CMPS 1600 Introduction to Computer Science II – Spring 19
Lab 5

Objectives

This lab is essentially an exercise that you completed in the beginning of the semester in Java. The goal is to express your existing knowledge of concepts in a new language (C++). You may find it helpful to refer to slides on Canvas and C++ chapters on Zybook when you have questions on C++ syntax.

First C++ program

Create a class `BankAccount`. An object of this class should have a person's `name` and a `balance`, stored as a `double`. Create methods `getName` and `getBalance`, and create `deposit` and `withdraw` methods to add and subtract a specified amount from the `balance`. Finally, create a `toString` method that gives the name and balance of the account.
Test your class. In the main() function, create two accounts: Create a `starter` with zero balance, and a name of your choice. Also create a `advanced` with a $500 initial balance and a name of your choice. Deposit $1000 into the `starter`. Withdraw $100 from the `advanced`. Now transfer $200 from the `starter` to the `advanced` using withdraw and deposit methods provided by the class. Print out both accounts and manually verify that the balances are as expected.

Inheritance

1. Now create another class named `SavingsAccount`. This should be a subclass of `BankAccount`. A `SavingsAccount` should have an interest rate that is specified in its constructor. Furthermore, there should be a method, `addInterest`, that deposits interest into account, based on the account balance and interest rate.

2. Create another subclass of `BankAccount` named `CheckingAccount`. A `CheckingAccount` should keep track of the number of transactions made (deposits and withdrawals). Name the field `transactionCount`. Also, the set fee for transactions is three dollars, so create a `static final` field, `TRANSACTION_FEE`, that is set to 3.0. Finally, create a method, `deductFees`, that deducts fees from the account balance based on the number of transaction. This method should reset the `transactionCount` to zero.

3. Now change your main method to test the classes that you've written. Create a `SavingsAccount` with zero balance, a 1% interest rate, and a name of your choice. Also create a `CheckingAccount` with a $500 initial balance and a name of your choice. Deposit $1000 into the `SavingsAccount`. Withdraw $100 from the `CheckingAccount`. Now transfer $200 from the `SavingsAccount` to the `CheckingAccount` using withdraw and deposit methods. Print out both accounts. Now add interest to the `SavingsAccount` and deduct fees from the

`CheckingAccount`. Print out both accounts and manually verify that the balances are as expected.

Optional extra credit (worth up to another lab problem): Create another class, Bank that should simulate work of a bank. The user should have an opportunity to create several accounts of different types using a text menu-based user interface, and to perform operations with those accounts. The system should support having multiple users, and multiple accounts for each user without explicit bound on those numbers (i.e. don't use constants for possible number of users or accounts). Implementation is up to you.

Submission

As always, make sure your code is clean, well-readable and well-commented. Your class definitions and class implementations should be in different files. So you'll submit BankAccount.h, BankAccount.cpp, CheckingAccount.h, CheckingAccount.cpp, SavingsAccount.h, SavingsAccount.cpp, that contain class declarations and implementations, and main.cpp with testing functionality.