**NAME:** GIKURU JOSEPH NDERITU

**REG NO:** SCT212-0574/2022

**COURSE:** BSc COMPUTER TECHNOLOGY

Y**EAR**: 3.1

**UNIT:** ADVANCED PROGRAMMING.

**ASSIGNMENT:** ASSIGNMENT 1 UNDER INTERFACES

**LINK TO THE ASSIGNMENT** =[ASSIGNMENT1](https://github.com/mulangonando/Advanced-Programming/blob/main/src/Lecture4_interfaces_abstract_classes/Assignment_I_Advanced_Programming.pdf)

**LECTURER:** Dr. Mulang’ Onando

20th November 2024

**Transaction Interface.java**

public interface TransactionInterface {

double getAmount();

Calendar getDate();

String getTransactionID();

void printTransactionDetails();

void apply(BankAccount ba);

}

**BaseTransaction.java**

public class BaseTransaction implements TransactionInterface {

protected double amount;

protected Calendar date;

protected String transactionID;

public BaseTransaction(double amount) {

this.amount = amount;

this.date = Calendar.getInstance();

this.transactionID = generateTransactionID();

}

@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) {

System.out.println("Processing base transaction");

}

private String generateTransactionID() {

return "TXN" + System.currentTimeMillis();

}

}

**DepositTransaction.java**

public class DepositTransaction extends BaseTransaction {

public DepositTransaction(double amount) {

super(amount);

}

@Override

public void apply(BankAccount ba) {

ba.setBalance(ba.getBalance() + amount);

System.out.println("Deposit of " + amount + " processed successfully");

}

}

**Withdrawal transaction.java**

public class WithdrawalTransaction extends BaseTransaction {

private double unprocessedAmount = 0;

public WithdrawalTransaction(double amount) {

super(amount);

}

@Override

public void apply(BankAccount ba) throws InsufficientFundsException {

if (ba.getBalance() < amount) {

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

}

ba.setBalance(ba.getBalance() - amount);

System.out.println("Withdrawal of " + amount + " processed successfully");

}

public void apply(BankAccount ba, boolean withdrawAvailable) {

try {

double balance = ba.getBalance();

if (balance > 0 && balance < amount) {

unprocessedAmount = amount - balance;

ba.setBalance(0);

System.out.println("Partial withdrawal of " + balance + " processed");

System.out.println("Unprocessed amount: " + unprocessedAmount);

} else {

apply(ba);

}

} catch (InsufficientFundsException e) {

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

} finally {

System.out.println("Transaction processing completed");

}

}

public boolean reverse() {

// Implementation for reversing withdrawal

return true;

}

}

**InsufficientFundsException.java**

public class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

**BankAccount.java**

public class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public double getBalance() {

return balance;

}

public void setBalance(double balance) {

this.balance = balance;

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

BankAccount account = new BankAccount(1000);

// Test deposit

TransactionInterface deposit = new DepositTransaction(500);

deposit.apply(account);

deposit.printTransactionDetails();

// Test withdrawal

WithdrawalTransaction withdrawal = new WithdrawalTransaction(300);

try {

withdrawal.apply(account);

withdrawal.printTransactionDetails();

} catch (InsufficientFundsException e) {

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

}

// Test partial withdrawal

WithdrawalTransaction largeWithdrawal = new WithdrawalTransaction(2000);

largeWithdrawal.apply(account, true);

// Test type casting

BaseTransaction baseTransaction = (BaseTransaction) withdrawal;

baseTransaction.printTransactionDetails();

}

}

**SUMMARY OF WHAT MY CODE DOES**

This banking system program manages deposits and withdrawals through several interconnected classes. The `TransactionInterface` sets the basic rules, while `BaseTransaction` provides common transaction features like ID generation and date tracking. `DepositTransaction` handles money additions to accounts, and `WithdrawalTransaction` manages money removals with overdraft protection and reversal options. The `BankAccount` class stores the balance and handles balance updates, while `InsufficientFundsException` catches and handles withdrawal errors when there isn't enough money. All these classes work together in the `Main` class, which demonstrates the system's functionality through various transaction scenarios, ensuring secure and accurate money management.