**Banking System Documentation**

**1. Introduction**

This program implements a banking management system to handle operations such as employee management, account handling, and branch operations. It uses object-oriented principles like inheritance, polymorphism, and aggregation to provide a flexible, scalable, and modular design.

**Purpose**

The main objective is to simulate a banking environment where branch and head office employees interact with accounts and customers under a centralized management system.

**Scope**

The system supports:

* Employee management (branch and head office employees).
* Account management (savings and current accounts).
* Branch-level and head-office-level operations.

**2. System Architecture**

**Class Overview**

The system is designed with the following primary classes:

* **Employee**: Base class for all employees.
* **BranchEmployee** and **HeadEmployee**: Derived classes to manage branch-specific and head office employees, respectively.
* **Account**: Base class for all bank accounts.
* **SavingsAccount** and **CurrentAccount**: Derived classes to manage specialized account types.
* **Bank**: Represents a branch, managing its employees and accounts.
* **HeadOffice**: Manages multiple branches and head office employees.

**Relationships**

* **Inheritance**:
  + Employee → BranchEmployee, HeadEmployee.
  + Account → SavingsAccount, CurrentAccount.
* **Aggregation**:
  + Bank manages a collection of employees and accounts.
  + HeadOffice aggregates multiple Bank instances and head office employees.

**3. Class Documentation**

**3.1 Employee (Base Class)**

* **Purpose**: Represents a generic employee with common attributes and methods.
* **Attributes**:
  + name (string): Name of the employee.
  + dob (int): Date of birth in ddmmyy format.
  + post (string): Job title.
  + salary (double): Monthly salary.
* **Methods**:
  + display(): Pure virtual function to be implemented by derived classes.

**3.2 BranchEmployee (Derived Class)**

* **Purpose**: Represents an employee assigned to a specific branch.
* **Attributes**:
  + branch (string): Name of the branch.
  + doj (int): Date of joining.
* **Methods**:
  + display(): Outputs the employee's details, including branch information.

**3.3 HeadEmployee (Derived Class)**

* **Purpose**: Represents an employee at the head office.
* **Attributes**:
  + doj (int): Date of joining.
* **Methods**:
  + display(): Outputs the head office employee's details.

**3.4 Account (Base Class)**

* **Purpose**: Represents a generic bank account.
* **Attributes**:
  + accountnumber (int): Unique identifier for the account.
  + password (string): Password for account access.
  + holdername (string): Name of the account holder.
  + balance (double): Current account balance.
* **Methods**:
  + deposit(Bank\*, double): Deposits a specified amount.
  + withdraw(double): Virtual function to handle withdrawals.
  + displayAccountDetails(): Displays account details.

**3.5 SavingsAccount (Derived Class)**

* **Purpose**: Specialized account with savings features.
* **Attributes**:
  + minBal (int): Minimum required balance.
  + overdraft (bool): Indicates overdraft capability.
  + rateOfInterest (int): Interest rate for the account.

**3.6 CurrentAccount (Derived Class)**

* **Purpose**: Specialized account for current (business) users.

**3.7 Bank**

* **Purpose**: Represents a branch of the bank.
* **Attributes**:
  + branch (string): Branch name.
  + IFSC (string): Unique branch code.
  + allBranchEmployees (vector): List of branch employees.
  + allSavingsAccounts (vector): List of savings accounts.
  + allCurrentAccounts (vector): List of current accounts.
* **Methods**:
  + addBranchEmployee(): Adds a new branch employee.
  + openSavingsAccount(), openCurrentAccount(): Handles account creation.
  + displayAllBranchEmployees(): Lists all branch employees.

**3.8 HeadOffice**

* **Purpose**: Represents the head office managing branches and employees.
* **Attributes**:
  + allBanks (vector): List of associated branches.
  + allHeadEmployees (vector): List of head office employees.
* **Methods**:
  + addBank(), addHeadEmployee(): Adds new entities.
  + displayAllBanks(): Displays details of all banks.
  + getBankByIFSC(): Retrieves a branch by its IFSC code.

**System Workflow**

This section describes how the different components of the banking system interact during typical operations. Here are the key workflows:

**1. Adding a New Branch Employee**

1. **Input**:
   * Employee details such as name, date of birth, post, salary, branch name, and date of joining.
2. **Process**:
   * Create a BranchEmployee object using the provided details.
   * Add the object to the allBranchEmployees vector of the corresponding Bank.
3. **Output**:
   * Confirmation message of the employee's addition.
   * Optionally display the employee's details.

**2. Adding a New Head Office Employee**

1. **Input**:
   * Head office employee details such as name, date of birth, post, salary, and date of joining.
2. **Process**:
   * Create a HeadEmployee object.
   * Add the object to the allHeadEmployees vector of the HeadOffice.
3. **Output**:
   * Confirmation of the head office employee's addition.

**3. Opening a New Savings Account**

1. **Input**:
   * Account holder details such as name, initial deposit, account number, and password.
2. **Process**:
   * Create a SavingsAccount object with attributes such as minBal, rateOfInterest, and overdraft capability.
   * Add the account to the allSavingsAccounts vector in the corresponding Bank.
3. **Output**:
   * Account creation success message.
   * Display of account details (e.g., balance, account number).

**4. Opening a New Current Account**

1. **Input**:
   * Similar to the savings account process but specific for business-related attributes.
2. **Process**:
   * Create a CurrentAccount object.
   * Add the object to the allCurrentAccounts vector in the Bank.
3. **Output**:
   * Account creation confirmation.

**5. Displaying All Employees in a Branch**

1. **Input**:
   * Call the displayAllBranchEmployees() method for a specific Bank object.
2. **Process**:
   * Iterate over the allBranchEmployees vector.
   * Display the details of each BranchEmployee object.
3. **Output**:
   * List of all branch employees, including their name, post, salary, and date of joining.

**6. Managing Multiple Banks in the Head Office**

1. **Adding a Bank**:
   * Input: Branch name and IFSC code.
   * Process: Create a Bank object and add it to the allBanks vector in HeadOffice.
   * Output: Confirmation of bank addition.
2. **Retrieving a Bank by IFSC**:
   * Input: IFSC code.
   * Process: Search the allBanks vector for a matching IFSC.
   * Output: Return the matching Bank object.
3. **Displaying All Banks**:
   * Input: Call displayAllBanks() from HeadOffice.
   * Process: Iterate over the allBanks vector and display each bank's details.
   * Output: List of all banks and their associated IFSC codes.

**7. Performing Account Operations**

1. **Depositing Money**:
   * Input: Account number, deposit amount.
   * Process: Locate the account, add the amount to balance, and update the total.
   * Output: Updated balance confirmation.
2. **Withdrawing Money**:
   * Input: Account number, withdrawal amount.
   * Process:
     + Check account type (SavingsAccount or CurrentAccount).
     + Verify sufficient balance and overdraft limits.
     + Deduct the amount from balance.
   * Output: Withdrawal confirmation or error message for insufficient balance.

**8. Displaying All Accounts in a Branch**

1. **Input**:
   * Call displayAllAccounts() from a Bank object.
2. **Process**:
   * Combine allSavingsAccounts and allCurrentAccounts vectors.
   * Iterate and display details for each account.
3. **Output**:
   * List of all accounts in the branch, including holder names and balances.