Peter irungu Kamau

SCT212-0044/2018

[08:03, 09/12/2024]Peter: Main.java

import Lecture1\_adt.\*; // Import all classes from Lecture1\_adt package to be used in this client code

import java.util.Calendar;

import java.util.GregorianCalendar;

import java.util.ArrayList;

import java.util.List;

/\*

\* Client Code for accessing the Lecture1\_adt.TransactionInterface.java module

\*/

public class Main {

public static void testTransaction1() {

Calendar d1 = new GregorianCalendar(); // d1 is an Object [Objects are Reference types]

Lecture1\_adt.Transaction1 t1 = new Lecture1\_adt.Transaction1(1000, d1); // amount and d1 are arguments

System.out.println(t1.toString());

System.out.println("Lecture1\_adt.TransactionInterface Amount: \t " + t1.amount);

System.out.println("Lecture1\_adt.TransactionInterface Date: \t " + t1.date);

}

public static Transaction2 makeNextPayment(Transaction2 t) {

Calendar d = t.getDate();

d.add(Calendar.MONTH, 1);

return new Transaction2(t.getAmount(), d);

}

public static void testTransaction2() {

Calendar d1 = new GregorianCalendar();

Lecture1\_adt.Transaction2 t = new Lecture1\_adt.Transaction2(1000, d1);

Lecture1\_adt.Transaction2 modified\_t = makeNextPayment(t);

System.out.println("\n\nState of the Object T1 After Client Code Tried to Change the Amount");

System.out.println("Lecture1\_adt.TransactionInterface Amount: \t "+modified\_t.getAmount());

System.out.println("Lecture1\_adt.TransactionInterface Date: \t "+modified\_t.getDate().getTime());

System.out.println("\n\nHow does T2 Look Like?????");

System.out.println("Lecture1\_adt.TransactionInterface Amount: \t "+modified\_t.getAmount());

System.out.println("Lecture1\_adt.TransactionInterface Date: \t "+modified\_t.getDate().getTime());

}

public static List<Transaction3> makeYearOfPayments (int amount) throws NullPointerException {

List<Transaction3> listOfTransaction3s = new ArrayList<Transaction3>();

Calendar date = new GregorianCalendar(2024, Calendar.JANUARY, 3);

for (int i = 0; i < 12; i++) {

listOfTransaction3s.add(new Transaction3(amount, date));

date.add(Calendar.MONTH, 1);

}

return listOfTransaction3s;

}

public static void testTransaction3() {

List<Transaction3> allPaymentsIn2024 = makeYearOfPayments(1000);

for (Transaction3 t3 : allPaymentsIn2024) {

System.out.println("\n\n ::::::::::::::::::::::::::::::::::::::::::::\n");

System.out.println("Lecture1\_adt.TransactionInterface Amount: \t "+t3.getAmount());

System.out.println("Lecture1\_adt.TransactionInterface Date: \t "+t3.getDate().getTime());

}

}

public static List<Transaction4> makeYearOfPaymentsFinal (int amount) throws NullPointerException {

List<Transaction4> listOfTransaction4s = new ArrayList<Transaction4>();

Calendar date = new GregorianCalendar(2024, Calendar.JANUARY, 3);

for (int i = 0; i < 12; i++) {

listOfTransaction4s.add(new Transaction4(amount, date));

date.add(Calendar.MONTH, 1);

}

return listOfTransaction4s;

}

public static void testTransaction4() {

List<Transaction4> transactionsIn2024 = makeYearOfPaymentsFinal(1200);

for (Transaction4 transact : transactionsIn2024) {

System.out.println("\n\n ::::::::::::::::::::::::::::::::::::::::::::\n");

System.out.println("Lecture1\_adt.TransactionInterface Amount: \t "+transact.getAmount());

System.out.println("Lecture1\_adt.TransactionInterface Date: \t "+transact.getDate().getTime());

}

}

// Polymorphism test

public static void testPolymorphism(BankAccount account) {

Calendar depositDate = new Gre…

[08:04, 09/12/2024] Alvine: BankAccount. Java

package Lecture4\_interfaces\_abstract\_classes;

public class BankAccount {

private double balance;

// Constructor initializes the bank account with a starting balance

public BankAccount(double balance) {

if (balance < 0) {

throw new IllegalArgumentException("Balance cannot be negative.");

}

this.balance = balance;

}

// Getter for the balance

public double getBalance() {

return balance;

}

// Setter for the balance, includes validation to prevent negative balances

public void setBalance(double balance) {

if (balance < 0) {

throw new IllegalArgumentException("Balance cannot be negative.");

}

this.balance = balance;

}

// Method to deposit an amount into the account

public void deposit(double amount) {

if (amount <= 0) {

throw new IllegalArgumentException("Deposit amount must be greater than zero.");

}

this.balance += amount;

}

// Method to withdraw an amount from the account, ensuring sufficient funds

public void withdraw(double amount) {

if (amount <= 0) {

throw new IllegalArgumentException("Withdrawal amount must be greater than zero.");

}

if (amount > balance) {

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

}

this.balance -= amount;

}

// Method to print account details

public void printAccountDetails() {

System.out.println("Bank Account Balance: $" + balance);

}

}

**Second code**

BankAccount. Java

package Lecture4\_interfaces\_abstract\_classes;

public class BankAccount {

private double balance;

// Constructor initializes the bank account with a starting balance

public BankAccount(double balance) {

if (balance < 0) {

throw new IllegalArgumentException("Balance cannot be negative.");

}

this.balance = balance;

}

// Getter for the balance

public double getBalance() {

return balance;

}

// Setter for the balance, includes validation to prevent negative balances

public void setBalance(double balance) {

if (balance < 0) {

throw new IllegalArgumentException("Balance cannot be negative.");

}

this.balance = balance;

}

// Method to deposit an amount into the account

public void deposit(double amount) {

if (amount <= 0) {

throw new IllegalArgumentException("Deposit amount must be greater than zero.");

}

this.balance += amount;

}

// Method to withdraw an amount from the account, ensuring sufficient funds

public void withdraw(double amount) {

if (amount <= 0) {

throw new IllegalArgumentException("Withdrawal amount must be greater than zero.");

}

if (amount > balance) {

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

}

this.balance -= amount;

}

// Method to print account details

public void printAccountDetails() {

System.out.println("Bank Account Balance: $" + balance);

}

}

**Third code**

WithdrawalTransaction.java

package Lecture4\_interfaces\_abstract\_classes;

import org.jetbrains.annotations.NotNull;

import java.util.Calendar;

public class WithdrawalTransaction extends BaseTransaction {

public WithdrawalTransaction(int amount, @NotNull Calendar date) {

super(amount, date);

}

/\*\*

\* Checks if the transaction amount is valid (greater than 0).

\* @param amt the amount to check

\* @return true if the amount is valid, false if negative

\*/

private boolean checkTransactionAmount(int amt) {

return amt >= 0;

}

/\*\*

\* Reverses the withdrawal transaction (adds the amount back to the account).

\* @return true if the reversal was successful

\*/

public boolean reverse(BankAccount ba) {

ba.setBalance(ba.getBalance() + getAmount()); // Re-credit the amount

return true; // Assuming reversal is always successful

}

/\*\*

\* Prints the transaction details for the withdrawal.

\*/

public void printTransactionDetails() {

System.out.println("Withdrawal Transaction: " + this.toString());

}

/\*\*

\* Applies the withdrawal to the bank account by decreasing the balance.

\* Throws InsufficientFundsException if the balance is insufficient.

\* @param ba the BankAccount to apply the transaction to

\* @throws InsufficientFundsException if the account balance is insufficient

\*/

@Override

public void apply(BankAccount ba) throws InsufficientFundsException {

double curr\_balance = ba.getBalance();

if (curr\_balance < getAmount()) {

throw new InsufficientFundsException("Insufficient funds for withdrawal of " + getAmount());

}

ba.setBalance(curr\_balance - getAmount());

}

}

**Fourth code**

TransactionInterface.java

package Lecture4\_interfaces\_abstract\_classes;

import java.util.Calendar;

/\*\*

\* Interface for Transactions

\* Any class that defines a transaction must implement this Interface.

\* The interface enforces methods to retrieve transaction details such as amount, date, and a unique ID.

\*/

public interface TransactionInterface {

/\*\*

\* Gets the amount of the transaction.

\* @return the transaction amount as a double

\*/

double getAmount();

/\*\*

\* Gets the date of the transaction.

\* @return the date of the transaction as a Calendar object

\*/

Calendar getDate();

/\*\*

\* Gets the unique transaction ID.

\* @return the unique transaction ID as a String

\*/

String getTransactionID();

}