CS 251 Intermediate Programming
Lab 4: Exceptions

Brooke Chenoweth

Spring 2024

This is a small quick assignment designed to get you more familiar with creating and using exceptions.

Problem description

Let's consider a bank account. What would we need to represent one as a class? At the very least, we probably should have a bank account number (so we can tell my bank account apart from yours, even if we currently have the same balance) and a value for the current balance. We probably should have accessors for those fields.

What should we be able to do with a bank account? Well, we want to be able to put money in and take money out, so we should have deposit and withdraw methods that will update the current balance by some amount. The bank generally frowns on withdrawing more money than is in the account, so if someone tries to do that, we'll throw an exception. (Of course, all responsible account holders would check the balance before taking out some cash, but we don't want this exceptional case to sneak by.)

Also, we probably shouldn't be able to deposit or withdraw a negative amount. (Withdrawing a negative amount to increase your account balance would be frowned upon at most banks.) This is something that could have been checked for by the calling program, but to be extra careful we will throw a particular *unchecked* exception from the standard libraries called an `IllegalArgumentException` if we try to pass a negative amount into those methods.

What do you need to do?

1. Write a `InsufficientFundsException` class.
    - This class should extend `Exception`, since it is going to be a checked exception.
    - The constructor should take the amount of the shortfall (as a `double`) and store it in a member variable.
    - We want to see some sort of message like "You need more money!" in the stack trace, so call a `Exception` constructor that takes a message by using the `super`

keyword at the beginning of your class's constructor and pass it an appropriate message string.

- The class should have a `getShortfallAmount` method to access the shortfall amount given in the constructor. This method should return a `double`.

2. Write a `BankAccount` class.

   - The constructor should take a `int` for the account ID number. Bank accounts should initially have a balance of zero.
   - Provide a `getAccountNumber` method to return the `int` account number.
   - Provide a `getBalance` method to return the current account balance. This value should be a `double`.
   - The class should have a `depositFunds` method that takes a `double` amount and adds it to the current account balance. This method returns nothing.

     If the amount attempted to deposit is negative, this method will throw an `IllegalArgumentException`. Since this is an unchecked exception type, your method does not need to explicitly state that it throws that type of exception.
   - The class should have a `withdrawFunds` method that takes a `double` amount and subtracts it from the current account balance. This method returns nothing.

     If the amount attempted to withdraw is greater than the current account balance, this method will throw an `InsufficientFundsException`.

     If the amount attempted to withdraw is negative, this method will throw an `IllegalArgumentException`. Since this is an unchecked exception type, your method does not need to explicitly state that it throws that type of exception.

3. Test your classes.

   - I have provided you with a `BankingTest` class to test your `BankAccount` implementation. It creates a bank account, deposits amount of money, and attempts to withdraw three easy payments of $19.95 (just can't resist those infomercials!), printing out the balance along the way.
   - If you call `BankingTest` with a large enough initial deposit value, the program will run without error and you'll see something like the following.

```
$ java BankingTest 100.00
Account 101 balance = $100.00
Withdraw payment 1 of $19.95
Account 101 balance = $80.05
Withdraw payment 2 of $19.95
Account 101 balance = $60.10
Withdraw payment 3 of $19.95
Account 101 balance = $40.15
Finished payments
Account 101 balance = $40.15
```

```
Done banking for now
Account 101 balance = $40.15
```

- On the other hand, if you don't deposit enough funds, you won't be able to withdraw all the money and your `BankAccount` will throw an exception.[1]

  ```
  $ java BankingTest 25.00
  Account 101 balance = $25.00
  Withdraw payment 1 of $19.95
  Account 101 balance = $5.05
  Withdraw payment 2 of $19.95
  Sorry, but you are short $14.90
  InsufficientFundsException: You need more money!
  at BankAccount.withdrawFunds(BankAccount.java:26)
  at BankingTest.main(BankingTest.java:53)
  Done banking for now
  Account 101 balance = $5.05
  ```

  Notice that we don't actually change the bank balance when we run short of funds.

- If you try to deposit a negative amount, you won't even be able to get started banking because of the illegal argument.

  ```
  $ java BankingTest -10.0
  Exception in thread "main" java.lang.IllegalArgumentException:
                            attempted to deposit negative amount: -10.0
  at BankAccount.depositFunds(BankAccount.java:11)
  at BankingTest.main(BankingTest.java:35)
  ```

- Try testing your classes in other ways.
  - What happens if you make more than one `BankAccount`?
  - What about if you alternate some deposits and withdrawals?
  - What other cases should we be checking?
- Do note that the `BankingTest` program will not actually work until you have implemented `BankAccount` and `InsufficientFundsException`.

Turning in your assignment

Once you are done with your assignment, use Canvas to turn in `BankAccount.java` and `InsufficientFundsException.java`. (Do not submit `BankingTest.java`, since you should not have changed it.[2])

[1] Exact details of the error message and specific line numbers in the stack trace will depend upon your BankAccount implementation details. I'm not expecting you to throw your exception on the same line in BankAccount.java as I did in my version.

[2] Well, you can change it much as you like as part of your testing, but your code should still work with the original version of the file.