Lab 04: Credit Card

COSC 102 - Spring '22

In this lab you will implement a class simulating the functionality of a *credit card*. The learning goals of this lab are:

- to define and write the object code which simulates the functionality of a `CreditCard` object (*i.e.* declare, define and update its instance variables and methods as well as its class counterparts).
- to write the client code that instantiates many `CreditCard` objects in order to utilize and test your object code.

1 Overview

A credit card is a payment card owned by an individual, used to make purchases. Each card holder is assigned a credit card with an *account number*, *CVV number*, *credit limit*, and *expiration date*; the holder is then able to "charge" purchases to this credit card. The below sections will define these terms describe how the credit card will function.

1.1 Definitions

We will define the terms for this lab as follows:

- **Account Number:** a numerical identifier unique to each credit card instance. Account numbers are serialized, meaning each new account gets assigned the next sequential account number.

  Note: since the account number is considered sensitive data, it is not made readily accessible to external code. This means it does not have a conventional accessor method (but is included in the `toString` information).
- **CVV:** a three-digit code on the back of the card used as a confirmation in certain transaction. For the sake of this lab, we will say the CVV *cannot contain leading zeroes*.

  Note: the CVV is guarded even more closely than the account number; it has no conventional accessor and is not included in the `toString()`.
- **Charge:** when a card holder makes a purchase with their credit card, the amount of the purchase is "charged" to their account, accruing to the card's balance (which must eventually be paid off).

  A charge which would cause the balance to exceed the card's credit limit is not allowed. Additionally, the payment processor requesting the charge must confirm the card's *CVV* in order for the charge to be successful.
- **Balance:** the sum of current, unpaid charges on a credit card. The balance *increases* when charges are made, and *decreases* when payments are made. The balance can equal but not exceed the credit limit.

  A credit card can have a negative balance if the owner overpays, but the balance cannot pass below negative the credit limit. For example, a card with a credit limit of `$500` can have a balance up to and including `-$500`.
- **Credit Limit:** dictates the maximum absolute balance (inclusive) that a credit card can have.
- **Payment:** a payment which pays off some, all, or beyond the total of outstanding charges. Payments *reduce* the current balance of the credit card account.
- **Expiration Date:** the date (month and year) when the card expires and is no longer usable for purchases. If a purchase is attempted using an expired card, the purchase fails. Cards are considered valid up to *and including* their expiration month. The expiration month and year are stored as two and four digit numbers, respectively.

1.2 Example Scenario

Below is an example scenario that shows how the charging, payment, and balance attributes work for the credit card:

- User *Jane Doe* opens a credit card account. The last created credit card prior to Jane was assigned an account number `2431`, therefore Jane's account number is `2432`. Jane's account has a credit limit of `$200`, an expiration date of `August 2023`, and the CVV is randomly generated as `451`.
- Jane makes a purchase for `$150` with her credit card. This means that a charge is made to the card for `$150`, resulting in a balance on the card of `$150` on her account.
- Jane attempts to make another purchase with her card for `$100`. The purchase is denied because the charge amount would cause Jane's balance to exceed her credit limit of `$200`.

- Jane makes a credit card payment for `$120`, reducing her balance to `$30`.
- Jane again attempts to make the purchase for `$100` with her credit card. This time the charge is successful, and her new balance is `$130`.

1.3 Your Implementation

Your `CreditCard` class implementation will be a generalized version of the above, having all of the attributes outlined above. Additionally, client code will will be able to make purchases and payments to a `CreditCard` object, as well as perform other operations such as extending expiration dates and merging accounts.

1.4 Starter File

Given to you is a starter **`CreditCard.java`** file, which contains the class declaration as well as the following two class functions; experiment with these functions to get familiar with how they work:

- **`static int getTodaysMonth()`**
- **`static int getTodaysYear()`**

2 Your Task

You will implement two classes in separate files: **`CreditCardClient`** and **`CreditCard`**, detailed below.

2.1 Credit Card Client

You will need to create a file named **`CreditCardClient.java`** where you will implement a `main` method to test the functionality of your `CreditCard`. **Test each function as you implement it**; write a variety of test cases to cover all the different scenarios your `CreditCard` must account for and provide comments that explain your coverage.

Furthermore, be sure to **read this entire document before you begin coding**. You will need to give careful consideration to the *order* in which you implement this functionality, so that you can test optimally and incrementally.

2.2 Credit Card

Your `CreditCard` must implement the following:

- **Constructor:** you will implement **three** constructors:
  - the first constructor accepts four arguments **in this order:**`(String accountName, int expirationMonth, int expirationYear, double creditLimit)`. Your constructor should set an account number instance variable. Account numbers are serialized as outlined previously and start at the number `2000`.
  - your second constructor will be the same as the first, except it will omit the *credit limit*, which will instead be assigned to the default limit, which is `$400`.
  - the third will omit *both* the *credit limit* and *expiration date*, the latter of which will instead be assigned to the default date, which is the current date plus three years.

  Your constructors must validate some of their arguments. If a negative or zero credit limit is provided, the default credit limit value is used.

  Additionally, if an invalid expiration date is provided, the default expiration date is used. If the month value is out of range, or the year is more or less than 10 years from the current year (regardless of month), the *entire* date (*i.e.* both the month and year) is considered invalid, and the default should be used.

  Finally, you must eliminate redundancy between your three constructors using *constructor redirection* – this means having certain constructors call other constructors. One constructor can call another using `this(...);` and inserting the appropriate arguments. Your goal is to ultimately funnel your constructor code into one place no matter which is called; 2 of the 3 constructors should only have one line of code in them (the call to `this`).

- `String toString()`: returns a `String` representation of the state of the referenced `CreditCard` object.

  Given an account for `Jane Doe`, with an account number of `#2017`, a CVV of `646`, a balance of `$411.87`, a credit limit of `$900`, and an expiration of `May 2024`, `toString()` should return the following `String`:

  ```
  Jane Doe (#2017): Current Balance = $411.87/$900.00, expires: 5/2024
  ```

  The dollar amounts do not have to display two decimal places.

- `getName()`, `getCreditLimit()`, `getExpireMonth()`, `getExpireYear()`, and `getBalance()`: Accessor methods which return their respective attribute (you **may not** add accessors for the CVV/account number)

- `boolean chargeCard(double amount, int cvv)`: accepts a charge amount and attempts to apply it to the referenced `CreditCard`, increasing the current balance. Returns `true` if the charge is successful, or `false` if the charge is not (*i.e.* denied). If the charge is unsuccessful, the balance is not modified.

  If the argument CVV doesn't match the reference card's CVV attribute, the charge fails. Additionally, there are **three other situations** in which a charge would fail – think about it and reread the document if you are unsure!

- `boolean payCard(double amount)`: accepts a payment amount, and attempts to apply it to the referenced `CreditCard`, lowering the current balance. Returns `true` if the payment is successful, or `false` if the payment is not. If the payment is unsuccessful, the balance is not modified.

  There are **two situations** in which a payment would fail (*note: payments can still be made on expired cards*).

- `void extendExpireDate(int months)`: extends a `CreditCard`'s expiration date by the specified number of months. If an invalid number of months is specified, this method does nothing.

  Example: a 14 month extension on a card expiring *June 2020* would yield a new expiration date of *August 2021*.

- `static void extendOnlyExpired(CreditCard[] cards, int months)`: extends the expiration dates of **only** the expired `CreditCard`s in `cards` by the specified number of months. If an invalid number of months is specified, this method does nothing.

- `static int retrieveCVV(CreditCard card, String name, int accountNumber)`: a class function which can be used to retrieve a card's CVV. If the provided account name (case sensitive) and number match the argument `CreditCard`'s respective attributes, the card's CVV is returned. Otherwise, this function returns **-1**.

- `static CreditCard mergeCards(CreditCard c1, CreditCard c2)`: creates and returns a brand new `CreditCard` object that is the result of merging the two argument `CreditCard`s objects. More specifically, the new merged `CreditCard`'s state will be as follows:

  - its balance and credit limit becomes the sum of both merged cards' balances and credit limits, respectively.
  - the name on the new account will be a compound of the names of the two merged accounts. For example, the merged account of "`Portia`" and "`Nerissa`" would have a name of "`Portia and Nerissa`".
  - a new account number and CVV are generated
  - the expiration date will be the *later* date of the two argument accounts.

  Lastly, **both argument `CreditCard`s** are "destroyed". More specifically, this means:

  - their account numbers are left unchanged
  - their names are set to "`Merged with #[...]`", with `[...]` being the other card's account number.
  - all other primitive attributes are set to `-1`.

3 Submission

Upload your completed **`CreditCard.java`** and **`CreditCardClient.java`** files to the **Lab 04** submission link on your lab Moodle or Google Classroom page. This assignment is due:

- **Tuesday, March 1st** by **10:00PM** for lab sections **A**, **B**, **C**, and **E** (which meet on Wednesday)
- **Wednesday, March 2nd** by **10:00PM** for lab section **D** (which meets on Thursday)