**ADVANCED PROGRAMMING ASSIGNMENT 1**

**SHARON NYABOKE**

**SCT212-0193/2022**

**Question 1: Extending Interface in a Concrete Class**

1. **Define the TransactionInterface:**

import java.util.Calendar;

public interface TransactionInterface {

double getAmount();

Calendar getDate();

String getTransactionID();

void printTransactionDetails();

void apply(BankAccount ba);

}

1. **Create the BaseTransaction class:**
   * Implement the methods defined in TransactionInterface.
   * Define common transaction fields: amount, date, and transactionID.

import java.util.Calendar;

public class BaseTransaction implements TransactionInterface {

private double amount;

private Calendar date;

private String transactionID;

public BaseTransaction(double amount, Calendar date, String transactionID) {

this.amount = amount;

this.date = date;

this.transactionID = transactionID;

}

@Override

public double getAmount() {

return amount;

}

@Override

public Calendar getDate() {

return date;

}

@Override

public String getTransactionID() {

return transactionID;

}

@Override

public void printTransactionDetails() {

System.out.println("Transaction ID: " + transactionID);

System.out.println("Amount: " + amount);

System.out.println("Date: " + date.getTime());

}

@Override

public void apply(BankAccount ba) {

// Default behavior for BaseTransaction

System.out.println("Applying transaction on account...");

}

}

1. **Create DepositTransaction and WithdrawalTransaction classes:**
   * Extend BaseTransaction and override the apply() method.

public class DepositTransaction extends BaseTransaction {

public DepositTransaction(double amount, Calendar date, String transactionID) {

super(amount, date, transactionID);

}

@Override

public void apply(BankAccount ba) {

ba.deposit(getAmount());

System.out.println("Deposit was successful.");

}

}

public class WithdrawalTransaction extends BaseTransaction {

public WithdrawalTransaction(double amount, Calendar date, String transactionID) {

super(amount, date, transactionID);

}

@Override

public void apply(BankAccount ba) {

if (ba.getBalance() >= getAmount()) {

ba.withdraw(getAmount());

System.out.println("Withdrawal was successful.");

} else {

System.out.println("Insufficient funds for withdrawal.");

}

}

}

**Question 2: Differentiating Deposit and Withdrawal**

1. **Implement reverse() in WithdrawalTransaction:**
   * Restore the balance to its original state.

public class WithdrawalTransaction extends BaseTransaction {

private boolean reversed = false;

public WithdrawalTransaction(double amount, Calendar date, String transactionID) {

super(amount, date, transactionID);

}

@Override

public void apply(BankAccount ba) {

if (ba.getBalance() >= getAmount()) {

ba.withdraw(getAmount());

} else {

System.out.println("Insufficient funds for withdrawal.");

}

}

public boolean reverse(BankAccount ba) {

if (!reversed) {

ba.deposit(getAmount());

reversed = true;

System.out.println("Withdrawal reversed successfully.");

return true;

}

return false;

}

}

**Question 3: Exception Handling**

1. **Create InsufficientFundsException:**

public class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

1. **Modify the apply() method in WithdrawalTransaction:**
   * Use the **throws** keyword and handle the exception.

@Override

public void apply(BankAccount ba) throws InsufficientFundsException {

if (ba.getBalance() < getAmount()) {

throw new InsufficientFundsException("Insufficient funds for this transaction.");

} else {

ba.withdraw(getAmount());

System.out.println("Withdrawal successful.");

}

}

1. **Overload apply() to handle partial withdrawal:**

public void apply(BankAccount ba, boolean allowPartial) {

try {

if (ba.getBalance() >= getAmount()) {

ba.withdraw(getAmount());

System.out.println("Withdrawal successful.");

} else if (allowPartial && ba.getBalance() > 0) {

double remainingBalance = ba.getBalance();

ba.withdraw(remainingBalance);

System.out.println("Partially withdrew: " + remainingBalance);

} else {

throw new InsufficientFundsException("Insufficient funds.");

}

} catch (InsufficientFundsException e) {

System.out.println(e.getMessage());

} finally {

System.out.println("Transaction attempt complete.");

}

}

**Question 4: Writing Client Code**

1. **Create the Main class:**

import java.util.Calendar;

public class Main {

public static void main(String[] args) {

BankAccount account = new BankAccount(500.0);

Calendar date = Calendar.getInstance();

DepositTransaction deposit = new DepositTransaction(200.0, date, "TXN001");

WithdrawalTransaction withdrawal = new WithdrawalTransaction(100.0, date, "TXN002");

deposit.apply(account);

deposit.printTransactionDetails();

try {

withdrawal.apply(account);

} catch (InsufficientFundsException e) {

System.out.println(e.getMessage());

}

withdrawal.printTransactionDetails();

// Test reverse functionality

withdrawal.reverse(account);

System.out.println("Balance after reverse: " + account.getBalance());

}

}

1. **Define the BankAccount class:**

public class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public double getBalance() {

return balance;

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

balance -= amount;

}

}