Concordia University
Comp 249 - Winter 2015
Object Oriented programming II
Assignment 1

Deadline:	By 11:59pm, Friday January 23, 2015
Evaluation:	3% of your final grade
Late Submission:	No late submission.
Teams:	The assignment can be done individually or in teams of 2 (from the same lecture section). Submit only one assignment per team.
Purpose:	The purpose of this assignment is to review objects, objects as attributes of other objects, arrays of objects and other concepts seen in COMP 248 as well as to generate Javadoc a new concept in COMP249.

In a women's wallet, we typically find coins and credit cards among other things. In this assignment you will write code to simulate a wallet which contain coins and credit cards only. To do so you will write a **Coins** class, a Credit**Card** class, a **Wallet** class and a driver.

Following is a description of the three classes you are required to implement and the methods that must be included. Document your three classes using Javadoc comments and generate the Javadoc files for each class.

Class Coins:

Attributes: you want to keep track of the number of nickels (5¢), dimes (10¢), quarters (25¢), loonies ($1) and toonies ($2). You should also have static constants for the values of each of these coins which you will need to use to evaluate how much money you have in a wallet.

Methods:

- Constructors:
  - Default constructor
  - Constructor with 5 integer parameters to set the number of each coin in the wallet
  - Copy constructor with one parameter of type *Coins*.
- Accessor and mutator methods for all attributes.
- *addCoins()* method with 5 integer parameters which allows you to increase the number of each coins by the indicated number.
- *coinsTotal()* method which returns a double indicating the total value of the coins in the wallet. For example, 0.55 means the total value of the coins adds up to 55¢, while 2.25 means that the total value of the coins adds up to$2 and 25¢.

- *toString()*method which will return a string clearly indicating the count of each coin in the wallet. You decide on the format.
- *equals()*method which will return *true* if the two objects of type *Coins* being compared have the same breakdown of coins and *false* otherwise.

Class CreditCard:

Attributes: you want to keep track of the credit card type (Visa, MasterCard, etc....), the name of the card holder, and the expiry month as an integer and the expiry year as an integer.

Methods:

- Constructors:
  - Default constructor
  - Constructor with 4 parameters to set the initial value of each attribute. If the month which is passed is not between 1 and 12, set it to 0.
  - Copy constructor with one parameter of type *CreditCard*.
- Accessor methods for all attributes.
- Mutator methods only for the expiry month and one for the expiry year. If the proposed month is not between 1 and 12, set it to 0.
- *toString()* which will return a string indicating the type of credit card, the name of the owner as well as the expiry date formatted as mm/yyyy. If the month number is less than 10, it must be preceded by a zero. For example a credit card that expires in January 2015 should show an expiry of 01/2015.
- *equals()* which will return *true* if the two objects of type *CreditCard* are identical in other words have exactly the same information (in which case one is an illegal copy ☹) and *false* otherwise.

Class Wallet:

Attributes: your wallet will contain coins and credit cards (typically more than one) which will be represented by an object of type *Coins* and an array of objects of type *CreditCards*.

Methods:

- Constructors:
  - Default constructor
  - Constructor with 2 parameters to set the initial value of each attribute. One attribute is of type *Coins* and the other is an array of type *CreditCard*. (***Warning***: Be sure to set the attributes of *Wallet* in such a way so as to avoid the risk of any privacy leaks). A wallet may contain no credit cards in which case the reference to the array of *CreditCard* objects is set to *null*.
  - There is **no** copy constructor.

- A method which will return *true* if the total value of the coins of two *Wallet* objects are equal, and *false* otherwise.
- A method which will return *true* if the number of each type of coins of two *Wallet* objects are equal, and *false* otherwise.
- A method which will return the total value of the coins in a wallet.
- A method which will return the number of credit cards in a wallet.
- A method which will add a new credit card to a wallet. You will need to account for a wallet not having any credit cards before the addition. (Reminder from COMP248 – how do you increase the number of elements in an array?). This method returns the number of credit cards in the wallet after the addition.
- A method which will remove a credit card from a wallet. You will need to account for a wallet that has no credit cards. (Reminder from COMP248 – how do you reduce the number of elements in an array?). This method returns true if the removal of the card was successful and false if it was not.
- A method which will update the expiry month and year of a credit card.
- A method which will add coins to a wallet. This method should have 5 parameters, one for each type of coin and returns the new total value of the money in the wallet.
- *equals()* method which will return *true* if the total value of coins and the number of credit cards coins of two *Wallet* objects are equal, and false otherwise.
- *toString()* which will return a string clearly indicating the number of each coin as well as the details of each credit card in the wallet. It is possible for a wallet to have no credit cards, in which case "No credit cards" should be indicated.
- A method which will return a string with just the breakdown of the coins.

Finally to test these three classes, you are to write a driver which behaves in the following way:

- Welcomes the user
- Creates at least 5 wallets such that
    - 2 wallets have exactly the same coin distribution and the same number of credit cards.
    - 1 wallet with the same total value of coins of another wallet but with a different configuration of coins and this wallet should have at least 3 cards
    - The last 2 wallets have the same breakdown of coins but different from the other 3 and both have no credit cards.
    - You can hardcode the wallets in your driver if you want; this way you won't need to enter all of this information from the keyboard every time you test your program.

- Presents the following menu to the user, hence possible actions:

```
What would you like to do?
  1. See the content of all wallets
  2. See the content of one wallet
  3. List wallets with same amount of money
  4. List wallets with same coins
  5. List wallets with same amount of money and same number of credit cards
  6. Add a credit card to an existing wallet
  7. Remove an existing credit card from a wallet
  8. Update the expiry date of an existing credit card
  9. Add coins to a wallet
  0. To quit

Please enter your choice and press <Enter>:
```

Brief explanation of each choice: (If the user enters an invalid choice, tell them and request a new choice).

1. Display the coins and credit cards of each wallet. Make sure all output is clearly labelled.
2. Ask the user which wallet they wish to see the content of, and display the coins and credit card(s) info for that wallet
3. Compare all wallets and display those pairs that have the same total money value along with the $ amount. If you have displayed the pair 1 and 3, do not display the pair 3 and 1 as that is repetitive.
4. Compare all wallets and display the pairs that have the same coin distribution. Similarly as in option 3, if you have displayed the pair 1 and 3, do not display the pair 3 and 1 as that is repetitive.
5. List all wallet pairs that are equal based on the definition of equal in the class *Wallet*. Avoid repetitive displays.
6. Ask the user which wallet they want to add a card to, as well as the card information, create the new card and add it to the wallet in question.
7. Ask the user which wallet they want to remove a card from, as well as the card index they want removed, and remove it from the wallet.
8. Ask the user which card from which wallet they want to update. Ask the user for the new expiry date and update the card.
9. Ask the user which wallet's coins they want to add to and the number of each coins they want to add. Then add these coins to the coins in the wallet.

When designing the driver use static methods to make the code easier to follow and to reduce repetitive code.

A sample run to illustrate how your code is to behave can be found in the .pdf file A1SampleOutput.pdf.

Submitting Assignment 1

- Naming convention for zip file: Create one zip file containing all source files and Javadoc files for your assignment using the following naming convention:
  - If the assignment is done by 1 student:
    The zip file should be called *a#_studentID*, where # is the number of the assignment and *studentID* is your student ID number.
    For example, for the first assignment, student 123456 would submit a zip file named a1_123456.zip.
  - If the assignment is done by 2 students:
    The zip file should be called *a#_studentID1_studentID2*, where *#* is the number of the assignment, and *studentID1* and *studentID2* are the student ID numbers of each student.
    For example, for the first assignment, student 123456 and 9876543 would submit a zip file named a1_123456_9876543.zip
- Submit your zip file at: https://fis.encs.concordia.ca/eas/ as **Programming Assignment** and submission **1** for **COMP249.**
- Be warned that assignments not submitted in the requested location, be it in a different folder than Programming Assignment, a different submission number than 1 or in the folder of a different course will not be graded. If the wrong files are submitted you will not have the chance of resubmitting. We were lenient in COMP 248, but now it is your responsibility to make sure your assignment is submitted in the proper location and that the requested files are present.
- Submit only ONE version of an assignment. If more than one version is submitted the first one will be graded and all others will be disregarded.

Evaluation Criteria for Assignment 1 (20 points)

Client Program: Comments - description of variables/ description of the steps in code/ Purpose of program/ Choice of variable names/ Indentation and readability of program/	3 pts
Implementation of classes based on specifications	4 pts
Completeness of Javadoc documentation for classes	4 pts
Efficiency of algorithm for driver/application – use of methods in classes, use of static method, non-repetitive code.	4 pts
Correctness of output/algorithm	3 pt
Format of output/ readability	2 pt