Day3 ASSIGNMENT

```
import java.util.*;

interface BankOperations {

    void deposit(double amount);

    void withdraw(double amount);

    void transfer(Account target, double amount);

    double checkBalance();

    void showTransactionHistory();

}

abstract class Account implements BankOperations {

    protected String accountNumber;

    protected double balance;

    protected List<String> transactionHistory = new ArrayList<>();

    public Account(String accountNumber, double initialBalance) {

        this.accountNumber = accountNumber;

        this.balance = initialBalance;

        addTransaction("Account created with balance ₹" + initialBalance);

    }


    public abstract void deposit(double amount);

    public abstract void withdraw(double amount);

    public void transfer(Account target, double amount) {

        if (this.balance >= amount) {

            this.balance -= amount;

            target.deposit(amount);

            addTransaction("Transferred ₹" + amount + " to Account " + target.accountNumber);
        } else {
            System.out.println("⍰ Insufficient balance for transfer.");
        }
    }

    public double checkBalance() {
        return balance;
    }

    protected void addTransaction(String info) {
        transactionHistory.add(info);
    }
    public void showTransactionHistory() {
        System.out.println("   Transaction History for Account: " + accountNumber);
        for (String entry : transactionHistory) {
            System.out.println(" - " + entry);
        }
    }
}

// SavingsAccount class
class SavingsAccount extends Account {
    private final double MIN_BALANCE = 1000.0;

    public SavingsAccount(String accountNumber, double initialBalance) {
        super(accountNumber, initialBalance);
    }

    public void deposit(double amount) {
        balance += amount;
        addTransaction("Deposited: ₹" + amount);
    }

    public void withdraw(double amount) {
        if (balance - amount >= MIN_BALANCE) {
            balance -= amount;
            addTransaction("Withdrawn: ₹" + amount);
        } else {
            System.out.println("⍰ Minimum balance requirement not met.");
        }
    }
}

// CurrentAccount class
class CurrentAccount extends Account {
    private final double OVERDRAFT_LIMIT = 2000.0;

    public CurrentAccount(String accountNumber, double initialBalance) {
        super(accountNumber, initialBalance);
    }
    public void deposit(double amount) {
        balance += amount;
        addTransaction("Deposited: ₹" + amount);
    }

    public void withdraw(double amount) {
        if (balance - amount >= -OVERDRAFT_LIMIT) {
            balance -= amount;
            addTransaction("Withdrawn: ₹" + amount);
        } else {
            System.out.println("� Overdraft limit exceeded.");
        }
    }
}

// Customer class
class Customer {
    private String customerId;
    private String name;
    private List<Account> accounts = new ArrayList<>();

    public Customer(String customerId, String name) {
        this.customerId = customerId;
        this.name = name;
    }

    public void addAccount(Account acc) {
        accounts.add(acc);
    }

    public List<Account> getAccounts() {
        return accounts;
    }

    public String getCustomerId() {
        return customerId;
    }

    public String getName() {
        return name;
    }
}

// BankBranch class
class BankBranch {
    private String branchId;
    private String branchName;
    private List<Customer> customers = new ArrayList<>();
```

Day3 ASSIGNMENT

```
    public BankBranch(String branchId, String branchName) {
        this.branchId = branchId;
        this.branchName = branchName;
        System.out.println("⍰ Branch Created: " + branchName + " [Branch ID: " + branchId + "]");
    }

    public void addCustomer(Customer c) {
        customers.add(c);
        System.out.println("⍰ Customer added: " + c.getName() + " [Customer ID: " +
c.getCustomerId() + "]");
    }

    public Customer findCustomerById(String id) {
        for (Customer c : customers) {
            if (c.getCustomerId().equals(id)) {
                return c;
            }
        }
        return null;
    }

    public void listAllCustomers() {
        for (Customer c : customers) {
            System.out.println("    " + c.getName() + " [ID: " + c.getCustomerId() + "]");
        }
    }
}

// Main class with Scanner
public class Day3BankingApp {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BankBranch branch = new BankBranch("B001", "Main Branch");

        System.out.print("Enter Customer ID: ");
        String customerId = sc.nextLine();
        System.out.print("Enter Customer Name: ");
        String name = sc.nextLine();
        Customer c = new Customer(customerId, name);
        branch.addCustomer(c);

        // Create Accounts
        SavingsAccount sa = new SavingsAccount("S001", 5000.0);
        CurrentAccount ca = new CurrentAccount("C001", 2000.0);
        c.addAccount(sa);
        c.addAccount(ca);

        boolean running = true;
        while (running) {
            System.out.println("\n   Choose Operation:");
```

Day3 ASSIGNMENT

```
System.out.println("1. Deposit to Savings");
System.out.println("2. Withdraw from Current");
System.out.println("3. Transfer Savings ➡ Current");
System.out.println("4. Show Balances");
System.out.println("5. Show Transactions");
System.out.println("6. Exit");

int choice = sc.nextInt();

switch (choice) {
  case 1:
    System.out.print("Enter deposit amount: ");
    double depAmt = sc.nextDouble();
    sa.deposit(depAmt);
    break;
  case 2:
    System.out.print("Enter withdrawal amount: ");
    double wdAmt = sc.nextDouble();
    ca.withdraw(wdAmt);
    break;
  case 3:
    System.out.print("Enter amount to transfer: ");
    double trAmt = sc.nextDouble();
    sa.transfer(ca, trAmt);
    break;
```

```
            case 4:
                System.out.println("Savings Balance: ₹" + sa.checkBalance());
                System.out.println("Current Balance: ₹" + ca.checkBalance());
                break;
            case 5:
                sa.showTransactionHistory();
                ca.showTransactionHistory();
                break;
            case 6:
                running = false;
                break;
            default:
                System.out.println("⍰ Invalid option.");
        }
    }

    sc.close();
  }
}
```