P04 Exceptional Bank Teller

Overview

In this assignment, we are going to implement a bank teller application to allow customers to access bank accounts through a unique identification. Using this teller application, a customer can complete any of the following transactions:

- withdraw money,
- deposit money,
- query account balances, and
- see the most recent 5 transactions.

The system must maintain the correct balances for all accounts. The system must be also able to process one or more transactions for any number of customers. It should also cope with erroneous input without crashing.

Learning Objectives

The goals of this assignment include:

- Gain more experience with creating classes and using objects.
- Learn how to improve the robustness of a program so it can survive unusual circumstances and cope with erroneous input without crashing.
- Develop your understanding of the difference between checked and unchecked exceptions, and get practice both throwing and catching exceptions of each kind.
- Get more practice writing tests, specifically tests that detect whether exceptions are thrown under the prescribed circumstances or not.

Grading Rubric

5 points	**Pre-Assignment Quiz:** You will not have access to this write-up without first completing this pre-assignment quiz through Canvas.
20 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
15 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

Additional Assignment Requirements

- All String comparisons in this assignment are CASE SENSITIVE.
- You MUST NOT add any additional fields either instance or static, and any public methods either static or instance to your AccountBank and BenkTeller classes, other than those defined in this write-up and these javadocs.
- You CAN add additional static test methods either public or private to your AccountBankTester and BankTellerTester classes, in addition to those specified in these javadocs.
- You CAN define local variables that you may need to implement the methods defined in this program.
- You CAN define private static methods to help implement the different public static methods defined in this program, if needed.

1 Getting Started

Start by creating a new Java Project in eclipse called P04 Exceptional Bank Teller, for instance. You have to ensure that your new project uses Java 8, by setting the "Use an execution

environment JRE:" drop down setting to "JavaSE-1.8" within the new Java Project dialog box. Following the next step, we are going to create 4 classes and add them to this project. We note also that Appendix A provides links to java API of the set of exception classes that you may use in this program. Appendix B presents a set of useful methods that you may use while developping this assignment.

2 Create the BankAccount and BankAccountTester classes

Create a new class called `BankAccount`. Each instance of `BankAccount` represents an account at a bank. Each BankAccount object should have its own account identifier (private field of type `String`), its own account balance (private field of type `int`), and its own list of transactions (private field of type `ArrayList<String>`). You are NOT allowed to add any additional instance or static field to the `BankAccount` class.

Now, implement the constructor and the public methods defined for the `BankAccount` class according to their detailed javadocs description provided within these javadocs. Read carefully the provided javadoc method headers, and pay close attention to the exceptions that should be thrown by the constructor and the public methods defined in the `BankAccount` class. For instance, the system must prevent invalid transactions such as cash withdrawal amounts that are negative or not a multiple of 10, or that are larger than the account's balance. Further restrictions related to the constructor and the other methods are described in the javadocs. Keep in mind to NOT catch an exception within the implementation details of a method if that exception has been declared to be thrown using `@throws` annotation in its javadoc method header. Let the exception propagate to the method's caller.

We would like to highlight that valid deposit and withdrawal operations must result into transactions added to the `BankAccount`'s transactions ArrayList. We represent a transaction using a String with the specific following format. A transaction must contain two portions that can be separated by one or more spaces as follows:

Format of a Transaction:
`transactionCode transactionAmount`

- `transactionCode` MUST be either "0" or "1". The code "0" refers to a withdrawal operation, while the code "1" refers to a deposit operation.
- `transactionAmount` represents the withdrawal or deposit amount.

For instance,
`0 30` → refers to a withdrawal transaction of an amount of 30.
`1 50` → refers to a deposit transaction of an amount of 50.

In order to check the correctness of the implementation of your `BankAccount` class, make sure to create a class called `BankAccountTester` and add it to your project. Your `BankAccountTester` class must implement at least the methods defined in these javadocs with exactly the same

signatures. You are encouraged to implement additional test methods to make sure that the constructor and public methods defined in your `BankAccount` class are implemented conforming to their specification.

We would like to highlight that none of your unit test methods should throw any exception. Your test method should catch the exceptions that may be thrown by the call of the appropriate `BankAccount` class's methods, and returns true if the expected behavior has been satisfied, and false otherwise.

3 Create the BankTeller and BankTellerTester classes

Now, create a new class called `BankTeller` and add it to your P04 project source folder. The BankTeller models the data type that will allow customers to access their bank accounts. It allows also creating new bank accounts. The `BankTeller` class MUST define ONLY one private instance field of type ArrayList that stores elements of type BankAccount. This list stores all the bank accounts created so far. You are not allowed to add any instance or static field or any public method to the `BankTeller` class not defined in these javadocs. Make sure to implement the constructor and the methods defined in this class with respect to the specification provided in these javadocs.

Create also a new class called `BankTellerTester` and add it to your project. This class should include all the test methods defined in these javadocswith exactly the same signatures. You can add further public or private test methods to your `BankTellerTester` class to check for the correctness of all the behaviors defined in the `BankTeller` class.

Important Notes

- Feel free to reuse the javadoc method headers provided in these javadocsin your class and method javadoc headers.
- The order of preceding of throwing multiple exceptions is not important in this assignment. A method may throw more than one exception such as BankTeller.addTransaction() method, which may throw NullPointerException and DataFormatException. When multiple exceptional conditions are met, it is OK to throw in first order either exception.
- BankTeller.loadTransactions() takes a reference to an object of type java.io.File as an input parameter. In your test methods which may call this BankTeller.loadTransactions() method, create the file object using the constructor of the java.io.File class that takes a path name String as input parameter.

4 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the course style guide, you should submit your final work through gradescope.com. The only 4 files that you must submit include: BankAccount.java, BankAccountTester.java, BankTeller.java, and BankTellerTester.java. Your score for this assignment will be based on your "active" submission made prior to the hard deadline of Due: **9:59PM on October 2**nd. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline. Finally, the third portion of your grade for your submission will be determined by humans looking for organization, clarity, commenting, and adherence to the course style guide.

Appendices

A Appendix A

java.util.Scanner
java.io.File
IllegalStateException
NoSuchElementException
DataFormatException
IOException
java.util.ArrayList
IllegalArgumentException
NullPointerException
NumberFormatException
FileNotFoundException

B Appendix B

The following list of methods may be useful while completing this assignment.

- String.split()
- String.equals()
- Integer.valueOf()

Extra Challenges

Here are some suggestions for interesting ways to extend this memory game, after you have completed, backed up, and submitted the graded portion of this assignment. **No extra credit will be awarded for implementing these features**, but they should provide you with some valuable practice and experience. DO NOT submit such extensions via gradescope.

1. Try to implement a driver application which provides a menu of options (set of available operations) and services both administration staff and customers. An administration staff can create new bank accounts. The bank teller machine must service customers who may wish to perform a balance query, a cash withdrawal, or a deposit. The bank teller driver application MUST not crash for any input erroneous or an invalid transaction. All exceptions must be handled appropriately.

2. Try to add a private instance field that represents the PIN associated to a bank account. The driver bank teller system must serve only valid bank customers (their bank account is found given the account identifier and PIN matched) and reject invalid access attempts to the system.