Assignment Day-3

```
// BankOperations.java

public interface BankOperations {
    void deposit(double amount);
    void withdraw(double amount);
    void transfer(Account target, double amount);
    double checkBalance();
    void showTransactionHistory();
}

//Account.java

import java.util.*;

public 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;
    }

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

    public void transfer(Account target, double amount) {
        if (this.balance >= amount) {
            this.withdraw(amount);
            target.deposit(amount);
            addTransaction("Transferred to Account " + target.accountNumber
+ ": " + amount);
            target.addTransaction("Received from Account " +
this.accountNumber + ": " + amount);
        } 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 transaction : transactionHistory) {
            System.out.println(" - " + transaction);
        }
```

```
    }

    public String getAccountNumber() {
        return accountNumber;
    }
}

// SavingsAccount.java

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

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

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

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            addTransaction("Withdrawn: " + amount);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

// CurrentAccount.java

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

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

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

    @Override
    public void withdraw(double amount) {
        if (balance - amount >= -OVERDRAFT_LIMIT) {
            balance -= amount;
            addTransaction("Withdrawn: " + amount);
        } else {
            System.out.println("Overdraft limit exceeded. Cannot withdraw "
+ amount);
        }
```

```
    }
}

// Customer.java

import java.util.*;

public 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.java


import java.util.*;

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

    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 to branch: " + c.getName());
    }

    public Customer findCustomerById(String id) {
```

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

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

// Main.java

public class Main {

    public static void main(String[] args) {

        BankBranch branch = new BankBranch("B001", "Main Branch");


        Customer c1 = new Customer("C001", "Alice");

        System.out.println("Customer Created: " + c1.getName() + "
[Customer ID: " + c1.getCustomerId() + "]");


        branch.addCustomer(c1);


        SavingsAccount sa = new SavingsAccount("S001", 5000.0);

        CurrentAccount ca = new CurrentAccount("C001", 2000.0);


        c1.addAccount(sa);

        c1.addAccount(ca);


        System.out.println("Savings Account [S001] opened with initial
balance: " + sa.checkBalance());

        System.out.println("Current Account [C001] opened with initial
balance: " + ca.checkBalance());


        sa.deposit(2000);

        ca.withdraw(2500);

        sa.transfer(ca, 1000);
```

```
        System.out.println("Final Balances:");
        System.out.println("Savings Balance: " + sa.checkBalance());
        System.out.println("Current Balance: " + ca.checkBalance());

        System.out.println("Transaction History:");
        sa.showTransactionHistory();
        ca.showTransactionHistory();
    }
}
```