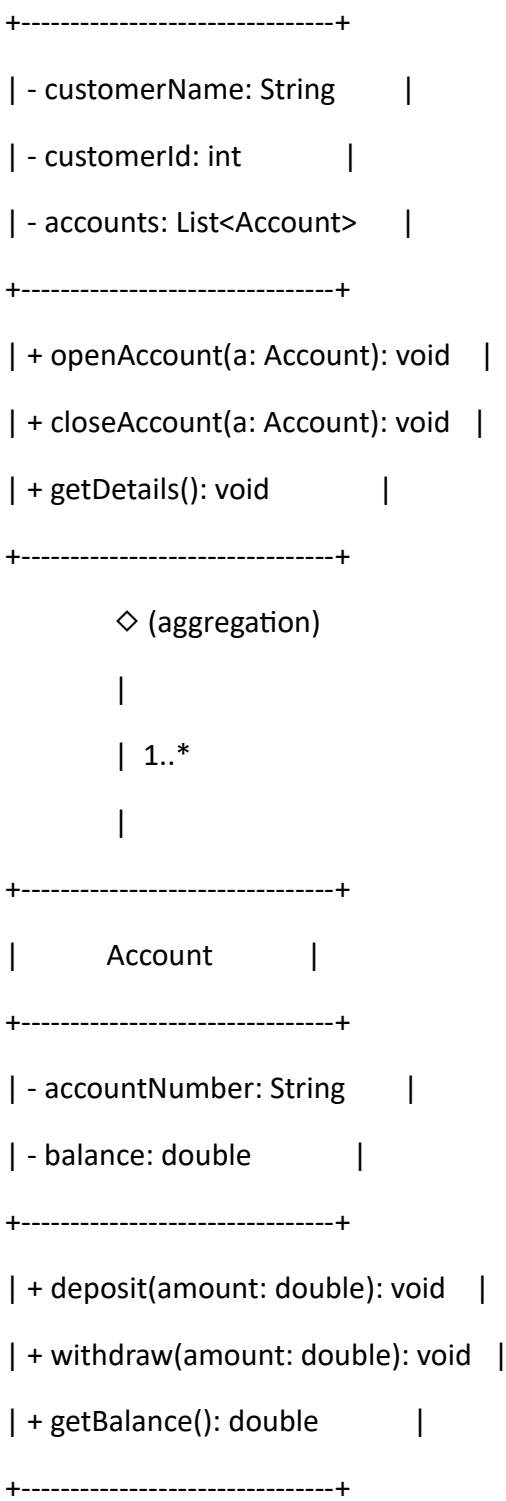
Draw a Class Diagram for a simple Bank Account System with classes such as Account, Customer, and Bank. Show attributes and methods for each class, and represent the relationships between them.

Hints:

- Use association between Bank and Customer.
- Represent aggregation between Customer and Account.
- Include visibility symbols (+, -, #) for attributes and methods.

1. UML Class Diagram (Text Representation)

```
+--------------------------------+
|            Bank                |
+--------------------------------+
| - bankName: String             |
| - branchCode: String           |
+--------------------------------+
| + addCustomer(c: Customer): void   |
| + viewCustomers(): void            |
+--------------------------------+
              1
              |
              |  (association)
              |
              *
+--------------------------------+
|           Customer             |
```


+--------------------------------+
| - customerName: String |
| - customerId: int |
| - accounts: List<Account> |
+--------------------------------+
| + openAccount(a: Account): void |
| + closeAccount(a: Account): void |
| + getDetails(): void |
+--------------------------------+
◇ (aggregation)
|
| 1..*
|
+--------------------------------+
| Account |
+--------------------------------+
| - accountNumber: String |
| - balance: double |
+--------------------------------+
| + deposit(amount: double): void |
| + withdraw(amount: double): void |
| + getBalance(): double |
+--------------------------------+

BankAccountSystem.java

BankAccountSystem.java > Customer > getDetails()

```java
import java.util.*;

class Account {
    private String accountNumber;
    private double balance;

    public Account(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: INR. " + amount + " | New Balance: INR. " + balance);
    }

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: INR. " + amount + " | Remaining Balance: INR. " + balance);
        } else {
            System.out.println("Insufficient balance!");
        }
    }

    public double getBalance() {
        return balance;
    }

    public String getAccountNumber() {
        return accountNumber;
    }
}
class Customer {
    private String customerName;
    private int customerId;
    private List<Account> accounts = new ArrayList<>();

    public Customer(String customerName, int customerId) {
        this.customerName = customerName;
        this.customerId = customerId;
    }

    public void openAccount(Account a) {
        accounts.add(a);
        System.out.println(customerName + " opened account: " + a.getAccountNumber());
    }

    public void closeAccount(Account a) {
        accounts.remove(a);
        System.out.println(customerName + " closed account: " + a.getAccountNumber());
    }

    public void getDetails() {
        System.out.println("Customer: " + customerName + " | ID: " + customerId);
        System.out.println("Accounts:");
        for (Account a : accounts) {
            System.out.println(" - " + a.getAccountNumber() + " | Balance: INR. " + a.getBalance());
        }
    }
}
```

```java
class Bank {
    private String bankName;
    private String branchCode;
    private List<Customer> customers = new ArrayList<>();

    public Bank(String bankName, String branchCode) {
        this.bankName = bankName;
        this.branchCode = branchCode;
    }

    public void addCustomer(Customer c) {
        customers.add(c);
        System.out.println("Added customer: " + c);
    }

    public void viewCustomers() {
        System.out.println("Bank: " + bankName + " | Branch: " + branchCode);
        System.out.println("Customers list:");
        for (Customer c : customers) {
            c.getDetails();
        }
    }
}
```

```java
public class BankAccountSystem {
    Run main | Debug main
    public static void main(String[] args) {
        Bank bank = new Bank("National Bank", "NB001");

        Customer c1 = new Customer("Karthik", 101);
        Account a1 = new Account("SAV123", 5000);
        Account a2 = new Account("CUR456", 15000);

        c1.openAccount(a1);
        c1.openAccount(a2);

        a1.deposit(2000);
        a2.withdraw(3000);

        bank.addCustomer(c1);
        bank.viewCustomers();
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Assignment HW\Program 1> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Assignment HW\Program 1\" ; if ($?) { javac BankAccountSystem.java } ; if ($?) { java BankAccountSystem }
Karthik opened account: SAV123
Karthik opened account: CUR456
Deposited: INR. 2000.0 | New Balance: INR. 7000.0
Withdrawn: INR. 3000.0 | Remaining Balance: INR. 12000.0
Added customer: Customer@73d16e93
Bank: National Bank | Branch: NB001
Customers list:
Customer: Karthik | ID: 101
Accounts:
 - SAV123 | Balance: INR. 7000.0
 - CUR456 | Balance: INR. 12000.0
```

QNO2→

Problem Statement 2:

Create an Object Diagram showing instances of Customer, Account, and Bank classes.

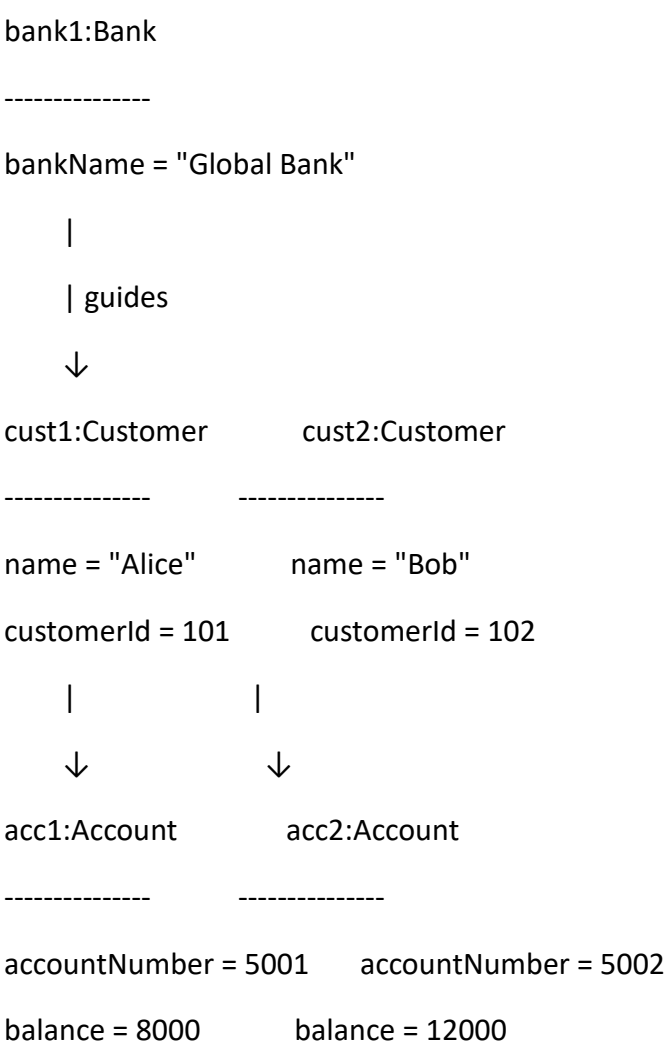
Display how specific objects are linked during runtime (e.g., one bank has multiple customers,

each with an account).

Hints:

- Show object names like cust1:Customer, acc1:Account.
- Include attribute values in objects (e.g., balance = 5000).
- Demonstrate links between instantiated objects.

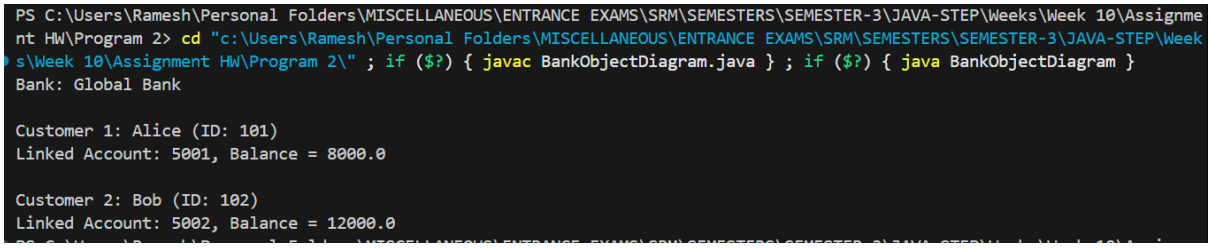
Visual Representation (Object Diagram)



BankObjectDiagram.java

```java
class Bank {
    String bankName;

    public Bank(String bankName) {
        this.bankName = bankName;
    }
}

class Customer {
    String name;
    int customerId;

    public Customer(String name, int customerId) {
        this.name = name;
        this.customerId = customerId;
    }
}

class Account {
    int accountNumber;
    double balance;

    public Account(int accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }
}

public class BankObjectDiagram {
    public static void main(String[] args) {
        // Create bank object
        Bank bank1 = new Bank("Global Bank");

        // Create customers
        Customer cust1 = new Customer("Alice", 101);
        Customer cust2 = new Customer("Bob", 102);

        // Create accounts
        Account acc1 = new Account(5001, 8000);
        Account acc2 = new Account(5002, 12000);

        // Display object relationships (conceptually showing runtime links)
        System.out.println("Bank: " + bank1.bankName);
        System.out.println("\nCustomer 1: " + cust1.name + " (ID: " + cust1.customerId + ")");
        System.out.println("Linked Account: " + acc1.accountNumber + ", Balance = " + acc1.balance);

        System.out.println("\nCustomer 2: " + cust2.name + " (ID: " + cust2.customerId + ")");
        System.out.println("Linked Account: " + acc2.accountNumber + ", Balance = " + acc2.balance);
    }
}
```

OUTPUT→


PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Assignment HW\Program 2> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Assignment HW\Program 2\" ; if ($?) { javac BankObjectDiagram.java } ; if ($?) { java BankObjectDiagram }
Bank: Global Bank
Customer 1: Alice (ID: 101)
Linked Account: 5001, Balance = 8000.0
Customer 2: Bob (ID: 102)
Linked Account: 5002, Balance = 12000.0

QNO3→

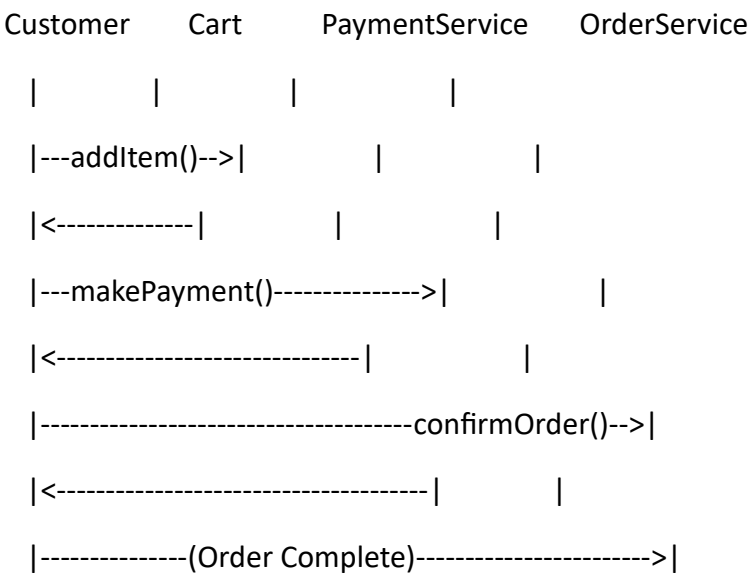
Problem Statement 3:

Draw a Sequence Diagram for an Online Shopping System showing interactions among Customer, Cart, PaymentService, and OrderService when a customer places an order.

Hints:

- Use lifelines for each participant.
- Show method calls (e.g., addItem(), makePayment(), confirmOrder()).
- Include return messages and activation boxes.

Sequence Diagram (Text Representation)



OnlineShoppingSequenceDiagram.java

```java
class Cart {
    void addItem(String item) {
        System.out.println("Cart: Item '" + item + "' added to cart.");
    }
}

class PaymentService {
    void makePayment(double amount) {
        System.out.println("PaymentService: Payment of ₹" + amount + " processed successfully.");
    }
}

class OrderService {
    void confirmOrder(String item) {
        System.out.println("OrderService: Order confirmed for item '" + item + "'.");
    }
}

class Customer {
    Cart cart = new Cart();
    PaymentService payment = new PaymentService();
    OrderService order = new OrderService();

    void placeOrder(String item, double amount) {
        System.out.println("Customer: Initiating order for '" + item + "'.");
        cart.addItem(item);
        payment.makePayment(amount);
        order.confirmOrder(item);
        System.out.println("Customer: Order placed successfully!");
    }
}

public class OnlineShoppingSequenceDiagram {
    public static void main(String[] args) {
        Customer customer = new Customer();
        customer.placeOrder("Wireless Mouse", 1200.00);
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Assignment HW\Program 3> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Assignment HW\Program 3\" ; if ($?) { javac OnlineShoppingSequenceDiagram.java } ; if ($?) { java OnlineShoppingSequenceDiagram }
Customer: Initiating order for 'Wireless Mouse'.
Cart: Item 'Wireless Mouse' added to cart.
PaymentService: Payment of ?1200.0 processed successfully.
OrderService: Order confirmed for item 'Wireless Mouse'.
Customer: Order placed successfully!
```

QNO4→

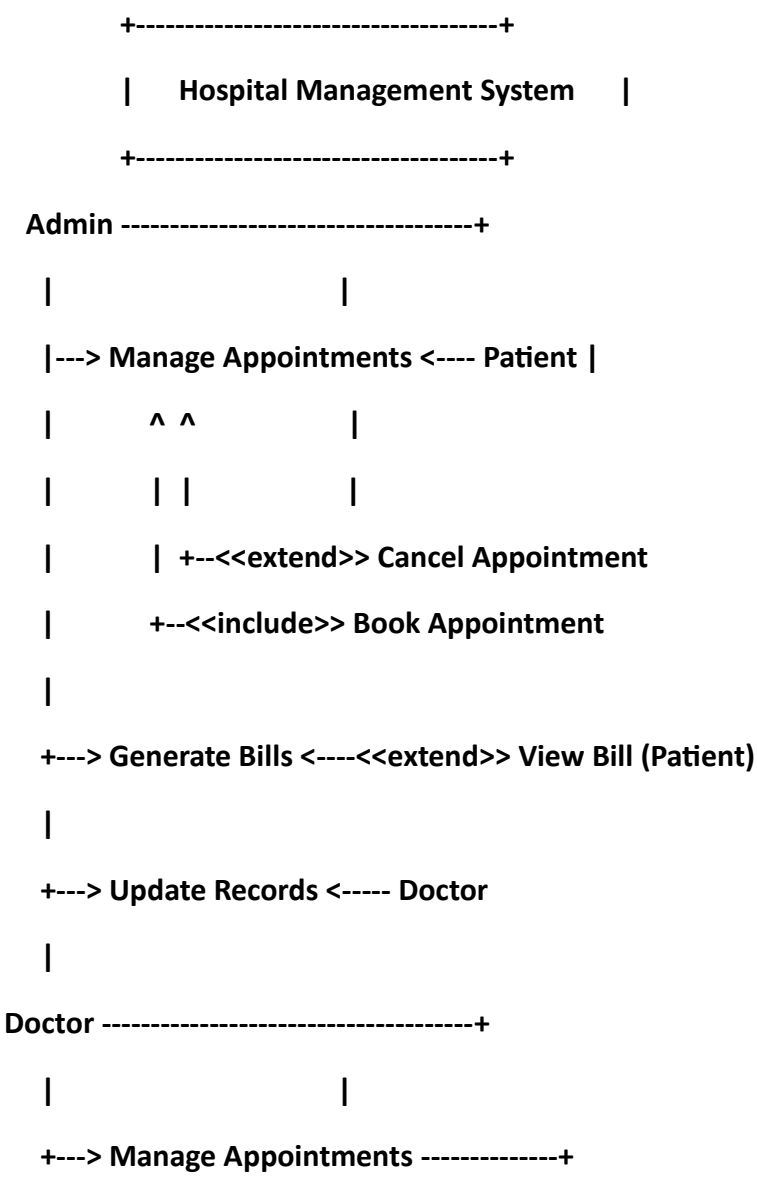
Problem Statement 4:

Create a Use Case Diagram for a Hospital Management System where an Admin, Doctor, and Patient interact with features like Manage Appointments, Update Records, and Generate Bills.

Hints:

- Use actor symbols for external users.
- Draw include and extend relationships between use cases.
- Label all relationships clearly.

Use Case Diagram (Text Representation)



+---> Update Records

|

Patient ----------+

+---> Manage Appointments

+---> View Bill

HospitalManagementUseCase.java

```java
import java.util.*;

class Appointment {
    String patientName;
    String doctorName;
    String date;

    Appointment(String patientName, String doctorName, String date) {
        this.patientName = patientName;
        this.doctorName = doctorName;
        this.date = date;
    }

    void display() {
        System.out.println("Appointment: " + patientName + " with Dr. " + doctorName + " on " + date)
    }
}

class Patient {
    String name;

    Patient(String name) {
        this.name = name;
    }

    void bookAppointment(List<Appointment> appointments, String doctorName, String date) {
        Appointment a = new Appointment(this.name, doctorName, date);
        appointments.add(a);
        System.out.println(name + " booked an appointment with Dr. " + doctorName + " on " + date);
    }

    void viewBill(String billDetails) {
        System.out.println(name + " views bill: " + billDetails);
```

```java
class Doctor {
    String name;

    Doctor(String name) {
        this.name = name;
    }

    void updateRecords(String patientName, String record) {
        System.out.println("Dr. " + name + " updated " + patientName + "'s record: " + record);
    }
}

class Admin {
    String name;

    Admin(String name) {
        this.name = name;
    }

    void generateBill(Patient patient, double amount) {
        String bill = "Bill for " + patient.name + ": ₹" + amount;
        System.out.println(name + " generated " + bill);
        patient.viewBill(bill);
    }

    void cancelAppointment(List<Appointment> appointments, Appointment a) {
        if (appointments.remove(a)) {
            System.out.println(name + " canceled appointment of " + a.patientName);
        }
    }
}
```

```java
public class HospitalManagementUseCase {
    Run main | Debug main
    public static void main(String[] args) {
        List<Appointment> appointments = new ArrayList<>();

        // Create actors
        Admin admin = new Admin("Admin John");
        Doctor doctor = new Doctor("Smith");
        Patient patient = new Patient("Alice");

        // Patient books an appointment (included in Manage Appointments)
        patient.bookAppointment(appointments, doctor.name, "2025-10-25");

        // Doctor updates patient's record
        doctor.updateRecords(patient.name, "Blood test results updated.");

        // Admin generates bill (extended by View Bill)
        admin.generateBill(patient, 2500.00);

        // Admin cancels appointment (extend example)
        if (!appointments.isEmpty()) {
            admin.cancelAppointment(appointments, appointments.get(0));
        }
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 10\Assignme
nt HW\Prorgam 4> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Week
s\Week 10\Assignment HW\Prorgam 4\" ; if ($?) { javac HospitalManagementUseCase.java } ; if ($?) { java HospitalManagement
UseCase }
Alice booked an appointment with Dr. Smith on 2025-10-25
Dr. Smith updated Alice's record: Blood test results updated.
Admin John generated Bill for Alice: INR. 2500.0
Alice views bill: Bill for Alice: INR. 2500.0
Admin John canceled appointment of Alice
```