COS 126 Programming Exam 2 (Spring 2021)

Instructions

Download zip now. Download the exam zip file from http://bit.ly/COS126PE2 and open it the same way as you do for your assignments. Open the folder PE2 in IntelliJ, *but do not examine the Java files yet*.

Before the exam. Read this page of instructions before the exam begins, but **do not** start *(even by reading the next page)* until so instructed.

Zoom. Please remain in Zoom during the exam. Leave your **camera off** and **mute** your mic. If you have a question about the exam content, send a **private** chat to:

- Adam Finkelstein, if your last name begins A-L
- Alan Kaplan, if your last name begins M-Z

Emergencies. If you have an emergency, contact Kobi Kaplan, either via Zoom chat or at kskaplan@princeton.edu.

Time. You have 50 minutes to complete the exam and upload the required files with your solution.

Recommended plan. This exam follows a step-by-step process. Unlike other exams where you might find it advantageous to read the whole exam before you start, here we recommend you read each step and implement it as you go along. Also you should upload partial solutions a few times during the exam, just as a precaution.

What code to edit. The provided code contains several classes to get you started. We *strongly recommend* that you only add/modify code between the pairs of comments indicating "STUDENT CODE BEGIN" and "STUDENT CODE END". This is the easiest way to solve the tasks in this exam. **Do not** modify the existing method signatures.

Resources. You may only use: your textbook, the booksite, your notes, your code from programming assignments and precepts, the code on the COS 126 course website, materials from lectures, labs and precepts, and Ed.

No communication is permitted (e.g., talking, texting, etc.) during the exam, except with course staff.

Do not discuss later. Due to multiple exam times, and various conflicts, some of your peers will take the exam at a different time. Do not discuss the exam contents with anyone *(not even other students that you know already took the exam!)* until after the graded exams are returned.

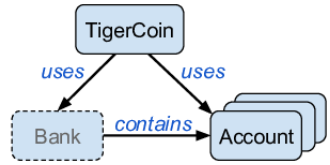
Honor Code pledge. Before submitting your solution, you must "electronically sign" the honor code by retyping the pledge and then your name in file `honor-code.txt`, and upload it with your solution.

Submissions. Submit your work here on TigerFile. You are responsible for submitting your solution before the time is called at the end of the exam.

Grading. Your program will be graded mainly on correctness. You will lose a substantial fraction of your overall grade if your program does not compile, or if it crashes on typical inputs. Clarity (including comments), design, and efficiency are secondary concerns with regards to grading on this timed exam. Nevertheless, writing clear code is always important, and will generally help you understand your own code better.

Getting Started

You are an engineer at a bank, writing software to support a new crypto-currency called TigerCoin (logo below). Your bank handles a series of accounts and transactions like deposits, withdrawals and transfers. Your project folder contains three Java files:



- **Account.java** – records information for a single account. You will implement several methods in this class.
- **Bank.java** – keeps track of all accounts. You do not need to examine this class, and **do not** modify it.
- **TigerCoin.java** – handles all transactions. You will implement some methods in this class.

Part 1

Edit the file `Account.java` to implement a mutable abstract data type `Account` that consists of an account holder's name (a String, like "Alice") and their current TigerCoin balance (a double, like 12345.67). **Do not** change the API – the public interface should remain unchanged. For all parts of this exam, you may assume that arguments representing TigerCoin funds (e.g., `amount` in the methods below) are non-negative – there are no transactions for negative funds. You may also assume that any name is a String containing a sequence of letters/digits (no spaces or punctuation), like "Alice" or "Alice2021". Implement these methods:

- `public Account(String name)` – Constructor: create an account with the given name and initial balance 0.0.
- `public String toString()` – Return a String representation in this format: `<name>: $<balance>`, for example, `Alice: $12,345.67`. Note the space after the colon! *(Hint: see the helpful method `formatTC`.)*
- `public void deposit(double amount)` – Increase the balance in this account by the given amount.
- `public boolean sufficientFunds(double amount)` – Return `false` if withdrawing the given amount would cause the account balance to be negative. Otherwise return true.
- `public boolean withdraw(double amount)` – Subtract the given amount from the account balance and return true, provided that this would not cause the balance to become negative. Otherwise, do not change the balance and return false.

Test your implementation using the provided `main` method, which should produce the following output. You are free to change this method to test your code however you like (as long as it compiles).

```
% javac-introcs Account.java
% java-introcs Account
Created account -- Alice: $0.00
After deposit of 300 -- Alice: $312.50
Does Alice have sufficient funds to withdraw 250? true
Does Alice have sufficient funds to withdraw 600? false
After withdrawing 250 -- Alice: $62.50
```

Upload through Part 1...

At this point you made many changes in the file `Account.java`. *Upload your partially completed exam, by uploading this file.* Also be sure you have already signed and uploaded the **HONOR CODE** in the file `honor-code.txt`.

Part 2

The remaining parts of this exam will be completed by editing code in `TigerCoin.java`. The `TigerCoin` object uses a `Bank` object to keep track of all its accounts. However, you do not need to understand how the Bank class works – it's just there to help. The TigerCoin `main` method reads lines from standard input and processes transactions one line at a time until it reaches the end of input. Each line has two words and then an amount, as in the example file `input4.txt` in the "transactions" folder:

```
DEPOSIT Alice 400
DEPOSIT Bob 300
WITHDRAW Bob 200
WITHDRAW Bob 200
```

The first word can be DEPOSIT or WITHDRAW, and the second word is a name of an account. The provided methods (`processTransaction` through `main`) in `TigerCoin.java` handle the processing of this input file – and creating accounts as needed. You do not need to read or understand these methods during the exam, but feel free to peruse them later. The method `processTransaction` calls several functions that you will implement.

Implement these two methods in `TigerCoin.java`:

- `public void depositInAccount(String name, double amount)` – Make a deposit of the given amount in the account with the given name. First you need to find the `Account` object with the given name. (Hint: *use the `getAccount` method for this object.*) Next call the `deposit` method for that account object.
- `public boolean withdrawFromAccount(String name, double amount)` – Make a withdrawal of this amount from the account with this name. Return `true` if funds are sufficient for withdrawal (otherwise `false`).

Below is an example of running this program after these methods are complete. Notice that Bob's second WITHDRAW fails because he does not have sufficient funds.

```
% javac-introcs TigerCoin.java
% java-introcs TigerCoin < transactions/input4.txt
DEPOSIT -> Alice : 400.0 (success: true)
DEPOSIT -> Bob : 300.0 (success: true)
WITHDRAW -> Bob : 200.0 (success: true)
WITHDRAW -> Bob : 200.0 (success: false)
All account balances:
Alice: $400.00
Bob: $100.00
```

Upload through Part 2...

This might be a good time to upload your largely completed exam.

Part 3 (Challenge - 5%)

Congratulations if you made it this far! Completing this next part may be difficult on this timed exam. It is meant to provide an extra challenge only for those of you who have sufficient time. You will only receive credit for this final part if your previous steps are all solved correctly; and even then a perfect solution to this part will only be worth 5% of the overall exam grade. So only attempt this part if you are confident in your previous solutions.

In this part you will handle a new kind of transaction called a *transfer* where money is moved from one account to another account. Implement the method:

- `public boolean transferFromOneToAnother(String fromName, String toName, double amount)` – Make a transfer from one account (fromName) to another (toName). Return `true` if funds are sufficient for transfer (otherwise `false`). (Hint: this is similar to the two methods from Part 2, but you need to think a bit about the success/failure cases.)

In the input file, transfers are indicated by listing two account handles and an amount, like lines 2, 4 and 6 of the file `input6.txt` where funds are transferred from Alice to Bob and *vice versa*:

```
DEPOSIT Alice 300
Alice Bob 150
DEPOSIT Alice 400
Alice Bob 250
WITHDRAW Bob 200
Bob Alice 600
```

The result of running it should look like this:

```
% javac-introcs TigerCoin.java
% java-introcs TigerCoin < transactions/input6.txt
DEPOSIT -> Alice : 300.0 (success: true)
Alice -> Bob : 150.0 (success: true)
DEPOSIT -> Alice : 400.0 (success: true)
Alice -> Bob : 250.0 (success: true)
WITHDRAW -> Bob : 200.0 (success: true)
Bob -> Alice : 600.0 (success: false)
All account balances:
Alice: $300.00
Bob: $200.00
```

Finishing Up

If you made it this far, you are done! Verify that you wrote and electronically signed the Honor Code pledge in the file `honor-code.txt`. Upload it together with your final versions of both `Account.java` and `TigerCoin.java`.

If you still have time, you can try out your `TigerCoin` class on larger provided files containing 100 or 1000 transactions, using the commands below. These commands save the output of your program in files like `my-output100.txt`. You could then compare those files with the solution files with similar names in the “transactions” folder.

```
% java-introcs TigerCoin < transactions/input100.txt > my-output100.txt
% java-introcs TigerCoin < transactions/input1000.txt > my-output1000.txt
```