Project #6: Filterable Accounts

Overview

Construct a hierarchy of classes representing savings and checking accounts as well as a means to 'filter' account objects.

Account Classes

Your solution will consist of 4 account classes as described below. It is your responsibility to define the exact relationship among those classes and translate that relationship into source code structure. There is a correct structure; the skeleton must be modified to reflect the correct relationships.

- `Account` – represents a general account with basic operations such as deposit, withdraw, etc.
- `Savings` – represents a savings account that must maintain a minimum balance.
- `Checking` – represents a checking account that has no minimum balance (this means a zero balance).
- `CappedChecking` – represents a checking account that disallows a balance above a particular maximum value.

Linking Accounts

A bank or credit typically allows a user the ability to link accounts together. That is, I might wish to link my checking account with my savings account. If I overdraw from my savings account, then the appropriate amount of money is transferred from my savings account into my checking account to prevent against the overdraw. In this program we wish to simulate this concept with the following requirements:

- A checking account may only be linked to a savings account (and vice versa).
- An account may only be linked to one other account.
- An account may not be linked to itself.
- An account may exist without a link to another account.

In this program, we consider a link to be bidirectional; that is, if account A is linked to account B, then B is only linked to account A. If the user wishes to link account C to A, then the bidirectional link from A to B must be unlinked before A is then linked to C.

Depositing / Withdrawing Funds and Exceptions

In this program we will use the following rule for deposits: if a deposit operation is unsuccessful a `LinkAccountException` will be thrown. For example, if a deposit exceeds the maximum amount allowed in the account and the account is not linked, an exception will be thrown. If the account is linked, the remaining monies will be transferred to the linked account.

If a withdrawal operation is unsuccessful an `InsufficientFundsException` will be thrown. For example, if a withdrawal exceeds the minimum of an account (and its linked account), an exception shall be thrown. In the case that a withdrawal results in an account dropping below the minimum, an appropriate amount of funds shall be transferred from the linked account to the current account in order to attain the minimum balance.

Filtering

In database computations, it is often necessary to filter a select set of objects from a larger set. In this project, each account will be labeled as `Filterable`. That is, you must appropriately define an abstract

class or interface representing the filterable notion. Each such filterable class must implement the method `boolean accept()` which returns true or false based on the following criteria:

- A checking account is deemed acceptable if it is linked to an account that is acceptable.
- A savings account is deemed acceptable if it maintains at least the minimum amount.

Bank

The `Bank` class will act as a repository for different accounts, both savings and checking. The class should be a simple class that:

- Maintains an `ArrayList` of accounts,
- The ability to add an account (`addAccount`),
- The ability to acquire the filtered list of accounts (`getFiltered` returns an `ArrayList` of `Account` objects). The filtered set of accounts is based on the `accept` method.
- An accessor method (`size`) that returns the number of accounts stored in the `Bank`.

Testing

A complete test suite has been provided. This includes `AccountTester`, `BankTester`, and the abstract parent class, `Tester`.

`Tester` – an abstract class defining several instance variables objects used by each inheriting testing class.

`AccountTester` – a concrete class that implements all functionality required to test the four account classes.

`BankTester` – a concrete class that implements all functionality required to test the `Bank` class.

The `Main` class initiates the complete test suite.

Requirements

The program must meet all the requirements as tested from `main` via the testing classes and described on the first page of this document.

There are also a few other requirements.

- The minimum amount for a savings account is $100. There is no maximum.
- The minimum amount for a checking account is $0. There is no maximum for a standard checking account, while capped checking accounts have maximum $10000.
- Any method that has an object as a parameter must be checked to verify the object is not `null`.

Specifications

For clarity, you will implement the following methods in the corresponding classes. You may implement accessor methods as required.

Class	Methods
`Bank`	`addAccount`, `getFiltered`, `size` as described above.
Account	`link` – given an account `acct`, link `this` account with `acct`.
	`unlink` – unlinks `this` account with `acct`.
	`deposit` – deposits a given amount of money into the account as described previously.

	`withdraw` – withdraw a given amount of money from the account as described previously.
	`getMinimum` – returns the minimum amount of money required by the particular account.
Checking	`accept` – as described previously.
	`link` – given an account `acct`, link `this` account with `acct`.
CappedChecking	`deposit` – deposits a given amount of money into the account as described previously.
	`link` – given an account `acct`, link `this` account with `acct`.

Recommendations

- Review the code that has been provided in the skeleton.
- In the corresponding `Tester` (`AccountTester` or `BankTester`), you may opt to comment out any of the tests in `runAll` while writing / testing your code.
- Test all account code first and perform bank tests second.

Submitting

Header Comments

Your program must use the following standard comment at the top of *each source code file*. Copy and paste this comment and modify the parenthesized values accordingly.

```
/*
* @author (Student Name)
* <p> (File Name)
* <p> (Assignment)
* <p> (Describe, in general, the code contained.)
*/
```

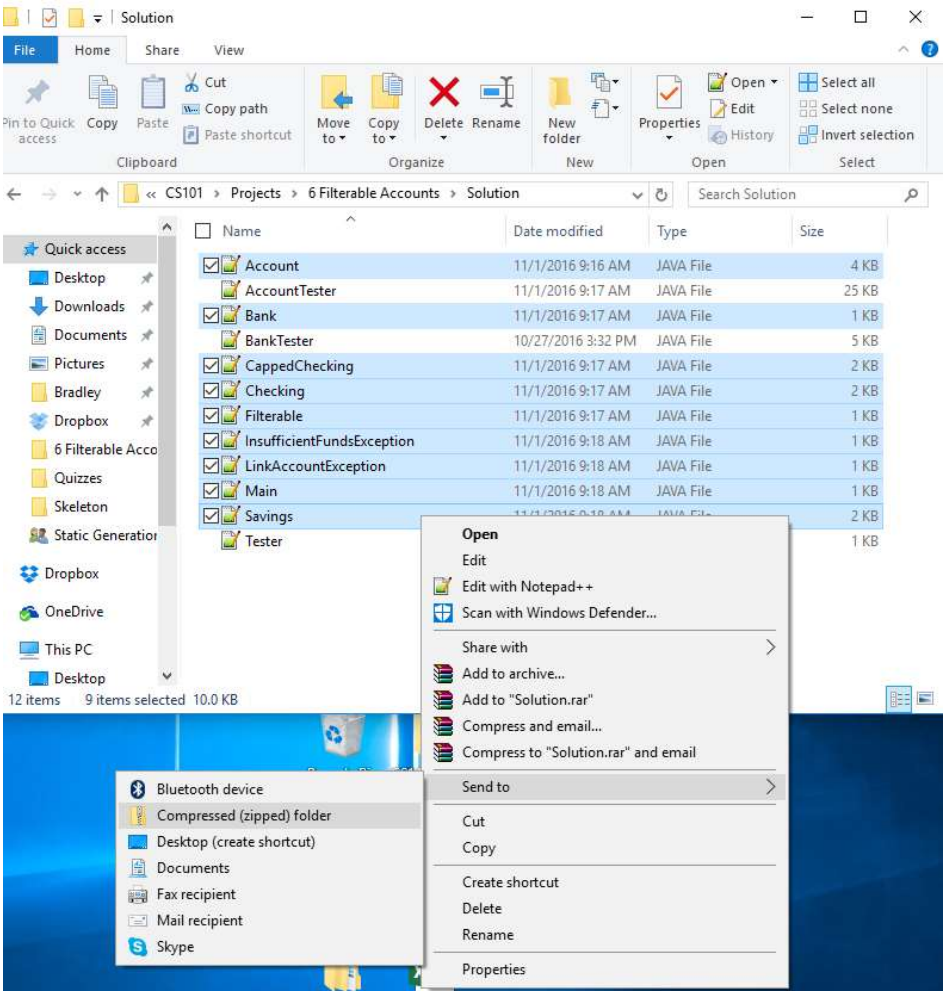
Inline Comments

Please comment your code with a reasonable amount of comments throughout the program. Each block of code (3-4 or more lines in sequence) in a method should be commented.

Although it is an issue of style and preference, please avoid long comments to the right of lines of source code. Long, ubiquitous comments to the right of code will result in a deduction.

Final Submission File

Create a zip file (`proj6.zip`) containing only the source code files. In Windows, (1) ***select all the non-testing source files themselves*** (not the containing folder), (2) right-click, and (3) Send to > Compressed (zipped) folder. Again, DO NOT submit `Tester`, `AccountTester`, and `BankTester`.



Submit your zip as directed by your instructor. Be sure to review the university policy on academic dishonesty: this is an individual project.