Advanced Programming Assignment One (1)

Dr. Mulang' Onando

October 2024, Semester: Sept - Dec. 2024

Introduction

This Assignment aims to provide practical experience for the student and cement knowledge of ADT Design - Specifically: how to implement ADTs through inheritance and polymorphism.

Problem Statement

You are tasked with creating a transaction system that processes different types of financial transactions (like Deposit and Withdrawal). To do this, you are provided with starter code from the lectures. You will implement several code snippets, and classes and override several methods. Further implementing client codes to demonstrate the functionality of the classes.

Objective

The objective of this question is to test students' understanding of the following key concepts in advanced object-oriented programming:

1. Defining and implementing an interface in a programming language.
2. Understanding the difference between Concrete Class and Abstract class inheriting from an Interface
3. Understanding the use of interfaces for abstraction.
4. Implementing methods basic transaction-related methods that implement polymorphism. Understand the concept of early and late binding as applied to polymorphic methods.
5. Create Java Exception Class; implement the exception class through both `throws` and `try...catch` block.

Question 1 - Extending Interface in Concrete Class

Aim: Creating a concrete class that implements an interface.

You will create a class called `BaseTransaction` that implements the `TransactionInterface`. The class should have the same fields implemented in the abstract `BaseTransaction` class. Be careful to implement all the abstract methods defined in the interface.

Implement the following methods:

- `getAmount(): double` - // Method to get the transaction amount
- `getDate(): Calendar` - // Method to get the transaction date
- `getTransactionID(): String` - // Method to get a unique identifier for the transaction

Implement common Transaction methods:

- `printTransactionDetails(): void` – A method to print out details of the transaction.
- `apply(BankAccount ba): void` – A method that applies this transaction on a Bank account object passed as a parameter. Ensure the implementation of this `apply()` method in the `BaseTransaction` class differs substantially from the implementations in both the `DepositTransaction` and the `WithdrawalTransaction` classes.

Implement method overriding in the Derived Classes:

- Modify the implementations of `DepositTransaction` and `WithdrawalTransaction` classes to override the definition of the `apply()` method inherited from the `BaseTransaction` class.
- Ensure to implement specifications for all methods defined in the `BaseTransaction` class

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

For this course, we shall differentiate the behavior of the two subclasses by making one design assumption: that all deposits are irreversible and withdrawals can be reversed.

Implement the `reverse(): boolean` method to reverse a `WithdrawalTransaction`. This method ensures that the balance in the `BankAccount` object that the transaction was initially applied to is restored to it's original amount.

Question 3 - Exception Handling and Client Codes

Write an exception handling class named `InsufficientFundsException` to handle errors that may occur during transactions on the bank account. Use knowledge of inheritance to ensure that this class is an actual Java Exception class.

- Use the `throws` keyword to handle the exceptions that occur when attempting a withdrawal in the `apply()` method in the `WithdrawalTransaction` class.
- Implement an overloaded `apply()` method in `WithdrawalTransaction` class that not only checks if the balance in the `BankAccount` covers the withdrawal amount but checks if the balance is greater than 0; in case the case when `0 < balance < withdrawal amount`, it withdraws all the balance in the `BankAccount` and keeps a record of the amount not withdrawn.
- Within the definition of the available balance withdrawal method defined above, implement exception handling using the `try{...} catch{...} final{...}` block

Question 4 - Writing the Client Code

Within the `Main` class; follow the examples of the functions to test specific classes already defined. Write code to test the functionality of the `DepositTransaction` and `WithdrawalTransaction`. Ensure you test the behavior of objects of the subclasses and the superclass.

Hint: create objects of the classes above and test the `apply()` method. use type casting to map the subtype objects to the base type object. Then test the `apply()` method of the base class object.

Grading Criteria

Correctness

- Does the implementation follow the OOP principles of abstraction and interface implementation?
- Does the `BaseTransaction` class correctly implement `Transaction interface`?
- Does the code handle exceptions correctly when insufficient funds exist?

Code Quality

- Is the code well-structured and documented?
- Are the methods logically named and implemented according to the problem statement?

Functionality

- Does the program perform credit and debit transactions as expected?
- Does the program properly print transaction details after each operation?

Exception Handling

Is the `InsufficientFundsException` properly thrown and handled in the program?