Concordia University COMP 248 – F2023 Assignment 4

Due Date:	By 11:59pm, December 1, 2023
Evaluation:	8% of final mark (see marking rubric at the end of handout)
Late Submission:	none accepted.
Purpose:	The purpose of this assignment is to help you learn the concepts of classes and arrays of objects in Java.
CEAB/CIPS Attributes:	Design/Problem analysis/Communication Skills.

NOTE 1: *You are NOT allowed to post any assignment and/or its solution anywhere on the World Wide Web (WWW) and/or the Internet. Intellectual Property Rights are reserved. If any case is discovered via your account or IP address; your submission will NOT be considered and will be reported immediately to the appropriate authority.*

NOTE 2:

i. Do not use/import any other library/package other than `java.util.Scanner`

ii. Do not import `java.util.Arrays` class or any data structure for any task herein.

General Guidelines When Writing Programs:

- Include the following comments at the top of your source codes.

```
// -----------------------------------------------------
// Assignment (include number)
// Written by: (include your name and student ID)
// For COMP 248 Section (your section) – Fall 2023
// -----------------------------------------------------
```

- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.
- Include comments in your program describing the main steps in your program. Focus on your comments should be on the “why” than the “how”.
- Display a welcome message.
- Display clear prompts for users when you are expecting the user to enter data from the keyboard.
- All output should be displayed with clear messages and in an easy-to-read format.
- End your program with a closing message so that the user knows that the program has terminated.

CostLessBites catering is coming to Concordia campuses to serve Concordia community (students and their family as well as employees) with healthy and budget sales throughout the year. Any member of the community can order meals using either their money or prepaid meal cards. In this assignment you will write code to simulate the **P**oint **of** **S**ale (PoS) for *CostLessBites* catering which handles orders either as on-demand *`sales`* or *`prepaid card`*. To do so you will write ***`Sales`*** class, a ***`PrePaiCard`*** class, a ***`PoS`*** class and a driver**:** ***`PoSDemo`***.

The catering restaurants offer the following:

- Five meal categories: *`junior`*, *`teen`*, *`medium`*, *`big`* and *`family`* size, for different diets and with flat rate price that includes taxes.
- Meal's menu from six diets: Carnivore, Halal, Kosher, Pescatarian, Vegetarian, Vigan.
- Yearly prepaid cards sold by type of diet.

Following is a description of the three classes you are required to implement and the methods that must be included.

Class *`Sales`*

Attributes: you want to keep track of the number of sales sold on-demand: junior ($5), teen ($10), medium ($12), big ($15) and family ($20). You should also have static constants for the values of each of these sales which you will need to use to evaluate how much fund you have in a till.

Methods:

- Constructors:
  - Default constructor
  - Constructor with 5 integer parameters to set the number of each meal category in the till.
  - Copy constructor with one parameter of type Sales.
- Accessor and mutator methods for all attributes.
- *`addSales()`* method with 5 integer parameters which allows you to increase the number of each meal category by the indicated number.
- *`SalesTotal()`* method which returns an integer indicating the total value of the sales in the PoS. For example, 85 means the total payment value sums up to $85.
- *`toString()`* method which will return a string clearly indicating the count of each meal in the till. You decide on the format.
- *`equals()`* method which will return *`true`* if the two objects of type Sales being compared have the same breakdown of meal category and *false* otherwise.

Class *PrePaiCard*

Attributes: you want to keep track of the prepaid card `type` customers used (Carnivore, Halal, Kosher, Pescatarian Vegetarian, Vigan), the ID of the card holder (student id: for students and their family members, or employee ID for Concordia employe), and the expiry `day` as an integer, and `month` as an integer.

Methods:

- Constructors:
  - Default constructor
  - Constructor with 4 parameters to set the initial value of each attribute. If the day which is passed is not between 1 and 31, set it to 0 and if the month which is passed is not between 1 and 12, set it to 0.
  - Copy constructor with one parameter of type *PrePaiCard*.
- Accessor methods for all attributes.
- Mutator methods for the due date and one for the due month. If the proposed day is not between 1 and 31, set it to 0, and if the proposed month is not between 1 and 12, set it to 0.
- `toString()` which will return a string indicating the type of the pre-paid card, ~~the name of the owner as well as,~~ the due date formatted as dd/mm. If the day number is less than 10, it must be preceded by a zero and if the month number is less than 10, it must be preceded by a zero. For example, a pre-paid card that expires on the seventh of January should show a due date of 07/01.
- `equals()` which will return *true* if the two objects of type *PrePaiCard* are identical in other words have exactly the same information and *false* otherwise.

Class *PoS*

Attributes: your `PoS` will contain ~~`Meals`~~ `Sales` and `PrePaiCards` (typically more than one) which will be represented by an object of type ~~***Meals***~~ `Sales` and an array of objects of type `PrePaiCards`.

Methods:

- Constructors:
  - Default constructor
  - Constructor with 2 parameters to set the initial value of each attribute. One attribute is of type *Meals* and the other is an array of type *PrePaiCard*. (***Warning***: Be sure to set the attributes of PoS in such a way so as to avoid the risk of any privacy leaks). A PoS may contain no pre-paid cards in which case the reference to the array of *PrePaiCard* objects is set to *null*.
  - There is **no** copy constructor.
- A method which will return *true* if the total $ value of sales of two *PoS* objects are equal, and *false* otherwise.

- A method which will return *true* if the number of each sales category of two *PoS* objects are equal, and *false* otherwise.
- A method which will return the total $ sales of a PoS. `
- A method which will return the number of *prepaid cards* in a PoS.
- A method which will add a new *PrePaiCard* to the PoS. You will need to account for a PoS not having any *PrePaiCard* before the addition. (Reminder from COMP248 – how do you increase the number of elements in an array?). This method returns the number of the pre-paid cards of a PoS after the addition.
- A method which will remove a pre-paid card from the PoS. You will need to account for a PoS that has no prepaid 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 day and month of a prepaid card.
- A method which will add meals sales to the PoS. This method should have 5 parameters, one for each meal category and returns the new total sales value payments of the PoS.
- `equals()` method which will return *true* if the total $ amount sales value and the number of pre-paid card of two *PoS* objects are equal, and false otherwise.
- `toString()` which will return a string clearly indicating the number of each meal sales category as well as the details of each pre-paid card of the PoS. It is possible for a PoS to have no pre-paid card, in which case "No pre-paid cards" should be indicated.
- A method which will return a string with just the breakdown of the sales.

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

- Welcomes the user.
- Creates at least 5 PoS such that:
  - 2 PoS have exactly the same sales categories distribution and the same number of prepaid cards.
  - 1 PoS with the same total $ amount of sales of another PoS but with a different configuration of sales categories and this PoS should have at least 3 prepaid cards.
  - The last 2 PoSs have the same breakdown of sales but different from the other 3 and both have no prepaid cards.
  - You can hardcode the PoSs 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 PoSs
   2. See the content of one PoS
   3. List PoSs with same $ amount of sales
   4. List PoS with same number of Sales categories
   5. List PoSs with same $ amount of Sales and same number of Prepaid cards
   6. Add a Prepaid card to an existing PoS
   7. Remove an existing Prepaid card from a PoS
   8. Update the expiry date of an existing Prepaid card
   9. Add sales to a PoS
   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 sales categories and prepaid cards of each PoS. Make sure all output is clearly labelled.
2. Ask the user which PoS they wish to see the content of and display the categories of the sales and the prepaid card(s) info for that PoS.
3. Compare all PoSs and display those pairs that have the same total $ amount of sales along with the $ amount. If you have displayed the pair 1 and 4, do not display the pair 4 and 1 as that is repetitive.
4. Compare all PoSs and display the pairs that have the same $ amount distribution. Similarly, as in option 3, if you have displayed the pair 1 and 4, do not display the pair 4 and 1 as that is repetitive.
5. List all PoSs pairs that are equal based on the definition of equal in the class PoS. Avoid repetitive displays.
6. Ask the user which PoS they want to add a prepaid card to, as well as the prepaid card information, create the new prepaid card and add it to the PoS in question.
7. Ask the user which PoS they want to remove a prepaid card from, as well as the prepaid card index, they want removed, and remove it from the PoS.
8. Ask the user which prepaid card from which PoS they want to update. Ask the user for the new expiry date and update the prepaid card.
9. Ask the user which PoS's sales they want to add to and the number of each category sales they want to add. Then add these sales categories to the sales in the PoS.

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- *A4Sample_Output*

Submitting Assignment 4

- Zip the source code (the .java files only please) of this assignment.
- Naming convention for zip file: Create one zip file, containing the source files for your assignment using the following naming convention:
  - ➢ 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 fourth assignment, student 123456 would submit a zip file named a4_123456.zip.
- Submit your zip file on e-Concordia or Moodle course webpage.

Evaluation Criteria for Assignment 4 (20 points)

Source Code	
Code style: Comments - description of variables/ description of the steps in code/ Purpose of program/ Choice of variable names/ Indentation and readability of the programs	3 pts
Implementation of the classes and driver based on specifications	8 pts
Efficiency of algorithm for driver and the classes– use of methods in classes, use of static method, non-repetitive code.	4 pts
Correctness of output/algorithm	3 pts
Format of output/ readability	2 pts