Title: Online Bank Management System

- **Subtitle:** Ex. 6 OO design – Use case Model, Class Model using ArgoUML
- **Name:** Karan Sehgal
- **Registration No:** 22BCE3939
- **Team No: 24**
- **Course/Subject:** Software Engineering Lab (BCSE301P)
- **Instructor's Name:** Dr. Mehfooza M

Date of Submission: 12/02/25

Use Case Diagram

1. Introduction

The Online Bank Management System is designed to provide banking services to customers, manage employee operations, and handle transactions securely. This documentation outlines the use cases, actors, and their relationships within the system.

2. Actors

Primary Actors:

1. **Customer** - Performs banking activities like transactions, loan applications, and account management.
2. **Bank Employee** - Manages loans, approvals, and customer support.
3. **Admin** - Oversees employee management and system logs.
4. **Notification System** - Sends alerts and updates to customers.
5. **External Payment Gateway** - Facilitates secure online transactions.

3. Use Cases and Relationships

Authentication & Account Management

- **Login/Authentication** (Customer) - Customers authenticate before accessing services.
    - Includes: **Authenticate** (Validates credentials)
    - Extends: **Forgot Password** (Handles password recovery)
- **User Registration** (Customer) - Registers a new user.
- **View Account Details** (Customer) - Allows viewing of account information.

Transaction Management

- **Transaction** (Customer, External Payment Gateway) - Manages financial transactions.
    - Includes: **Credit** (Deposits money)
    - Includes: **Debit** (Withdraws money)
    - Includes: **Update Account Balance** (Reflects new balance after transactions)

- **Transfer Funds** (Customer) - Moves money between accounts.
  - Includes: **Credit** and **Debit**
  - Extends: **Send Notifications** (Alerts customers of the transfer)
- **Cancel Transaction** (Customer, Bank Employee) - Allows transaction reversals.
- **Dispute Transaction** (Customer, Bank Employee) - Enables customers to dispute erroneous transactions.

Loan Management

- **Loan** (Customer, Bank Employee) - Handles loan-related processes.
  - Includes: **Request Loan** (Customer applies for a loan)
  - Includes: **Credit Score Check** (Verifies eligibility)
  - Includes: **Term of the Loan** (Defines repayment terms)
  - Extends: **Approve Loan** (Handled by Bank Employee)
  - Extends: **Send Notifications** (Alerts customer about loan status)

System Management

- **Employee Management** (Admin) - Handles bank employee records.
- **System Logs** (Admin) - Maintains security logs.

Notifications

- **Send Notifications** (Notification System) - Sends transaction, loan, and security-related alerts.

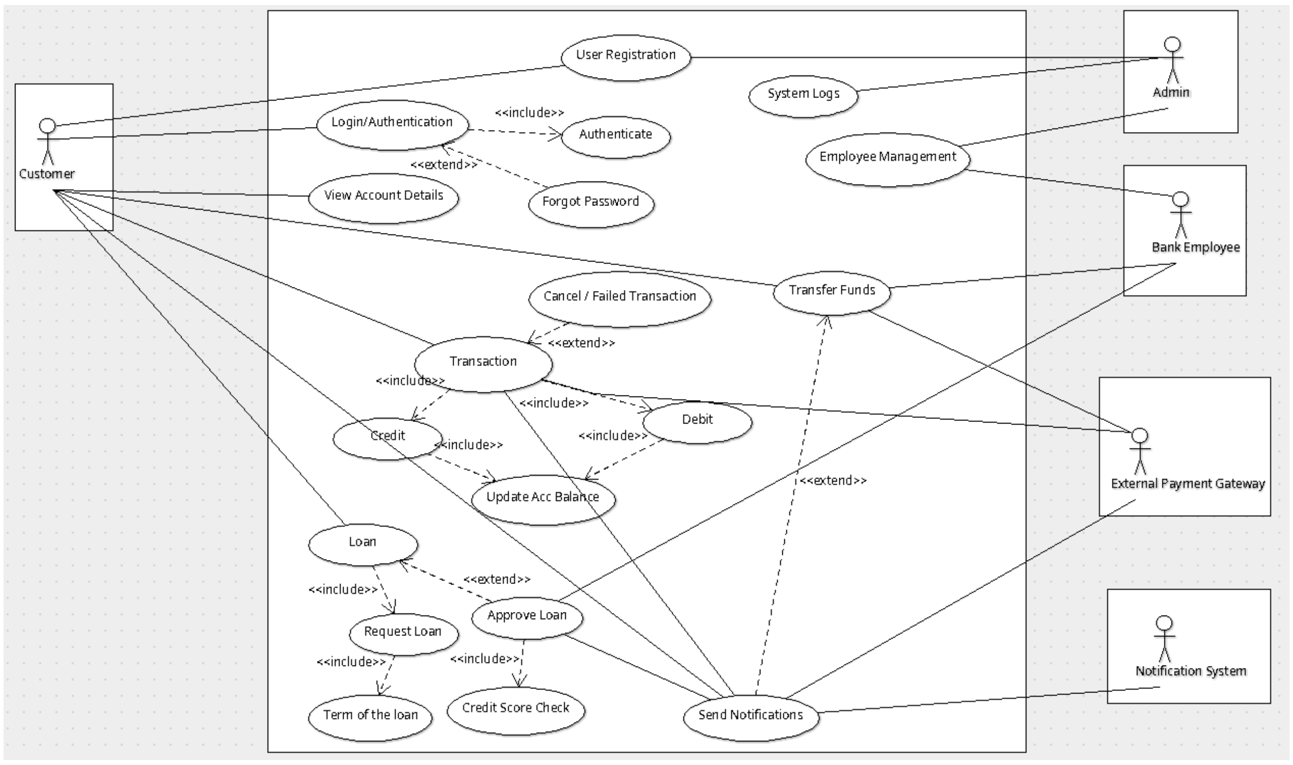
4. Improvements & Justifications

- **Stronger Actor-Use Case Links** - Explicitly connected Bank Employee and External Payment Gateway to relevant financial operations.
- **Detailed Loan Processing** - Enhanced with Credit Score Checks and Loan Terms.
- **Transaction Safety Measures** - Introduced "Cancel Transaction" and "Dispute Transaction" for error resolution.
- **Automated Notifications** - Ensures users receive updates on critical actions.

5. Conclusion

This use case model provides a structured approach to online banking, ensuring clarity in functionality and interactions between actors. The refinements help strengthen security, customer support, and system efficiency.

Diagram:



Class Diagram

Overview

This class diagram represents the structure and relationships between various entities in a Bank Management System. The main components include customers, employees, transactions, loans, accounts, notifications, and an external payment gateway. The system facilitates user interactions such as account management, fund transfers, loans, notifications, and transaction processing.

Classes and Their Descriptions

1. Customer

- **Attributes:**
  - `customerID: int` (Unique identifier for the customer)
  - `name: string` (Customer's full name)
  - `email: string` (Email ID for communication)
  - `phoneNumber: string` (Contact number of the customer)
  - `address: string` (Residential address)
  - `accountNumber: int` (Linked bank account number)
  - `balance: int` (Current account balance)
  - `username: string` (Login username)
  - `password: string` (Login password)
- **Methods:**
  - `login(): void` (Logs in the customer)
  - `viewAccountDetails(): void` (Displays account details)
  - `depositFunds(amount: double): void` (Adds money to account)

        - `withdrawFunds(amount: double): void` (Withdraws money from account)
        - `transferFunds(targetAccount: int, amount: double): void` (Transfers funds to another account)
        - `applyForLoan(amount: double, tenure: int): void` (Requests a loan)
        - `updateProfile(): void` (Updates customer information)

2. Account

- **Attributes:**
    - `accountNumber: int` (Unique account number)
    - `accountType: string` (Type of account - Savings, Current, etc.)
    - `balance: double` (Account balance)
    - `status: string` (Active or Inactive status)
- **Methods:**
    - `getBalance(): double` (Retrieves current balance)
    - `updateBalance(amount: double): void` (Modifies account balance)
    - `closeAccount(): void` (Closes the account)

3. Employee

- **Attributes:**
    - `employeeID: int` (Unique ID of the employee)
    - `name: string` (Employee's name)
    - `role: string` (Job role)
    - `email: string` (Contact email)
    - `phoneNumber: string` (Contact number)

        - `department: string` (Department of employment)
    - **Methods:**
        - `manageTransactions(): void` (Oversees transactions)
        - `approveLoans(): void` (Approves or rejects loan applications)
        - `viewCustomerDetails(): void` (Retrieves customer information)

4. Admin (Inherits Employee)

- **Methods:**
    - `manageEmployees(): void` (Handles employee operations)
    - `viewSystemLogs(): void` (Views system activity logs)
    - `configureSettings(): void` (Updates system configurations)

5. Loan

- **Attributes:**
    - `loanID: int` (Unique loan identifier)
    - `customerID: int` (Associated customer ID)
    - `amount: double` (Loan amount)
    - `interestRate: double` (Applicable interest rate)
    - `tenure: int` (Loan repayment duration in months)
    - `status: string` (Loan approval status)
    - `newAttr: Integer` (Placeholder attribute)
- **Methods:**
    - `applyForLoan(): void` (Initiates a loan application)
    - `approveLoan(): void` (Processes loan approval)

- `rejectLoan(): void` (Declines the loan application)

6. Transaction

- **Attributes:**
  - `transactionID: int` (Unique identifier for transactions)
  - `transactionType: string` (Deposit, Withdrawal, Transfer, etc.)
  - `amount: double` (Transaction amount)
  - `date: DateTime` (Timestamp of the transaction)
  - `status: string` (Transaction status - Success, Pending, Failed)
- **Methods:**
  - `processTransaction(): void` (Executes transaction)
  - `validateTransaction(): boolean` (Checks transaction validity)
  - `reverseTransaction(): boolean` (Cancels a transaction if needed)

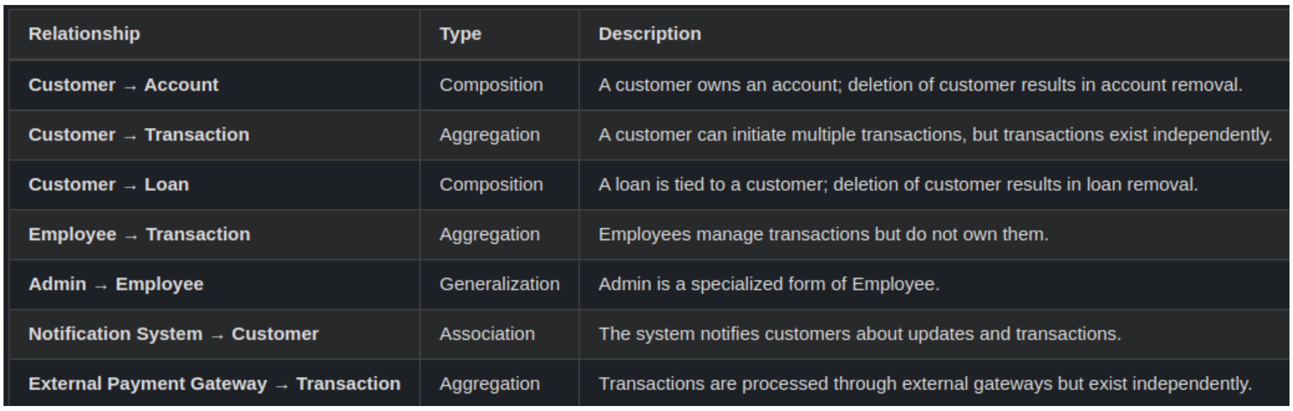
7. Notification System

- **Attributes:**
  - `notificationID: int` (Unique notification ID)
  - `message: string` (Notification message content)
  - `timestamp: DateTime` (Notification time)
  - `recipient: string` (Recipient of notification)
- **Methods:**
  - `sendNotification(): void` (Sends notification to customers)

8. Payment Gateway

- **Attributes:**
  - `gatewayID: int` (Unique identifier for gateway)
  - `providerName: string` (Third-party payment processor name)
  - `status: string` (Operational status)
- **Methods:**
  - `processPayment(): void` (Handles external transactions)

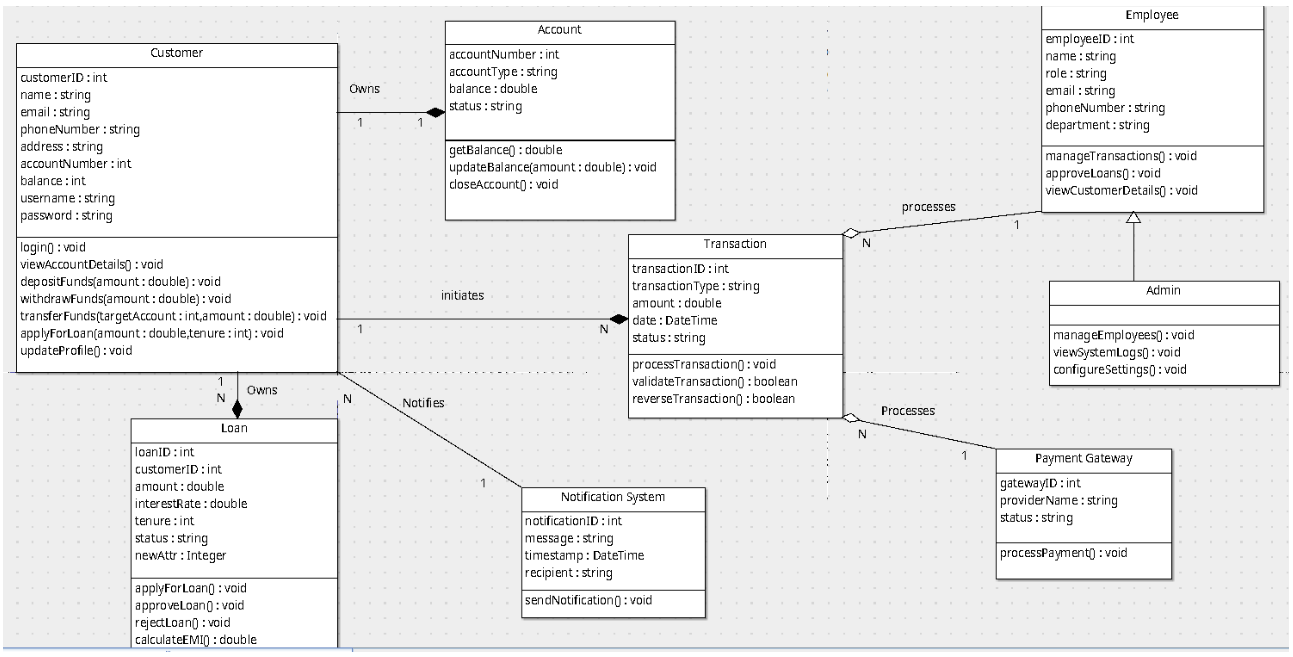
Relationships and Associations

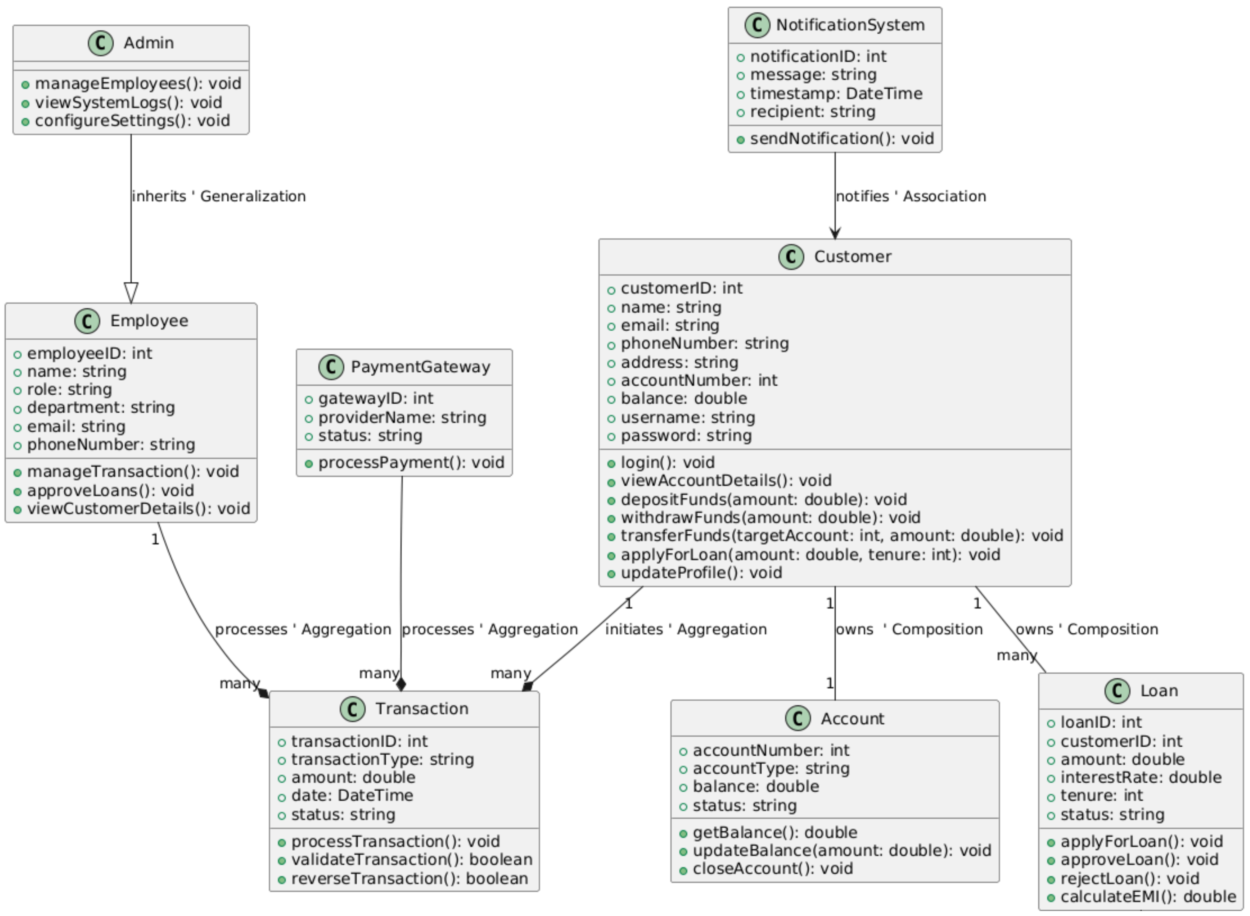


Relationship	Type	Description
Customer → Account	Composition	A customer owns an account; deletion of customer results in account removal.
Customer → Transaction	Aggregation	A customer can initiate multiple transactions, but transactions exist independently.
Customer → Loan	Composition	A loan is tied to a customer; deletion of customer results in loan removal.
Employee → Transaction	Aggregation	Employees manage transactions but do not own them.
Admin → Employee	Generalization	Admin is a specialized form of Employee.
Notification System → Customer	Association	The system notifies customers about updates and transactions.
External Payment Gateway → Transaction	Aggregation	Transactions are processed through external gateways but exist independently.

This class diagram effectively models the major functionalities of a banking system while maintaining scalability and modularity.

Diagram:





(for better readability I have attached web based plantuml output. Refer to above ArgoUML for specific relationships)

Code For PlantUML

```
@startuml
'    Customer Class
class Customer {
    + customerID: int
    + name: string
    + email: string
    + phoneNumber: string
    + address: string
    + accountNumber: int
    + balance: double
    + username: string
    + password: string
    + login(): void
    + viewAccountDetails(): void
    + depositFunds(amount: double): void
    + withdrawFunds(amount: double): void
    + transferFunds(targetAccount: int, amount: double): void
    + applyForLoan(amount: double, tenure: int): void
    + updateProfile(): void
}

'    Account Class
class Account {
    + accountNumber: int
    + accountType: string
    + balance: double
    + status: string
    + getBalance(): double
    + updateBalance(amount: double): void
    + closeAccount(): void
```

```
}

'   Loan Class
class Loan {
    + loanID: int
    + customerID: int
    + amount: double
    + interestRate: double
    + tenure: int
    + status: string
    + applyForLoan(): void
    + approveLoan(): void
    + rejectLoan(): void
    + calculateEMI(): double
}

'   Transaction Class
class Transaction {
    + transactionID: int
    + transactionType: string
    + amount: double
    + date: DateTime
    + status: string
    + processTransaction(): void
    + validateTransaction(): boolean
    + reverseTransaction(): boolean
}

'   Employee Class
class Employee {
    + employeeID: int
    + name: string
    + role: string
    + department: string
    + email: string
    + phoneNumber: string
    + manageTransaction(): void
    + approveLoans(): void
    + viewCustomerDetails(): void
}

'    Admin Class
class Admin {
    + manageEmployees(): void
    + viewSystemLogs(): void
    + configureSettings(): void
}

'    Notification System Class
class NotificationSystem {
    + notificationID: int
    + message: string
    + timestamp: DateTime
    + recipient: string
    + sendNotification(): void
}

'    Payment Gateway Class
class PaymentGateway {
    + gatewayID: int
    + providerName: string
    + status: string
    + processPayment(): void
}

'    Relationships
Customer "1" -- "1" Account : owns  ' Composition
Customer "1" --* "many" Transaction : initiates ' Aggregation
Customer "1" -- "many" Loan : owns ' Composition
Employee "1   " --*    "many" Transaction : processes ' Aggregation
Admin --|> Employee : inherits ' Generalization
NotificationSystem --> Customer : notifies ' Association
PaymentGateway --*    "many" Transaction : processes ' Aggregation
@enduml
```