Lab 04 - Building Classes

Instructions:

- **The lab requires completing a few tasks.**
- **Your submissions must be submitted to the GitHub repository in the Lab04 directory.**
- **Cheating of any kind is prohibited and will not be tolerated.**
- **Violating and failing to follow any of the rules will result in an automatic zero (0) for the lab.**

TO ACKNOWLEDGE THAT YOU HAVE READ AND UNDERSTOOD THE INSTRUCTIONS ABOVE, AT THE BEGINNING OF YOUR SUBMISSION(S), ADD A COMMENT THAT CONSISTS OF YOUR NAME AND THE DATE.

Grading

Task	Maximum Points	Points Earned
1	2.5	
2	2.5	
Total	5	

Note: solutions will be provided for tasks colored blue only.

The bank, Sigma, provides both checking and savings accounts. The fee schedule for its accounts are

Fee	Charge
Checking	
Monthly	$10 if balance under $500.00 None if balance $500.00 or more
Overdraft	$5.00 per transaction
Savings	
Monthly	$3.50 if balance under $500.00 None if balance $500.00 or more

Additionally, its savings account receives a 5% APY. Your objective is to create an application for Sigma Bank that allows users to start and perform transactions on checking and savings accounts. To accomplish your objective, you are required to create three classes named *Account*, *Checking*, and *Savings*. They **must** contain

- **Account**
  - A private class field named *acc_num_gen* initialized to 162410010000.
  - A private field named *balance*.
  - A private field named *accountnumber*.
  - A public default constructor that initializes *balance* and *accountnumber* to 0 and *acc_num_gen*, respectively, and increments *acc_num_gen* by 1.
  - A public overloaded constructor that takes a float parameter that initializes *balance* and *accountnumber* to the parameter and *acc_num_gen*, respectively, and increments *acc_num_gen* by 1. If the parameter is less than 0, *balance* is initialized to 0.
  - A public Boolean method named `Deposit()` that takes a float parameter. It adds the parameter to *balance* and returns true if the parameter is not negative; otherwise, it returns false.
  - A public float method named `Withdraw()` that takes a float parameter. It subtracts the parameter from *balance* and returns the parameter if the parameter is not negative and *balance* minus the parameter is not negative; otherwise, it returns 0.
  - A public getter method for *balance* named `Balance()`.
  - A public getter method for *accountnumber* named `Account()`.
  - A public void method named `Process()` that takes no parameters and does nothing.
  - A public toString method that takes no parameters or friend ostream operator if applicable, that displays the string

    "*x*:\n$ *y*"

    where *x* and *y* are the values of *accountnumber* and *balance*, respectively.
- **Checking**
  - It (publicly) inherits *Account*
  - A private field named *fee*.
  - A public default constructor that initializes *fee*, *balance*, *accountnumber* to 0, 0 and *acc_num_gen*, respectively, and increments *acc_num_gen* by 1.
  - A public overloaded constructor that takes a float parameter that initializes *fee*, *balance* and *accountnumber* to 0, the parameter and *acc_num_gen*, respectively, and increments *acc_num_gen* by 1. If the parameter is less than 0, *balance* is initialized to 0.
  - A public overridden `Deposit()` method. It adds the parameter minus *fee* to *balance*, assigns 0 to *fee*, and returns true if the parameter is not negative and the parameter minus *fee* is not negative; otherwise, it subtracts the parameter from *fee* and returns false if the parameter is not negative; otherwise, it returns false.
  - A public overridden `Withdraw()` method. It subtracts the parameter from *balance* and returns the parameter if the parameter is not negative and *balance* minus the parameter is not negative; otherwise, it adds 5 and the parameter minus *balance* to *fee*, assigns 0 to *balance* and returns the previous value of *balance* if the parameter is not negative and *balance* minus the parameter is negative; otherwise, it returns 0.
  - A public getter method for *fee* named `Fee()`.
  - A public overridden `Process()` method. It adds 10 to *fee* if *balance* is less than 500.
  - A public overridden toString method or friend ostream operator if applicable, that displays the string

    "*x*:\n$ *y*"

    where *x* and *y* are the values of *accountnumber* and *balance* minus *fee*, respectively.

- **Savings**
  - It (publicly) inherits *Account*
  - A private field named *fee*.
  - A public default constructor that initializes *fee*, *balance*, *accountnumber* to 0, 0 and *acc_num_gen*, respectively, and increments *acc_num_gen* by 1.
  - A public overloaded constructor that takes a float parameter that initializes *fee*, *balance* and *accountnumber* to 0, the parameter and *acc_num_gen*, respectively, and increments *acc_num_gen* by 1. If the parameter is less than 0, *balance* is initialized to 0.
  - A public overridden `Deposit()` method. It adds the parameter minus *fee* to *balance*, assigns 0 to *fee*, and returns true if the parameter is not negative and the parameter minus *fee* is not negative; otherwise, it subtracts the parameter from *fee* and returns false if the parameter is not negative; otherwise, it returns false.
  - A public getter method for *fee* named `Fee()`.
  - A public overridden `Process()` method. It adds 3.50 to *fee* if *balance* is less than 500 and assigns the product of *balance* and 1.00426 to *balance*.
  - A public overridden toString method or friend ostream operator if applicable, that displays the string

    "*x*:\n$ *y*"

    where *x* and *y* are the values of *accountnumber* and *balance* minus *fee*, respectively.

Afterward, define a function named *app()* that takes no parameters. It prompts a menu to a user until the user chooses the quit option. The other menu options must be to '`start an account`' and '`perform a transaction`'.

If the user chooses the '`start an account`' option, it should prompt the user for the type of account and the initial balance. Afterward, it displays the account object created.

If the user chooses the '`perform a transaction`' option, it should initially prompt the user for the account number. If the account is found, it should prompt the user for the type of transaction (deposit or withdraw) and the amount. If the transaction is successful, it should display the new balance of the account; otherwise, it should state that an error occurred.

Task 1

Write the above program in C++.

Task 2

Write the above program in Ruby.

Extra Credit

Modify the *Account* class to keep track of the number of accounts created. It must have a class method that returns the number of accounts. Likewise, modify `app()` to display the number of available accounts before each menu prompt.