Programming Project 1

Due Friday, February 16 at 11:59pm

IMPORTANT: *Read the* ***Do's and Dont's*** *in the* ***Course Honor Policy*** *found on blackboard.*

I. Overview

The theme of this assignment is to write code to create an account for a student at a university. The purpose is to give you practice writing classes, fields, methods, conditional statements, and constructors. You will also be introduced to polymorphism.

II. Code Readability (20% of your project grade)

Once upon a time, getting a program to work was the only goal of programming. That was before computers took over the world. Now, with highly complex software running much of our lives, the industry has learned that computer code is a written document that must be able to communicate to other humans what the code is doing. If the program is too hard for a human to quickly understand, the company does not want it.

Most companies enforce readable code by having very strict rules about how the program should look. This class will do the same, but will not be quite as strict so you can have some freedom for developing your own style.

To receive the full readability marks, your code must follow the following guideline:

- All variables (fields, parameters, local variables) must be given appropriate and descriptive names.
- All variable and method names must start with a lowercase letter. All class names must start with an uppercase letter.
- The class body should be organized so that all the fields are at the top of the file, the constructors are next, and then the rest of the methods.
- There should not be two statements on the same line.
- All code must be properly indented (see page 689 of the Lewis book for an example of good style). The amount of indentation is up to you, but it should be at least 2 spaces, and it must be used consistently throughout the code.
- You must be consistent in your use of {, }. The closing } must be on its own line and indented the same amount as the line containing the opening {.
- There must be an empty line between each method.
- There must be a space separating each operator from its operands as well as a space after each comma.
- There must be a comment at the top of the file that includes both your name and a description of what the class represents.

- There must be a comment directly above each method that, in one or two lines, states *what* task the method is doing, not how it is doing it. Do not directly copy the homework instructions.
- There must be a comment directly above each field that, in one line, states what the field is storing.
- There must be a comment either above or to the right of each non-field variable indicating what the variable is storing. Any comments placed to the right should be aligned so they start on the same column.

III. Program Testing Document (20% of your project grade)

Once upon a time, companies thought errors in code were only a minor inconvenience. That was before software glitches started killing people and destroying companies. Now, standard practice is that all code must be thoroughly verified before a company is willing to release it. At a large number of firms, programmers are required to *first* design the test cases the program must pass and *then* start writing the code. We will not be that strict in this class, but you will need to test your code.

To receive full testing marks, you must write a testing report that shows that you thoroughly tested every method of the program. The report should be a short English description for each test (what you are testing and what the expected result of the test is) followd by the actual result of the test. If you are using DrJava, you can enter the test into the interactions pane and then copy and paste the test code plus the result to your report. If you fail to complete the program, your report should indicate how you would go about testing the incomplete methods.

Your grade on the testing report is how thoroughly you test your code, not how correctly your code runs. If your code is not 100% correct then your report *should* show an incorrect result to some test. Testing methods that do not have conditional statements should be pretty straightforward, but you need to put thought into testing methods with conditional statements so that each branch of the if-statement is tested.

Hint 1: You can test multiple methods with one test. For example, you can test each setter/getter method pair together or you can test constructors and getter methods together.

Hint 2: Do not put off testing to the end! Test each method after you complete it. Many methods depend on other methods. Delaying testing could mean cascading errors that cause your whole project to collapse. Since you need to test anyway, copy the tests you do into a document, and you are most of the way to completing your report.

If you are not using DrJava, you are allowed (but not required) create a separate class that tests your program. You must still write a testing report that documents the tests you do in this class. Do not place testing code into a main method of the classes below. That is not the purpose of a main method.

IV. Java Programming (55% of your grade)

Guidelines for the program:

- All the listed methods must be public.
- You will need to create several instance fields to store data, and every field must be private.

- All fields must be initialized to an appropriate value. They can be initialized either as part if the field declaration or in the constructor. Even if you feel that the default value provided by Java is appropriate, you still must give an explicit initialization.
- Any method whose name begins with set should only assign a value to an appropriately named field. The method should do no other processing. Any processing described in a set method description below is for information only. That actual processing will be done by other methods.
- Any method whose name begins with get should only return the appropriate value. No other processing should occur in these methods.
- Your class must include only the methods listed. You may not write any other methods.
- The behavior of your methods must match the descriptions below.
- You should not write any loops in your program (though loops are allowed in the testing code).

The program: You will create the following four classes. The classes are listed in order of challenge from more straightforward to more challenging.

1. Account
   The Account represents a general purpose account that records the amount of money an individual owes. The Account class will need instance fields to keep track of an account number, the current balance for the account, and a balance limit for the account. The class will have the following methods:

   1. getAccountNumber
      Takes no input and returns a String. Returns the account number of the account. (The account number will never change.)
   2. **setBalance**
      Takes a single double value as input and returns nothing. Changes the balance to be the input value.
   3. getBalance
      Takes no input and returns a double. Returns the current balance of the account.
   4. charge
      Takes a single double value as input and returns nothing. The balance is increased by the input amount.
   5. credit
      Takes a single double value as input and returns nothing. The balance is decreased by the input amount.
   6. setBalanceLimit
      Takes a single int value as input and returns nothing. Changes the balance limit to be the input value.
   7. getBalanceLimit
      Takes no input and returns an int. Returns the balance limit of the account.

   The Account class will have two constructors:
   - The first constructor takes a single String input that is the account number.
   - The second constructor takes two inputs, a String that is the account number and an int that is the balance limit.
2. LibraryAccount
   A LibraryAccount is the financial record of a library patron. It records whether the patron has any overdue material and how much money, in fines, has been assessed to the patron. The LibraryAccount class

should *extend* the Account class. Besides including all the methods that Account has, the LibraryAccount will have the following additional methods:

1. setBookFine
   Takes a single double value as input and returns nothing. The input value will be the size of the fine that is assessed each day a book is overdue.
2. getBookFine
   Takes no input and returns a double. Returns the amount that will be fined for each day a book is overdue.
3. setReserveFine
   Takes a single double value as input and returns nothing. The input value will be the size of the fine that is assessed each day a reserved item is overdue.
4. getReserveFine
   Takes no input and returns a double. Returns the amount that will be fined for each day a reserved item is overdue.
5. incrementOverdueBooks
   Takes no input and returns nothing. Increases by one the number of overdue books on the account.
6. decrementOverdueBooks
   Takes no input and returns nothing. Decreases by one the number of overdue books on the account. Does not decrease the count below zero.
7. **setNumberOverdueBooks**
   Takes a single int value as input and returns nothing. Changes the number of overdue books to be the input value.
8. getNumberOverdueBooks
   Takes no input and returns an int. Returns the number of overdue books on the account.
9. incrementOverdueReserve
   Takes no input and returns nothing. Increases by one the number of overdue reserved items on the account.
10. decrementOverdueReserve
    Takes no input and returns nothing. Decreases by one the number of overdue reserved items on the account. Does not decrease the count below zero.
11. **setNumberOverdueReserve**
    Takes a single int value as input and returns nothing. Changes the number of overdue reserves to be the input value.
12. getNumberOverdueReserve
    Takes no input and returns an int. Returns the number of overdue reserved items on the account.
13. canBorrow
    Takes no input and returns a boolean. Returns true if the account's balance is equal or less than the account's balance limit. Returns false otherwise.
14. endOfDay
    Takes no input and returns nothing. Increases the account's balance by the product of the number of overdue books and the book fine, and increases the account's balance by the product of the number of overdue reserved items and the reserved items fine.

The LibraryAccount should have two constructors.

- the first constructor takes a single String input that is the account number.
- the second constructor takes four inputs: a String that is the account number, an int that is the balance limit, a double that is the book fine, and a double that is the reserved items fine.

3. CreditAccount

The CreditAccount class represents a situation where a customer can borrow money. The money is to be repaid at the end of the month, but if it is not repaid, an interest penalty is assessed. The CreditAccount class should *extend* the Account class. Besides including all the methods that Account has, the CreditAccount class will have the following additional methods:

1. setInterestRate
   Takes a single double value as input and returns nothing. Changes the interest rate for the account to the input value.
2. getInterestRate
   Takes no input and returns a double. Returns the interest rate for the account.
3. **setMonthlyPayment**
   Takes a single double as input and returns nothing. Changes the monthly payment for the account to the input value.
4. getMonthlyPayment
   Takes no input and returns a double. Returns the monthly payment amount. The montly payment is the amount that must be paid this month to avoid receiving an interest charge on the account balance. The monthly payment is different from the balance.
5. **setAmountPaidThisMonth**
   Takes a single double value as input and returns nothing. Changes the amount paid this month for the account to the input value.
6. getAmountPaidThisMonth
   Takes no input and returns a double. Returns the amount that has been credited to the account this month.
7. endOfMonth
   Takes no input and returns nothing. First, the method should see if interest must be charged. If the amount paid this month is less than the monthly payment, the account's balance is increased by product of the interest rate and the balance, divided by 12.
   After checking for an interst charge, the method should reset the account by setting the monthly payment to be equal to the balance and setting the amount paid this month to zero.

Besides these methods, the CreditAccount class needs a different behavior for the credit method from that of the Account class.

- The credit method in CreditAccount must **decrease** the account balance and **increse** the amount paid this month by the input value.

The CreditAccount should have a single constructor that takes 2 inputs: a String that is the account number and a double that is the interest rate.

4. StudentAccount

The StudentAccount class represents the financial accounts of a student. A student's account is divided into

three parts: the amount they owe for tuition, the amount they owe for any dining charges, and the amount owed to the library for overdue items. the StudentAccount class should *extend* the Account class. Besudes incuding all the methods that Account has, the StudentAccount class will have the following additional methods:

1. setName
   Takes a single input that is a String value and returns nothing. Changes the account owner's name to the input value.
2. getName
   Takes no input and returns a String value. Returns the account owner's name.
3. setAddress
   Takes a single input that is a String value and returns nothing. Changes the account owner's address to the input value.
4. getAddress
   Takes no input and returns a String value. Returns the account owner's address.
5. setLibraryAccount
   Takes a single input that is a LibraryAccount value and returns nothing. Changes the library account associated with this account to the input value. There does not have to be a library account associated with this account so the input value may be null.
6. getLibraryAccount
   Takes no input and returns a LibraryAccount value. Returns the library account associated with this account.
7. setTuitionAccount
   Takes a single input that is a CreditAccount value and returns nothing. Changes the tuition account associated with this account to the input value. There does not have to be a tuition account associated with this account so the input value may be null.
8. getTuitionAccount
   Takes no input and returns a CreditAccount value. Returns the tuition account associated with this account.
9. setDiningAccount
   Takes a single input that is a CreditAccount value and returns nothing. Changes the dining account associated with this account to the input value. There does not have to be a dining account associated with this account so the input value may be null.
10. getDiningAccount
    Takes no input and returns a CreditAccount value. Returns the dining account associated with this account.

Besides these methods, the StudentAccount class needs to a different behavior for the following methods of Account:

- getBalance: the getBalance method of StudentAccount should sum the balances of the library account, tuition account, and dining account that are associated with this account (if those accounts exist). The method should then subtract the account's refund amount **(represented by the balance of the student account itself)** from this sum and return that value.

- charge: the charge method of StudentAccount should first subtract the input amount by the accounts's refund amount. If this difference is positive and the tuition account exists, the tuition account's balance should be increased by the difference. Otherwise, the account's refund amount should be set to the negation of this difference.
- credit: the credit method of StudentAccount processes a payment to the student account, and the payment should be used to decrease the balances of the associated accounts in the following order: tuition, dining, library (if they exist). That is, you first apply the payment to reduce the tuition account balance. If there is any money left after reducing the tuition account, you reduce the dining account balance next and then the library account balance. If after reducing each existing account there is money left over, it should be added to the student account's refund amount. The total amount you apply to the associated accounts' balances should not exceed the input to this method.
  For the tuition and dining accounts, the account's balance should only be decreased until the account's amount paid this month equals the monthly payment.
  For the library account, the account's balance should not be reduced below zero.

In addition, the StudentAccount should have a single constructor that takes two String values as input. The first is the account number and the second is the account owner's name.

V. Submitting Your Project

Submit the .java files (not the .class files or the .java~ files) for each of your four classes plus the testing report on Canvas.