**Assignment 1**

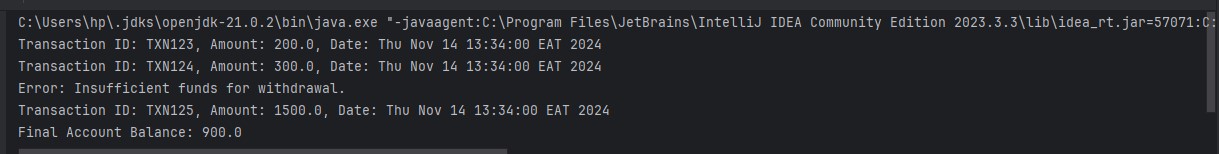
**SCT212-0153/2022**

**Njenga Beatrice Wambui**

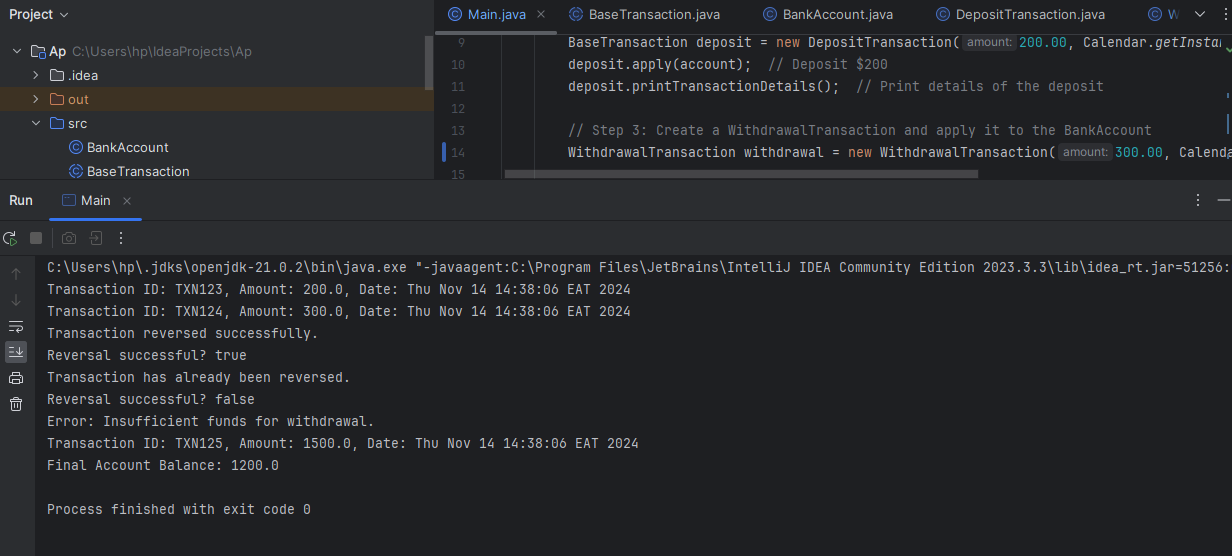
**Advanced Programming Assignment One (1)**

**Dr. Mulang’ Onando**

**Question 1 - Extending Interface in Concrete Class**



**Question 2 - Differentiate functionality of DepositTransaction and WithdrawalTransaction**



**Main.java**

**Update the Main class to test the reverse() method of WithdrawalTransaction**

import java.util.Calendar;

public class Main {

public static void main(String[] args) throws InsufficientFundsException {

// Step 1: Create a BankAccount object

BankAccount account = new BankAccount("123456", 1000.00);

// Step 2: Create a DepositTransaction and apply it to the BankAccount

BaseTransaction deposit = new DepositTransaction(200.00, Calendar.getInstance(), "TXN123");

deposit.apply(account); // Deposit $200

deposit.printTransactionDetails(); // Print details of the deposit

// Step 3: Create a WithdrawalTransaction and apply it to the BankAccount

WithdrawalTransaction withdrawal = new WithdrawalTransaction(300.00, Calendar.getInstance(), "TXN124");

try {

withdrawal.apply(account); // Attempt to withdraw $300

} catch (InsufficientFundsException e) {

System.out.println("Error: " + e.getMessage());

}

withdrawal.printTransactionDetails(); // Print details of the withdrawal

// Step 4: Attempt to reverse the withdrawal transaction

boolean reversed = withdrawal.reverse(account); // Attempt to reverse the withdrawal

System.out.println("Reversal successful? " + reversed);

// Step 5: Attempt to reverse the withdrawal again to test that reversal can only be done once

reversed = withdrawal.reverse(account);

System.out.println("Reversal successful? " + reversed);

// Step 6: Attempt a large withdrawal to trigger InsufficientFundsException

WithdrawalTransaction largeWithdrawal = new WithdrawalTransaction(1500.00, Calendar.getInstance(), "TXN125");

try {

largeWithdrawal.apply(account); // This should trigger InsufficientFundsException

} catch (InsufficientFundsException e) {

System.out.println("Error: " + e.getMessage());

}

largeWithdrawal.printTransactionDetails();

// Step 7: Print the Final Account Balance

System.out.println("Final Account Balance: " + account.getBalance());

}

}

**WithdrawalTransaction.java**

**updated WithdrawalTransaction class with the new reverse() method:**

import java.util.Calendar;

public class WithdrawalTransaction extends BaseTransaction {

private boolean reversed; // Tracks if the transaction has already been reversed

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

super(amount, date, transactionID);

this.reversed = false; // Initialize reversed flag as false

}

@Override

public void apply(BankAccount ba) throws InsufficientFundsException {

if (ba.getBalance() < amount) {

throw new InsufficientFundsException("Insufficient funds for withdrawal.");

}

ba.withdraw(amount);

this.reversed = false; // Reset reversed flag each time apply is called

}

// Method to reverse the withdrawal

public boolean reverse(BankAccount ba) {

if (!reversed) {

ba.deposit(amount); // Restore the withdrawn amount to the account balance

reversed = true; // Mark this transaction as reversed

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

return true;

} else {

System.out.println("Transaction has already been reversed.");

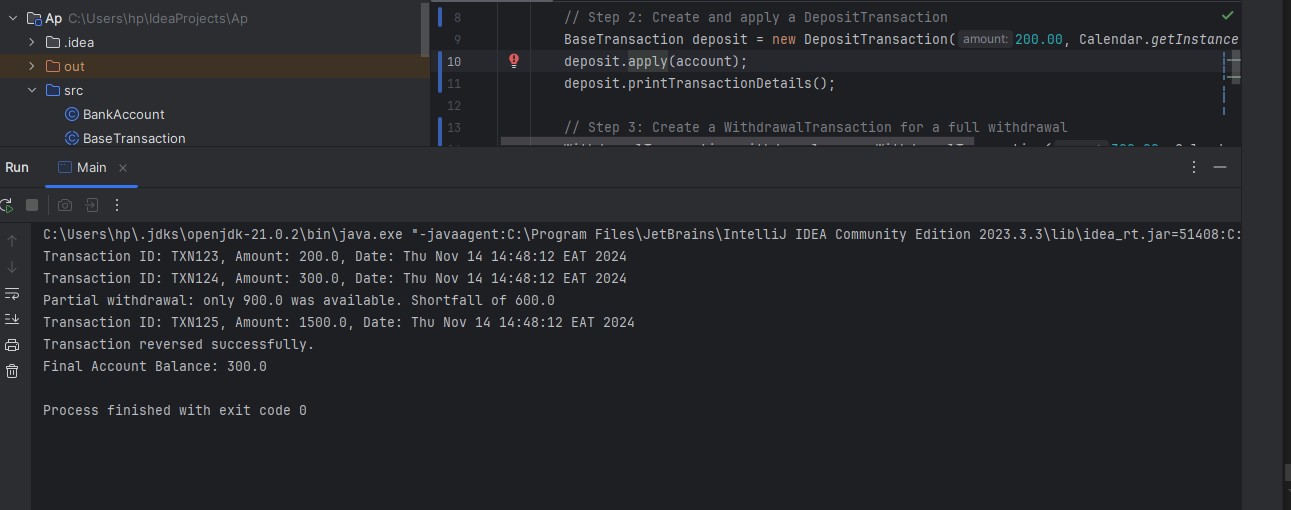
return false;

}

}

}

**Question 3 - Exception Handling and Client Codes**

****

**InsufficientFundsException**

**InsufficientFundsException to handle errors that may occur during transactions on the bank account**

public class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

**Update WithdrawalTransaction.java with an Overloaded apply() Method**

import java.util.Calendar;

public class WithdrawalTransaction extends BaseTransaction {

private boolean reversed; // Tracks if the transaction has already been reversed

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

super(amount, date, transactionID);

this.reversed = false;

}

@Override

public void apply(BankAccount ba) throws InsufficientFundsException {

if (ba.getBalance() < amount) {

throw new InsufficientFundsException("Insufficient funds for withdrawal.");

}

ba.withdraw(amount);

this.reversed = false; // Reset reversed flag each time apply is called

}

// Overloaded apply method for partial withdrawals

public void apply(BankAccount ba, boolean allowPartialWithdrawal) {

try {

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

ba.withdraw(amount); // Full withdrawal

System.out.println("Full withdrawal successful.");

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

double availableBalance = ba.getBalance();

ba.withdraw(availableBalance); // Withdraw all available funds

double shortfall = amount - availableBalance;

System.out.println("Partial withdrawal: only " + availableBalance + " was available. Shortfall of " + shortfall);

} else {

throw new InsufficientFundsException("Insufficient funds for withdrawal.");

}

} catch (InsufficientFundsException e) {

System.out.println("Error: " + e.getMessage());

}

}

// Method to reverse the withdrawal

public boolean reverse(BankAccount ba) {

if (!reversed) {

ba.deposit(amount); // Restore the withdrawn amount to the account balance

reversed = true; // Mark this transaction as reversed

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

return true;

} else {

System.out.println("Transaction has already been reversed.");

return false;

}

}

}

**Update Main.java to Test Both Full and Partial Withdrawals**

import java.util.Calendar;

public class Main {

public static void main(String[] args) {

// Step 1: Create a BankAccount object

BankAccount account = new BankAccount("123456", 1000.00);

// Step 2: Create and apply a DepositTransaction

BaseTransaction deposit = new DepositTransaction(200.00, Calendar.getInstance(), "TXN123");

deposit.apply(account);

deposit.printTransactionDetails();

// Step 3: Create a WithdrawalTransaction for a full withdrawal

WithdrawalTransaction withdrawal = new WithdrawalTransaction(300.00, Calendar.getInstance(), "TXN124");

try {

withdrawal.apply(account); // Full withdrawal

} catch (InsufficientFundsException e) {

System.out.println("Error: " + e.getMessage());

}

withdrawal.printTransactionDetails();

// Step 4: Attempt a large withdrawal with partial allowed

WithdrawalTransaction largeWithdrawal = new WithdrawalTransaction(1500.00, Calendar.getInstance(), "TXN125");

largeWithdrawal.apply(account, true); // Allow partial withdrawal

largeWithdrawal.printTransactionDetails();

// Step 5: Attempt to reverse the first withdrawal

withdrawal.reverse(account);

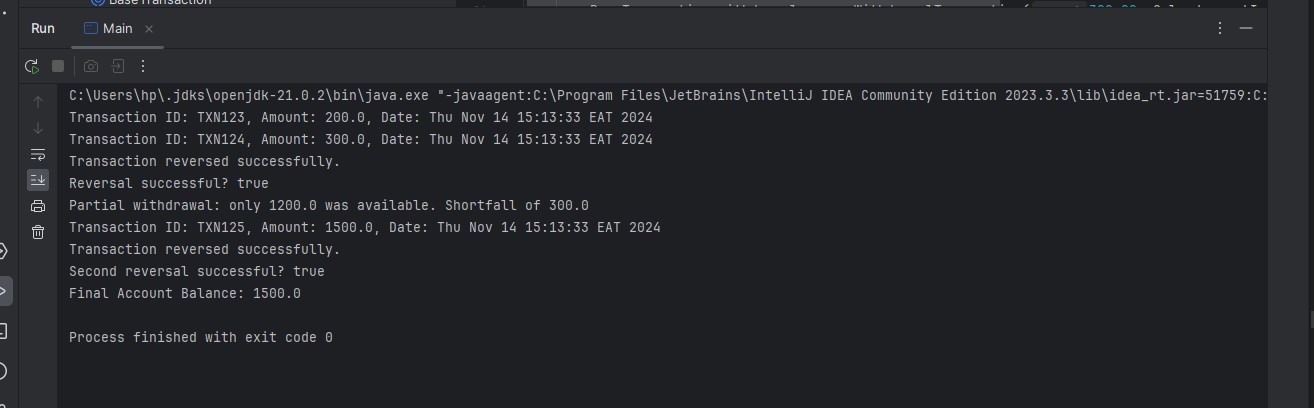
// Final Account Balance

System.out.println("Final Account Balance: " + account.getBalance());

}

}

**Question 4 - Writing the Client Code**

****

**Updated Main Class to Demonstrate All Requirements**

**Below is the modified Main class, which fulfills the requirements of Question 4 by testing the functionality of the DepositTransaction**

**and WithdrawalTransaction . Ensure you test the behavior of objects of the subclasses and the superclass.**

import java.util.Calendar;

public class Main {

public static void main(String[] args) throws InsufficientFundsException {

// Step 1: Create a BankAccount object

BankAccount account = new BankAccount("123456", 1000.00);

// Step 2: Create and apply a DepositTransaction as a BaseTransaction (Polymorphism)

BaseTransaction deposit = new DepositTransaction(200.00, Calendar.getInstance(), "TXN123");

deposit.apply(account);

deposit.printTransactionDetails();

// Step 3: Create a WithdrawalTransaction and treat it as a BaseTransaction

BaseTransaction withdrawal = new WithdrawalTransaction(300.00, Calendar.getInstance(), "TXN124");

// Using type casting to cast back to WithdrawalTransaction to access the reverse functionality

if (withdrawal instanceof WithdrawalTransaction) {

try {

withdrawal.apply(account); // Attempt full withdrawal

} catch (InsufficientFundsException e) {

System.out.println("Error: " + e.getMessage());

}

withdrawal.printTransactionDetails();

// Cast withdrawal back to WithdrawalTransaction to call reverse()

WithdrawalTransaction withdrawalTransaction = (WithdrawalTransaction) withdrawal;

boolean reversed = withdrawalTransaction.reverse(account); // Attempt to reverse the withdrawal

System.out.println("Reversal successful? " + reversed);

}

// Step 4: Test Partial Withdrawal with allowPartialWithdrawal flag

WithdrawalTransaction largeWithdrawal = new WithdrawalTransaction(1500.00, Calendar.getInstance(), "TXN125");

largeWithdrawal.apply(account, true); // Allow partial withdrawal if full amount isn't available

largeWithdrawal.printTransactionDetails();

// Step 5: Attempt to reverse the same transaction again to verify prevention of multiple reversals

boolean secondReversal = largeWithdrawal.reverse(account);

System.out.println("Second reversal successful? " + secondReversal);

// Step 6: Display the Final Account Balance

System.out.println("Final Account Balance: " + account.getBalance());

}

}