**Introduction**

The banking sector plays a pivotal role in the global economy, facilitating financial transactions, providing various financial services, and serving as a crucial pillar for economic stability and growth. In the modern era, with the advent of technology and changing consumer preferences, the banking industry has undergone significant transformation, leading to the emergence of innovative business models and customer-centric approaches.

In this assignment, we delve into the realm of banking management by conceptualizing a comprehensive business model that encapsulates the core functionalities and operations of a banking institution. Our scenario revolves around the establishment of a fictitious banking entity aimed at catering to the diverse financial needs of individuals and businesses.

The primary objective of our banking management business model is to design and implement an efficient, secure, and user-friendly platform that enables customers to perform various banking activities seamlessly. From account creation and fund management to transaction processing and customer service, our application encompasses a wide array of features to deliver a holistic banking experience.

**Scenario**

In our banking management system, we aim to provide a comprehensive solution for managing various aspects of banking operations. The system is designed to cater to the needs of both customers and bank employees, facilitating seamless interactions and efficient management of financial services.

**Customer Perspective:**

Customers can access a range of services through our banking management system, including account management, transaction handling, and loan processing. They have the ability to create new bank accounts, deposit and withdraw funds, apply for loans, and view transaction history. This empowers customers to manage their finances effectively and conveniently.

**Bank Employee Perspective:**

Bank employees play a crucial role in the management and administration of banking operations. They have access to advanced functionalities such as customer management, branch management, and transaction handling. Employees can add and remove customers, manage branch information, perform transactions on behalf of customers, and assist with loan processing.

**Key Features:**

**Account Management**: Customers can create new bank accounts, deposit and withdraw funds, and apply interest to savings accounts.

**Customer Management:** Bank employees can add new customers, remove existing customers, and display customer details as needed.

**Branch Management:** Employees can manage branch information, including displaying branch details and finding the nearest branch location.

**Transaction Handling:** Both customers and employees can perform transactions, including deposits and withdrawals, with accurate tracking of transaction history.

**Loan Processing:** Customers can apply for loans with specified amounts, interest rates, and

**Design**

Our banking management system is designed to be robust, scalable, and user-friendly, providing a comprehensive solution for managing various banking operations. The design incorporates multiple classes that encapsulate different functionalities and interactions within the system. Below is a brief overview of the classes and their relationships:

**Main Class (AccountDriver):**

Serves as the entry point for the application.

Manages user interactions and controls the flow of the banking management system.

**Parent Class: Account**

Represents a generic bank account.

Contains attributes and methods common to all types of accounts.

Manages transactions and account balance.

**Child Classes: SavingAccount, CheckingAccount**

Inherit from the Account class.

Implement specific functionalities for saving and checking accounts, such as interest calculation and transaction fees.

**Group Class: Customer**

Manages customer information and account associations.

Contains methods to add, remove, and display customer accounts.

Additional Classes: Employee, Loan, Transaction, BranchLocator, CSVFile

**Employee:** Manages employee information and customer interactions.

**Loan:** Represents a loan associated with a customer's account.

**Transaction:** Tracks transaction history for accounts.

**BranchLocator:** Provides utility methods for locating branch locations.

**CSVFile:** Handles reading and writing data to a CSV file for data storage.

**Class Relationships:**

**Account:**

* Has a one-to-many relationship with Transaction and Customer.
* This means that an account can have multiple transactions associated with it, and it belongs to a single customer.
* SavingAccount and CheckingAccount extend from Account, indicating an "is-a" relationship. This means that both saving and checking accounts are specialized types of accounts.

**Customer:**

* Has a one-to-many relationship with Account.
* This means that a customer can have multiple accounts associated with them.

**Employee:**

* Interacts with Customer to manage customers and perform transactions.

**Loan:**

* Is associated with Account in a one-to-one relationship.
* This means that a loan is linked to a single account, indicating that each account can have at most one associated loan.

**Transaction:**

* Tracks transaction history within accounts.

**BranchLocator and CSVFile:**

* Provide utility functions without direct relationships with other classes. They are standalone utility classes used for specific functionalities.

**Code**

In this section, we provide the complete implementation of our banking management system. The system is designed to handle various banking operations including account management, transaction handling, loan management, and employee operations. The code is organized into multiple classes, each serving a specific purpose in the overall system. Below is an outline of the classes and their functionalities:

**AccountDriver.java**

package NewBusinessModel;

import java.util.Scanner;

public class AccountDriver {

public static void main(String [] args) {

Scanner keyboard = new Scanner(System.in);

Account accounts[] = new Account[10];

int numAccounts = 0;

int choice;

do {

System.out.println("Welcome to the Bank!");

System.out.println("Are you a Customer or an Employee?");

System.out.println("1. Customer");

System.out.println("2. Employee");

System.out.print("Enter your choice: ");

choice = keyboard.nextInt();

System.out.println();

if (choice == 1) {

handleCustomerActions(accounts, numAccounts, keyboard);

} else if (choice == 2) {

handleEmployeeActions(keyboard);

} else {

System.out.println("Invalid choice. Please try again.");

}

System.out.println();

} while (choice != 1 && choice != 2); // Continue until a valid choice is made

}

public static void handleCustomerActions(Account accounts[], int numAccounts, Scanner keyboard) {

int choice;

do {

choice = menu(keyboard);

System.out.println();

switch (choice) {

case 1:

accounts[numAccounts++] = createAccount(keyboard);

break;

case 2:

doDeposit(accounts, numAccounts, keyboard);

break;

case 3:

doWithdraw(accounts, numAccounts, keyboard);

break;

case 4:

applyInterest(accounts, numAccounts, keyboard);

break;

case 5:

findNearestBranch();

break;

case 6:

applyForLoan(accounts, numAccounts, keyboard);

break;

case 7:

makeLoanPayment(accounts, numAccounts, keyboard);

break;

case 8:

System.out.println("Goodbye!");

break;

default:

System.out.println("Invalid choice. Please try again.");

break;

}

System.out.println();

} while (choice != 8);

}

public static void handleEmployeeActions(Scanner keyboard) {

System.out.println("Employee Panel:");

System.out.println("1. Manage Customers");

System.out.println("2. Manage Branches");

System.out.println("3. Quit");

int choice;

do {

System.out.print("Enter your choice: ");

choice = keyboard.nextInt();

switch (choice) {

case 1:

manageCustomers();

break;

case 2:

manageBranches();

break;

case 3:

System.out.println("Goodbye!");

break;

default:

System.out.println("Invalid choice. Please try again.");

break;

}

System.out.println();

} while (choice != 3);

}

public static int menu(Scanner keyboard) {

System.out.println("Bank Account Menu");

System.out.println("1. Create New Account");

System.out.println("2. Deposit Funds");

System.out.println("3. Withdraw Funds");

System.out.println("4. Apply Interest");

System.out.println("5. Find Nearest Branch");

System.out.println("6. Apply for a Loan");

System.out.println("7. Make a Loan Payment");

System.out.println("8. Quit");

int choice;

do {

System.out.print("Enter choice: ");

choice = keyboard.nextInt();

} while (choice < 1 || choice > 8);

return choice;

}

public static Account createAccount(Scanner keyboard) {

SavingAccount account = null;

int choice = accountMenu(keyboard);

int accountNumber;

System.out.print("Enter Account Number: ");

accountNumber = keyboard.nextInt();

if (choice == 1) {

System.out.print("Enter Transaction Fee: ");

double fee = keyboard.nextDouble();

account = new SavingAccount(accountNumber, fee);

} else {

System.out.print("Please enter Interest Rate: ");

double ir = keyboard.nextDouble();

account = new SavingAccount(accountNumber, ir);

}

return account;

}

public static int accountMenu(Scanner keyboard) {

System.out.println("Select Account Type");

System.out.println("1. Checking Account");

System.out.println("2. Savings Account");

int choice;

do {

System.out.print("Enter choice: ");

choice = keyboard.nextInt();

} while (choice < 1 || choice > 2);

return choice;

}

public static int searchAccount(Account accounts[], int count, int accountNumber) {

for (int i = 0; i < count; i++) {

if (accounts[i].getAccountNumber() == accountNumber) {

return i;

}

}

return -1;

}

public static void doDeposit(Account accounts[], int count, Scanner keyboard) {

System.out.print("\nPlease enter account number: ");

int accountNumber = keyboard.nextInt();

int index = searchAccount(accounts, count, accountNumber);

if (index >= 0) {

System.out.print("Please enter Deposit Amount: ");

double amount = keyboard.nextDouble();

accounts[index].deposit(amount);

} else {

System.out.println("No account exist with AccountNumber: " + accountNumber);

}

}

public static void doWithdraw(Account accounts[], int count, Scanner keyboard) {

System.out.print("\nPlease enter account number: ");

int accountNumber = keyboard.nextInt();

int index = searchAccount(accounts, count, accountNumber);

if (index >= 0) {

System.out.print("Please enter Withdraw Amount: ");

double amount = keyboard.nextDouble();

accounts[index].withdraw(amount);

} else {

System.out.println("No account exist with AccountNumber: " + accountNumber);

}

}

public static void applyInterest(Account accounts[], int count, Scanner keyboard) {

System.out.print("\nPlease enter account number: ");

int accountNumber = keyboard.nextInt();

int index = searchAccount(accounts, count, accountNumber);

if (index >= 0) {

if (accounts[index] instanceof SavingAccount) {

((SavingAccount)accounts[index]).applyInterest();

}

} else {

System.out.println("No account exist with AccountNumber: " + accountNumber);

}

}

public static void findNearestBranch() {

BranchLocator branchLocator = new BranchLocator();

branchLocator.displayAllBranches();

branchLocator.findNearestBranch();

}

public static void manageCustomers() {

// Implement method to manage customers

System.out.println("Managing Customers...");

}

public static void manageBranches() {

// Implement method to manage branches

System.out.println("Managing Branches...");

}

public static void applyForLoan(Account accounts[], int numAccounts, Scanner keyboard) {

System.out.print("\nPlease enter account number: ");

int accountNumber = keyboard.nextInt();

int index = searchAccount(accounts, numAccounts, accountNumber);

if (index >= 0) {

System.out.print("Enter loan amount: ");

double loanAmount = keyboard.nextDouble();

// Create a new loan and add it to the account

accounts[index].applyForLoan(loanAmount);

} else {

System.out.println("No account exists with Account Number: " + accountNumber);

}

}

public static void makeLoanPayment(Account accounts[], int numAccounts, Scanner keyboard) {

System.out.print("\nPlease enter account number: ");

int accountNumber = keyboard.nextInt();

int index = searchAccount(accounts, numAccounts, accountNumber);

if (index >= 0) {

System.out.print("Enter loan payment amount: ");

double paymentAmount = keyboard.nextDouble();

// Make a payment towards the loan

accounts[index].makeLoanPayment(paymentAmount);

} else {

System.out.println("No account exists with Account Number: " + accountNumber);

}

}

}

**Account.java**

package NewBusinessModel;

import java.util.ArrayList;

import java.util.List;

public class Account {

protected int accountNumber;

protected double balance;

protected static Customer customer; // Reference to the Customer object

protected List<Transaction> transactions; // List to store transactions

// Constructor

public Account(int accountNumber, Customer customer) {

this.accountNumber = accountNumber;

Account.customer = customer; // Initialize the customer

this.balance = 0.0; // Initially, balance is zero

this.transactions = new ArrayList<>(); // Initialize the transactions list

}

// Deposit funds into the account

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

transactions.add(new Transaction("deposit", amount, accountNumber));

System.out.println("Deposit successful.");

} else {

System.out.println("Invalid deposit amount.");

}

}

// Withdraw funds from the account

public void withdraw(double amount) {

if (amount > 0 && balance >= amount) {

balance -= amount;

transactions.add(new Transaction("withdraw", amount, accountNumber));

System.out.println("Withdrawal successful.");

} else {

System.out.println("Invalid withdrawal amount or insufficient funds.");

}

}

// Get the account number

public int getAccountNumber() {

return accountNumber;

}

// Get the current balance of the account

public double getBalance() {

return balance;

}

// Get the customer associated with the account

public Customer getCustomer() {

return customer;

}

// String representation of the account

@Override

public String toString() {

return "Account Number: " + accountNumber + ", Balance: $" + balance;

}

public String getBranchLocation() {

// Return a default branch location for now

return "Main Branch";

}

// Display transaction history

public void displayTransactions() {

System.out.println("Transaction History for Account " + accountNumber + ":");

for (Transaction transaction : transactions) {

System.out.println(transaction);

}

}

public void makeLoanPayment(double paymentAmount) {

// Placeholder for making loan payment

System.out.println("Making loan payment of $" + paymentAmount);

}

public void applyForLoan(double loanAmount) {

// Placeholder for applying for a loan

System.out.println("Applying for a loan of $" + loanAmount);

}

public int getLoanId() {

// Placeholder for getting loan ID

return 12345;

}

public double getLoanAmount() {

// Placeholder for getting loan amount

return 5000.0;

}

}

**SavingAccount.java**

package NewBusinessModel;

public class SavingAccount extends Account {

private double interestRate;

public SavingAccount(int accountNumber, double interestRate) {

super(accountNumber, customer);

this.interestRate = interestRate;

}

public double getInterestRate() {

return interestRate;

}

public void setInterestRate(double interestRate) {

this.interestRate = interestRate;

}

public double calcInterest() {

return balance \* interestRate;

}

public void applyInterest() {

double interest = calcInterest();

System.out.printf("Interest amount %.2f added to balance%n", interest);

deposit(interest);

}

@Override

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

transactions.add(new Transaction("Deposit", amount, accountNumber)); // Record deposit transaction

System.out.printf("Amount %.2f deposited%n", amount);

System.out.printf("Current Balance is: %.2f%n", balance);

} else {

System.out.println("A negative amount cannot be deposited");

}

}

@Override

public void withdraw(double amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

transactions.add(new Transaction("Withdrawal", amount, accountNumber)); // Record withdrawal transaction

System.out.printf("Amount of %.2f withdrawn from Account%n", amount);

System.out.printf("Current Balance is: %.2f%n", balance);

} else {

System.out.println("Invalid amount or insufficient balance");

}

}

}

**CheckingAccount.java**

package NewBusinessModel;

public class CheckingAccount extends Account {

private double transactionFee;

public CheckingAccount(int accountNumber, double transactionFee) {

super(accountNumber, customer);

this.transactionFee = transactionFee;

}

public double getTransactionFee() {

return transactionFee;

}

public void setTransactionFee(double transactionFee) {

this.transactionFee = transactionFee;

}

@Override

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

transactions.add(new Transaction("Deposit", amount, accountNumber)); // Record deposit transaction

System.out.printf("Amount %.2f deposited%n", amount);

System.out.printf("Current Balance is: %.2f%n", balance);

} else {

System.out.println("A negative amount cannot be deposited");

}

}

@Override

public void withdraw(double amount) {

if (amount > 0 && amount + transactionFee <= balance) {

balance -= (amount + transactionFee);

transactions.add(new Transaction("Withdrawal", amount, accountNumber)); // Record withdrawal transaction

System.out.printf("Amount of %.2f withdrawn from Account%n", amount);

System.out.printf("Current Balance is: %.2f%n", balance);

} else {

System.out.println("Invalid amount or insufficient balance");

}

}

}

**Customer.java**

package NewBusinessModel;

import java.util.ArrayList;

import java.util.List;

public class Customer {

private int customerId;

private String name;

private String address;

private String contactDetails;

private List<Account> accounts;

private List<Loan> loans;

public Customer(int customerId, String name, String address, String contactDetails) {

this.customerId = customerId;

this.name = name;

this.address = address;

this.contactDetails = contactDetails;

this.accounts = new ArrayList<>();

this.loans = new ArrayList<>();

}

// Getter and setter methods for customer attributes

public int getCustomerId() {

return customerId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getContactDetails() {

return contactDetails;

}

public void setContactDetails(String contactDetails) {

this.contactDetails = contactDetails;

}

// Method to add an account to the customer's list of accounts

public void addAccount(Account account) {

accounts.add(account);

}

// Method to remove an account from the customer's list of accounts

public void removeAccount(Account account) {

accounts.remove(account);

}

// Method to get a list of all accounts associated with the customer

public List<Account> getAccounts() {

return accounts;

}

// New: Method to add a loan to the customer's list of loans

public void addLoan(Loan loan) {

loans.add(loan);

}

// New: Method to remove a loan from the customer's list of loans

public void removeLoan(Loan loan) {

loans.remove(loan);

}

// New: Method to get a list of all loans associated with the customer

public List<Loan> getLoans() {

return loans;

}

// Method to display customer information

@Override

public String toString() {

return "Customer ID: " + customerId + "\nName: " + name + "\nAddress: " + address + "\nContact Details: " + contactDetails;

}

// Method to display transaction history for all accounts associated with the customer

public void displayTransactionHistory() {

System.out.println("Transaction History for Customer: " + name);

for (Account account : accounts) {

account.displayTransactions();

}

}

// Method to perform a transaction on a specific account associated with the customer

public void performTransaction(Account account, double amount, String transactionType) {

if (transactionType.equalsIgnoreCase("deposit")) {

account.deposit(amount);

} else if (transactionType.equalsIgnoreCase("withdraw")) {

account.withdraw(amount);

} else {

System.out.println("Invalid transaction type.");

}

}

}

**Transaction.java**

package NewBusinessModel;

import java.time.LocalDateTime;

public class Transaction {

private String type;

private double amount;

private LocalDateTime timestamp;

private int loanId; // New field to specify loan ID

// Constructor

public Transaction(String type, double amount, int loanId) {

this.type = type;

this.amount = amount;

this.timestamp = LocalDateTime.now(); // Record the current timestamp

this.loanId = loanId; // Set loan ID

}

// Getter methods

public String getType() {

return type;

}

public double getAmount() {

return amount;

}

public LocalDateTime getTimestamp() {

return timestamp;

}

public int getLoanId() {

return loanId;

}

// String representation of the transaction

@Override

public String toString() {

return "Type: " + type + ", Amount: $" + amount + ", Timestamp: " + timestamp + ", Loan ID: " + loanId;

}

}

**Employee.java**

package NewBusinessModel;

import java.util.List;

public class Employee {

private int employeeID;

private String name;

private String role;

// Constructor

public Employee(int employeeID, String name, String role) {

this.employeeID = employeeID;

this.name = name;

this.role = role;

}

// Getter and setter methods for employee attributes

public int getEmployeeID() {

return employeeID;

}

public void setEmployeeID(int employeeID) {

this.employeeID = employeeID;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getRole() {

return role;

}

public void setRole(String role) {

this.role = role;

}

// Additional methods

public void performDuties() {

// Implement method to perform employee duties

System.out.println(name + " is performing duties as " + role);

}

// Method to add a new customer

public void addCustomer(List<Customer> customers, Customer customer) {

customers.add(customer);

System.out.println("Customer added successfully.");

}

// Method to remove a customer

public void removeCustomer(List<Customer> customers, Customer customer) {

customers.remove(customer);

System.out.println("Customer removed successfully.");

}

// Method to display all customers

public void displayCustomers(List<Customer> customers) {

System.out.println("List of Customers:");

for (Customer customer : customers) {

System.out.println(customer);

}

}

// Method to perform a transaction for a specific customer and account

public void performTransaction(Customer customer, Account account, double amount, String transactionType) {

if (transactionType.equalsIgnoreCase("deposit")) {

account.deposit(amount);

customer.performTransaction(account, amount, "deposit");

} else if (transactionType.equalsIgnoreCase("withdraw")) {

account.withdraw(amount);

customer.performTransaction(account, amount, "withdraw");

} else {

System.out.println("Invalid transaction type.");

}

}

public void addAccount(Account account) {

// Placeholder for adding account

System.out.println("Adding account: " + account);

}

}

**Loan.java**

package NewBusinessModel;

import java.time.LocalDate;

public class Loan {

private int loanId;

private double amount;

private double interestRate;

private LocalDate startDate;

private LocalDate endDate;

private LoanStatus status;

private Customer customer;

private Account account;

public Loan(int loanId, double amount, double interestRate, LocalDate startDate, int durationMonths, Customer customer, Account account) {

this.loanId = loanId;

this.amount = amount;

this.interestRate = interestRate;

this.startDate = startDate;

this.endDate = startDate.plusMonths(durationMonths);

this.status = LoanStatus.ACTIVE;

this.customer = customer;

this.account = account;

}

// Method to calculate monthly payment amount

public double calculateMonthlyPayment() {

double monthlyInterestRate = interestRate / 12 / 100;

int totalMonths = (int) Math.ceil(LocalDate.now().until(endDate).getDays() / 30.44); // Assuming 30.44 days in a month

return (amount \* monthlyInterestRate) / (1 - Math.pow(1 + monthlyInterestRate, -totalMonths));

}

// Method to make a loan payment

public void makePayment(double paymentAmount) {

if (paymentAmount > 0 && status == LoanStatus.ACTIVE) {

account.withdraw(paymentAmount);

if (account.getBalance() <= 0) {

status = LoanStatus.PAID\_OFF;

}

System.out.println("Payment of $" + paymentAmount + " made successfully.");

} else {

System.out.println("Invalid payment amount or loan is already paid off.");

}

}

// Method to get loan status

public LoanStatus getStatus() {

return status;

}

// Method to get remaining balance

public double getRemainingBalance() {

return (status == LoanStatus.ACTIVE) ? calculateRemainingBalance() : 0.0;

}

// Method to calculate remaining balance

private double calculateRemainingBalance() {

double monthlyPayment = calculateMonthlyPayment();

int totalMonthsPaid = (int) Math.ceil(startDate.until(LocalDate.now()).getDays() / 30.44); // Assuming 30.44 days in a month

double interestPaid = totalMonthsPaid \* monthlyPayment - amount;

return amount - interestPaid;

}

// Method to get loan details

@Override

public String toString() {

return "Loan ID: " + loanId +

"\nAmount: $" + amount +

"\nInterest Rate: " + interestRate + "%" +

"\nStart Date: " + startDate +

"\nEnd Date: " + endDate +

"\nStatus: " + status +

"\nCustomer: " + customer.getName();

}

// Enum to represent loan status

public enum LoanStatus {

ACTIVE,

PAID\_OFF

}

}

**BranchLocator.java**

package NewBusinessModel;

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class BranchLocator {

private Scanner scanner;

private Map<Integer, Branch> branches;

public BranchLocator() {

this.scanner = new Scanner(System.in);

this.branches = new HashMap<>();

initializeBranches();

}

// Initialize branch information

private void initializeBranches() {

branches.put(1, new Branch("Dhaka", "Dhaka, Bangladesh", "+8801XXXXXXXXX"));

branches.put(2, new Branch("Chittagong", "Chittagong, Bangladesh", "+8801XXXXXXXXX"));

branches.put(3, new Branch("Rajshahi", "Rajshahi, Bangladesh", "+8801XXXXXXXXX"));

branches.put(4, new Branch("Khulna", "Khulna, Bangladesh", "+8801XXXXXXXXX"));

branches.put(5, new Branch("Barisal", "Barisal, Bangladesh", "+8801XXXXXXXXX"));

// Add more branches as needed

}

// Display information for all branches

public void displayAllBranches() {

System.out.println("\t===================================");

System.out.println("\t\t Banking Management Branch Locator");

System.out.println("\t===================================");

System.out.println("\t'-->Navigate by inserting index<--'");

for (Map.Entry<Integer, Branch> entry : branches.entrySet()) {

System.out.println("\t" + entry.getKey() + ". " + entry.getValue().getName());

}

System.out.println("\t===================================");

}

// Display information for the nearest branch based on user's current location

public void findNearestBranch() {

System.out.println("\tInsert number of your current location:");

int userChoice = scanner.nextInt();

scanner.nextLine(); // Consume newline

Branch nearestBranch = branches.get(userChoice);

if (nearestBranch != null) {

displayBranchInformation(nearestBranch);

handleBranchTransaction(nearestBranch); // Simulate branch transaction

} else {

System.out.println("\tInvalid choice!");

} }

// Display branch information

public void displayBranchInformation(Branch branch) {

System.out.println("\t===================================");

System.out.println("\t\t" + branch.getName() + " Branch");

System.out.println("\t===================================");

System.out.println("\tAddress: " + branch.getAddress());

System.out.println("\tCall: " + branch.getContact());

System.out.println("\t===================================");

}

// Simulate branch transaction (e.g., branch visit or activity)

public void handleBranchTransaction(Branch branch) {

// Add code here to simulate branch transaction, such as logging branch visit or activity

System.out.println("\tBranch transaction simulated: Visit to " + branch.getName() + " Branch");

}

public static void main(String[] args) {

BranchLocator branchLocator = new BranchLocator();

branchLocator.displayAllBranches();

branchLocator.findNearestBranch();

}

// Inner class representing a Branch

private static class Branch {

private String name;

private String address;

private String contact;

public Branch(String name, String address, String contact) {

this.name = name;

this.address = address;

this.contact = contact;

}

public String getName() {

return name;

}

public String getAddress() {

return address;

}

public String getContact() {

return contact;

}

}

}

**CSVFile.java**

package NewBusinessModel;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.List;

public class CSVFile {

private static final String CSV\_FILE\_PATH = "BankingData.csv";

public static void writeToCSV(List<Account> accounts) {

try (PrintWriter writer = new PrintWriter(new BufferedWriter(new FileWriter(CSV\_FILE\_PATH)))) {

// Write CSV header

writer.println("Account Number,Account Type,Balance,Customer Name,Customer ID,Address,Contact,Branch Location,Loan ID,Loan Type,Loan Amount");

// Write account data

for (Account account : accounts) {

String accountType = (account instanceof SavingAccount) ? "Savings" : "Checking";

String customerName = account.getCustomer().getName();

int customerId = account.getCustomer().getCustomerId();

String address = account.getCustomer().getAddress();

String contact = account.getCustomer().getContactDetails();

String branchLocation = account.getBranchLocation();

int loanId = account.getLoanId();

int loanType = account.getLoanId();

double loanAmount = account.getLoanAmount();

writer.println(account.getAccountNumber() + "," + accountType + "," + account.getBalance() + ","

+ customerName + "," + customerId + "," + address + "," + contact + "," + branchLocation + ","

+ loanId + "," + loanType + "," + loanAmount);

}

System.out.println("Data has been written to the CSV file successfully.");

} catch (IOException e) {

System.out.println("Error writing to the CSV file: " + e.getMessage());

}

}

public static void addAccountToCSV(Account account) {

try (PrintWriter writer = new PrintWriter(new BufferedWriter(new FileWriter(CSV\_FILE\_PATH, true)))) {

String accountType = (account instanceof SavingAccount) ? "Savings" : "Checking";

String customerName = account.getCustomer().getName();

int customerId = account.getCustomer().getCustomerId();

String address = account.getCustomer().getAddress();

String contact = account.getCustomer().getContactDetails();

String branchLocation = account.getBranchLocation();

int loanId = account.getLoanId();

int loanType = account.getLoanId();

double loanAmount = account.getLoanAmount();

writer.println(account.getAccountNumber() + "," + accountType + "," + account.getBalance() + ","

+ customerName + "," + customerId + "," + address + "," + contact + "," + branchLocation + ","

+ loanId + "," + loanType + "," + loanAmount);

System.out.println("New account has been added to the CSV file successfully.");

} catch (IOException e) {

System.out.println("Error writing to the CSV file: " + e.getMessage());

}

}

// Add transaction information to CSV file

public static void addTransactionToCSV(int accountNumber, String transactionType, double amount) {

try (PrintWriter writer = new PrintWriter(new BufferedWriter(new FileWriter(CSV\_FILE\_PATH, true)))) {

writer.println(accountNumber + "," + transactionType + "," + amount);

System.out.println("Transaction logged successfully.");

} catch (IOException e) {

System.out.println("Error writing transaction to the CSV file: " + e.getMessage());

}

}

}

**Database**

**BankingData.csv**

Account Number,Account Type,Balance,Customer Name,Customer ID,Address,Contact,Branch Location,Loan ID,Loan Type,Loan Amount

1001001,Savings,5000.0,John Doe,10001,123 Main St,123-456-7890,Dhaka,101,Personal,2000.0

1001002,Checking,3000.0,Jane Smith,10002,456 Oak St,987-654-3210,Chittagong,,,

1001003,Savings,7000.0,Alice Johnson,10003,789 Elm St,111-222-3333,Rajshahi,,,

1001004,Savings,4000.0,Robert Lee,10004,321 Pine St,444-555-6666,Khulna,102,Home,5000.0

1001005,Checking,2000.0,Michael Brown,10005,567 Cedar St,333-222-1111,Barisal,,,

1001006,Savings,6000.0,Emily Wilson,10006,890 Walnut St,666-777-8888,Rajshahi,,,

1001007,Checking,3500.0,David Taylor,10007,901 Maple St,999-888-7777,Dhaka,,,

1001008,Savings,4500.0,Sarah Clark,10008,234 Birch St,777-888-9999,Chittagong,,,

1001009,Checking,3200.0,Lisa Martinez,10009,543 Elm St,555-666-7777,Barisal,103,Education,3000.0

1001010,Savings,5500.0,Mark Anderson,10010,678 Pine St,222-333-4444,Dhaka,,,

1001011,Checking,2800.0,Karen Thompson,10011,789 Maple St,888-999-0000,Chittagong,,,

1001012,Savings,6200.0,Christopher White,10012,456 Cedar St,111-222-3333,Barisal,,,

1001013,Checking,3900.0,Michelle Garcia,10013,890 Oak St,444-555-6666,Rajshahi,,,

1001014,Savings,4800.0,Daniel Hernandez,10014,123 Birch St,777-888-9999,Khulna,104,Personal,4000.0

1001015,Checking,4100.0,Tiffany Martinez,10015,678 Walnut St,333-444-5555,Dhaka,,,

1001016,Savings,5100.0,Matthew Lopez,10016,901 Pine St,666-777-8888,Chittagong,,,

1001017,Checking,3400.0,Amanda Perez,10017,234 Maple St,999-000-1111,Rajshahi,,,

1001018,Savings,4300.0,Rachel Gonzales,10018,543 Cedar St,222-333-4444,Barisal,,,

1001019,Checking,4700.0,Anthony Wilson,10019,678 Oak St,555-666-7777,Khulna,,,

1001020,Savings,5300.0,Jason Martinez,10020,890 Pine St,888-999-0000,Dhaka,105,Home,6000.0

1001021,Checking,3000.0,Kimberly Johnson,10021,123 Cedar St,111-222-3333,Chittagong,,,

1001022,Savings,6200.0,Joshua Davis,10022,456 Oak St,444-555-6666,Barisal,,,

1001023,Checking,3900.0,Ashley Martinez,10023,789 Elm St,777-888-9999,Rajshahi,,,

1001024,Savings,4800.0,Emily Hernandez,10024,901 Pine St,333-444-5555,Khulna,,,

1001025,Checking,4100.0,Justin Wilson,10025,234 Birch St,666-777-8888,Dhaka,,,