# Case Study Title: Banking System Application Using OOPs Concepts

 **Objective**

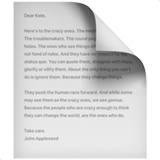
Design a **modular banking system** that supports:

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

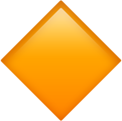
This case study uses the following **OOPs principles**:

|  |  |
| --- | --- |
| **Concept** | **Applied Through** |
| Abstraction | Abstract class Account |
| Inheritance | SavingsAccount and CurrentAccount from  Account |
| Polymorphis m | 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();

**package** oops;

**public** **interface** BankOperations {

**void** deposit(**double** amount);

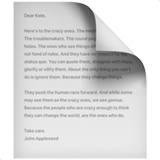
**void** withdraw(**double** amount);

**void** transfer(Account target, **double** amount);

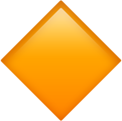
**double** checkBalance();

**void** showTransactionHistory();

}

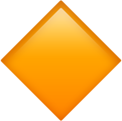
1. **Account (Abstract Class)**

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

**package** oops;

**public** **abstract** **class** Account {

**protected** String accountNumber;

**protected** **double** balance;

**private** String[] transactions = **new** String[10];

**private** **int** transactionCount = 0;

**public** Account(String accountNumber, **double** initialBalance) {

**this**.accountNumber = accountNumber;

**this**.balance = initialBalance;

}

**public** **abstract** **void** deposit(**double** amount);

**public** **abstract** **void** withdraw(**double** amount);

**public** **void** addTransaction(String info) {

**if** (transactionCount < transactions.length) {

transactions[transactionCount++] = info;

} **else** {

// shift left to make room for latest

**for** (**int** i = 1; i < transactions.length; i++) {

transactions[i - 1] = transactions[i];

}

transactions[transactions.length - 1] = info;

}

}

**public** **void** showTransactionHistory() {

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

**for** (**int** i = 0; i < transactionCount; i++) {

System.***out***.println(transactions[i]);

}

}

**public** **double** checkBalance() {

**return** balance;

}

**public** **void** transfer(Account target, **double** amount) {

**if** (**this**.balance >= amount) {

**this**.withdraw(amount);

target.deposit(amount);

} **else** {

System.***out***.println("Insufficient balance to transfer.");

}

}

**public** String getAccountNumber() {

**return** accountNumber;

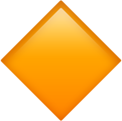
}

}

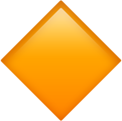
1. **** **SavingsAccount (extends Account, implements**

## 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

**package** oops;

**public** **class** SavingsAccount **extends** Account {

**public** SavingsAccount(String accNo, **double** initialBalance) {

**super**(accNo, initialBalance);

}

@Override

**public** **void** deposit(**double** amount) {

balance += amount;

}

@Override

**public** **void** withdraw(**double** amount) {

**if** (balance >= amount) {

balance -= amount;

} **else** {

System.***out***.println("Insufficient balance in Savings Account.");

}

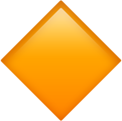
}

}

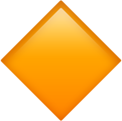
1. **** **CurrentAccount (extends Account, implements**

## 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

**package** oops;

**public** **class** CurrentAccount **extends** Account {

**private** **double** overdraftLimit = 1000;

**public** CurrentAccount(String accNo, **double** initialBalance) {

**super**(accNo, initialBalance);

}

@Override

**public** **void** deposit(**double** amount) {

balance += amount;

}

@Override

**public** **void** withdraw(**double** amount) {

**if** (balance + overdraftLimit >= amount) {

balance -= amount;

} **else** {

System.***out***.println("Overdraft limit exceeded in Current Account.");

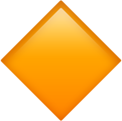
}

}

}

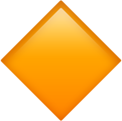
## Customer

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()

**package** oops;

**public** **class** Customer {

**private** String customerId;

**private** String name;

**private** Account account1; // supports max 2 accounts without List

**private** Account account2;

**public** Customer(String id, String name) {

**this**.customerId = id;

**this**.name = name;

}

**public** **void** addAccount(Account account) {

**if** (account1 == **null**) {

account1 = account;

} **else** **if** (account2 == **null**) {

account2 = account;

} **else** {

System.***out***.println("Customer can have only two accounts.");

}

}

**public** String getName() {

**return** name;

}

**public** String getCustomerId() {

**return** customerId;

}

**public** Account getAccount1() {

**return** account1;

}

**public** Account getAccount2() {

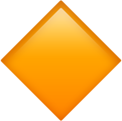
**return** account2;

}

}

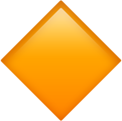
## BankBranch

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()

**package** oops;

**public** **class** BankBranch {

**private** Customer customer1; // max 2 customers for simplicity

**private** Customer customer2;

**public** BankBranch(String id, String name) {

// **TODO** Auto-generated constructor stub

}

**public** **void** addCustomer(Customer customer) {

**if** (customer1 == **null**) {

customer1 = customer;

} **else** **if** (customer2 == **null**) {

customer2 = customer;

} **else** {

System.***out***.println("Branch full. Can't add more customers.");

}

}

}

# Sample Use Case

**Step-by-step:**

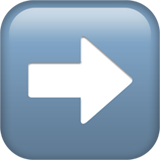
1. Bank Manager creates a branch: BankBranch branch = new BankBranch("B001", "Main Branch");
2. A customer is created and added: Customer c1 = new Customer("C001", "Alice");
3. Customer opens a Savings Account and a Current Account.
4. Deposits money, withdraws from the current account with overdraft.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

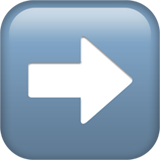
# Sample Output:

 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

**MY OUTPUT:**

Enter amount to deposit in Savings Account: 15000

Depositing Rs.:15000.0to Savings Account

Savings Balance: 20000.0

Enter amount to withdraw from Current Account: 50000

Withdrawing Rs.50000.0from Current Account

Overdraft limit exceeded in Current Account.

Current Balance: 2000.0

Enter amount to transfer from Savings to Current: 20000

Transferring Rs.20000.0from Savings to Current

Savings Balance: 0.0

Current Balance: 22000.0