Academic Honesty Rules for This Assignment

This assignment is individual work. It is an assessment of your ability. You may:

- Use the textbook and class videos, including code in the textbook and videos.
- Use the documentation at docs.oracle.com/en/java/ as a resource. Do not borrow code from the documentation.
- Ask me questions.

It is a violation of the class academic honesty policy to discuss this assignment with any other person (in person or on the Internet), search for answers on the Internet, or use sources other than the ones listed above. Please ask if you have any questions about this.

Specifications

Problem Description

For this assignment, you will code a class called `SecureBankAccount` to the specifications below, and a `JUnit` test suite to unit test the class.

`SecureBankAccount.java`

This class will represent a bank account that allows only a certain number of transactions of a certain type before it is "locked." A locked account can be reset. The account contains a balance, which is of type double. The account allows making deposits, making withdrawals, and accessing the balance. If withdrawals reach or exceed $500, the account is locked*. If there are two balance requests, the account is locked. If the account Is locked, it can be reset (so that withdrawals are now allowed up to the maximum and two balance requests can be made). The account cannot be reset if it is not locked. If the account is locked, no transactions are allowed (no deposits, withdrawals, or balance accesses) until it is reset.

Follow this class diagram for `SecureBankAccount`:

SecureBankAccount
- locked: boolean - balance: double - numBalanceRequests: int - amountWithdrawn: double
+ SecureBankAccount(startingBalance: double) + getBalance(): double + withdraw(amount: double): double + deposit(amount: double): double + reset(): boolean

* We will not complicate the class (and the unit testing) by constraining the amount withdrawn to $500, which is more realistic, but once it reaches or exceeds $500, the account is locked.

The constructor will set the balance field to the starting balance. The object begins in an unlocked state, with the number of balance requests and the amount withdrawn set to zero.

When the getBalance message is sent, the object checks to see if it is locked. If it is locked, it returns -1. If it is not locked, it returns the balance. It increments the number of balance requests if it is not locked. If the number of balance requests is 2, locked is set to true.

When the withdraw message is sent, the object checks to see if it is locked. It checks to see if the balance is high enough to cover the withdrawal. If both conditions are met, the amount to withdraw is subtracted from the balance, added to the amountWithdrawn, and returned from the method. If the conditions are not met, -1 is returned from the method and no changes are made to the state of the object. If money was successfully withdrawn, then the value of amountWithdrawn is checked against 500. If it meets or exceeds 500, locked is set to true.

With the deposit message is sent, the object checks to see if it is locked. If it is not, the amount passed is added to the balance and is returned from the method. If it is locked, then -1 is returned from the method and no changes are made to the balance.

When the reset message is sent, the object checks to see if it is locked. If it is, it is unlocked (locked set to false), and numbalanceRequests and amountWithdrawn are each reset to zero. It returns true. If it is not locked, no changes are made to the state of the object and it returns false.

If you wish, you may include private constants in your class for the maximum number of balance requests (2) and the withdrawal threshold (500).

`JUnit` tests

Design a thorough set of test cases to test the behavior of your `SecureBankAccount` class and implement them using `JUnit`. Provide thorough documentation within JUnit to explain the test cases.

Design constraints

Use only techniques that are covered in chapters 1-8, 10, or 11 of Gaddis/Muganda, Naik/Tripathy Chapter 3, or covered in this course. The JUnit tests should use assert statements and there should be no output except the JUnit reporting in Eclipse. Do not, under any circumstances, use System.exit, or any other code (e.g. return, break) that artificially breaks out of a logically controlled block of code (conditional statement or loop). Do not submit code that includes these constructs. Submit whatever you can write that does not include these constructs.

Turn in

Please turn in your `SecureBankAccount.java` file and your `JUnit` class or classes (java files), zipped together in a folder. (Please do not submit your entire Eclipse project.) Use the zip compression format. Please remove all package specifications from the top of your files.

Please include, in a Javadoc comment at the top of the `SecureBankAccount.java` file, your name, what I should call you, and pronunciation help for what you would like me to call you so I can try to pronounce your name correctly. Describe any errors in your program. In this comment you may also include anything else you'd like to tell me about the assignment.

Rubric

An exceptional-quality assignment will meet the following standards:

- Program submitted according to instructions, 10 points
  The assigned was submitted under the correct assignments link in Blackboard and packaged according to the *Turn In* section above.
- Meeting functional and design specifications, 80 points
  The JUnit test cases run successfully. The class is designed according to specifications. The test cases are thorough. The programmer has used only the allowed programming techniques. If the program misses specifications or does not function correctly, errors are acknowledged in the comment at the top of the program.
- Communicating with identifiers and white space, 5 points
  The program makes appropriate use of variables. Variables, constants, classes, and methods are named according to convention and are named for understandability and purpose. White space, both vertical and horizontal, is correctly used for readability and meets programming conventions.
- Communicating through documentation, 5 points
  The Java program contains comments including the programmer's name and date. There are block comments (as many as necessary) for each distinct block of code which accurately describe what the block is accomplishing by relating the code to the problem being solved. Javadoc is included and meets the javadoc standards.