CompSci 251: Intermediate Computer Programming
Summer 2015

Programming Assignment 5
Due Monday, July 27, 11:59pm

1 Overview

In this assignment we will use inheritance to represent several different kinds of card payment methods. You will implement classes representing:

- A pre-paid gift card
- A credit card with a hard credit limit
- A debit card with opt-in overdraft (and a $35 fee)
- A debit card with overdraft protection linked to a secondary account

A test driver has been supplied for you. No user input or output will be necessary for this program.

2 Requirements

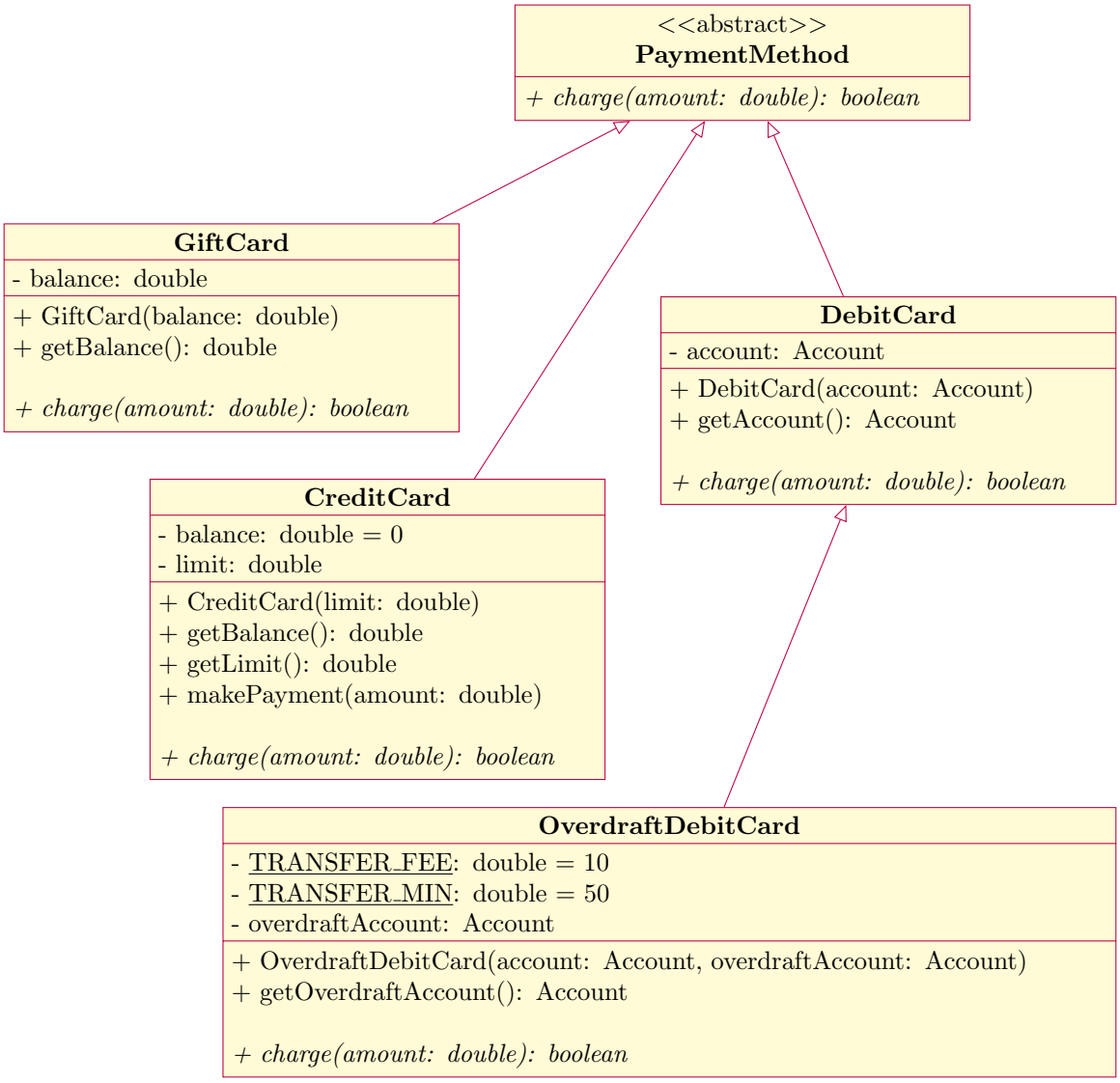
You will first need to implement an `Account` class, which will be linked to two kinds of debit cards (opt-in overdraft and overdraft protection). You should implement this class according to the following UML diagram. Be sure to include a small comment for at least each *non-trivial* method.

Account
- OVERDRAFT_FEE: double = 35 - balance: double = 0
+ getBalance(): double + canWithdraw(amount: double): boolean + withdraw(amount: double) + deposit(amount: double)

The `canWithdraw` method returns `true` if there are sufficient funds to withdraw `$amount`. The `deposit` and `withdraw` methods are basic mutator methods, except you will charge an additional $35 fee when there are insufficient funds for a withdrawl (of course, this will make insufficient funds even more insufficient, but who am I to argue with bank logic).

Notice that we have a static member `OVERDRAFT_FEE`. This is used to give a self-documenting name to an otherwise magic constant. **Do not** have numeric literals floating around in your program. They are hard to track down and are not easily explained by context. Give them a name, as we do here.

Next, you must implement a class hierarchy for payment methods described by the following UML diagram. Notice that all the concrete classes (directly or indirectly) inherit `PaymentMethod`. Also notice that `PaymentMethod` is marked as abstract (as it has no implementation for the `charge` method without knowing *what* kind of payment method it is). Since this class has *only* abstract members and no data, you could convert this abstract class into an interface (but is not required).



The `charge` method for *every* concrete class will either successfully charge the given amount (and possibly some related fee) and return `true`, or realize that there's a problem (insufficient funds, reached credit limit, etc) and return `false`. If `false` is returned no other observable changes should have taken place (if a card is declined, no funds should have moved).

The `GiftCard` payment method is pre-loaded with some amount set in the constructor. Charging a gift card is only successful if there is a sufficient balance on the card. The `CreditCard` payment method starts with a zero-balance and a hard limit set in the constructor. Charging a credit card is successful if the balance *after the charge* does not exceed the credit limit.

The `DebitCard` payment method is linked to a bank account. A charge is successful *always*, but the charge may incur an (opt-in) overdraft fee if the balance of the account becomes (or remains) negative.

The `OverdraftDebitCard` payment method is the most complex. The `charge` method will first check if there are insufficient funds in the primary account (the one defined in the superclass). If the primary account would overdraft, we attempt to move some amount of funds from the overdraft account to the primary account, and then process the charge as a normal debit card would. Remember that you can call a superclass method form a subclass - **do not** duplicate the logic in `DebitCard`'s `charge` method in the subclass).

Suppose that we are charging $150 to an account with only $35. We need an additional $115 so that we do not overdraft. In this specific case, we transfer $150 dollars from the overdraft account to the primary account, and charge the overdraft account a $10 fee. We transfer $150 because we must transfer *at least* $115, but must always transfer a multiple of $50.

Notice that if the overdraft account has insufficient funds for the transfer (**and** the fee) it **does not** do it, and instead charges a regular overdraft fee to the primary account. This behavior is **identical** to the superclass behavior.

Test Driver A test driver is supplied. You **must not** modify this file to receive full credit. If you've implemented everything correctly you should see the output `All tests passed!`. Otherwise, you will see a specific error message saying *which* of your implementation classes has an error. Passing these tests is not a guarantee that your code is 100% correct, it only shows that you did not make the common mistakes I assumed you might. You are encouraged to write some additional tests yourself.

3 Submission

Submit all Java files in a zip file to the D2L dropbox before the posted deadline.