🏦 Case Study Title: Banking System Application Using OOPs Concepts

🎯 Objective

Design a **modular banking system** that supports:

- Customer management
- Account operations (deposit, withdraw, transfer)
- Transaction history
- Branch-level customer segregation

This case study uses the following **OOPs principles**:

Concept	Applied Through
Abstraction	Abstract class `Account`
Inheritance	`SavingsAccount` and `CurrentAccount` from `Account`
Polymorphism	Method overriding (run-time), overloading (compile-time)
Encapsulation	All data members are private with public getters/setters
Interface	Interface `BankOperations` implemented by accounts
Aggregation	`BankBranch` HAS-A list of `Customer` objects

🧰 Key Classes & Design

1. 📄 `BankOperations` (Interface)

Defines the essential banking operations.

🔶 **Methods:**

```
void deposit(double amount);
void withdraw(double amount);
void transfer(Account target, double amount);
double checkBalance();
void showTransactionHistory();
```

2. 📄 `Account` (Abstract Class)

Provides a common structure for all account types.

🔶 **Data Members:**

```
protected String accountNumber;
protected double balance;
protected List<String> transactionHistory;
```

🔶 **Methods:**

- `abstract void deposit(double amount);`
- `abstract void withdraw(double amount);`
- `void transfer(Account target, double amount)` – Common logic
- `double checkBalance()` – Returns balance
- `void addTransaction(String info)` – Utility to log transactions
- `void showTransactionHistory()` – Displays all transactions

3. 🏦 `SavingsAccount` (extends `Account`, implements `BankOperations`)

Represents a savings account with withdrawal limits.

🔶 **Additional Data Members:**

```
private final double MIN_BALANCE = 1000.0;
```

🔶 **Methods:**

- `void deposit(double amount)`
- `void withdraw(double amount)` – Checks for MIN_BALANCE

4. 🏦 `CurrentAccount` (extends `Account`, implements `BankOperations`)

Represents a current account with overdraft support.

🔶 **Additional Data Members:**

```
private final double OVERDRAFT_LIMIT = 2000.0;
```

🔶 **Methods:**

- `void deposit(double amount)`
- `void withdraw(double amount)` – Allows overdraft up to limit

5. 👱 `Customer`

Represents a customer of the bank.

🔶 **Data Members:**

```
private String customerId;
private String name;
private List<Account> accounts;
```

🔶 **Methods:**

- `void addAccount(Account acc)`
- `List<Account> getAccounts()`
- `String getCustomerId()`
- `String getName()`

6. 🏢 `BankBranch`

Represents a physical bank branch.

🔶 **Data Members:**

```
private String branchId;
private String branchName;
private List<Customer> customers;
```

🔶 **Methods:**

- `void addCustomer(Customer c)`
- `Customer findCustomerById(String id)`
- `void listAllCustomers()`

👓 Sample Use Case

Step-by-step:

1. Bank Manager creates a branch: `BankBranch branch = new BankBranch("B001", "Main Branch");`
2. A customer is created and added: `Customer c1 = new Customer("C001", "Alice");`
3. Customer opens a Savings Account and a Current Account.
4. Deposits money, withdraws from the current account with overdraft.
5. Transfers funds from savings to current account.
6. All operations logged in transaction history.

✅ Sample Output:

✅ Branch Created: Main Branch [Branch ID: B001]
✅ Customer Created: Alice [Customer ID: C001]
✅ Customer added to branch.

✅ Savings Account [S001] opened with initial balance: ₹5000.0
✅ Current Account [C001] opened with initial balance: ₹2000.0 and overdraft limit ₹1000.0

➡️ Deposited ₹2000.0 to Savings Account [S001]
💰 Current Balance: ₹7000.0

➡️ Withdrawn ₹2500.0 from Current Account [C001]
💰 Current Balance: -₹500.0 (Using Overdraft)

🔄 Transferred ₹1000.0 from Savings Account [S001] to Current Account [C001]
💰 Savings Balance: ₹6000.0
💰 Current Balance: ₹500.0

📋 Transaction History:
Account: S001
- Deposited: ₹2000.0
- Transferred to Account C001: ₹1000.0

Account: C001
- Withdrawn: ₹2500.0
- Received from Account S001: ₹1000.0