Concordia University
Computer Science & Software Engineering
Faculty of Engineering and Computer Science

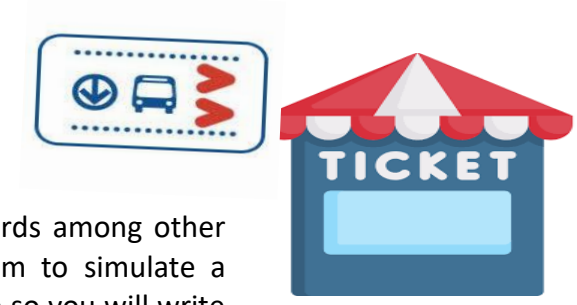
Concordia University
COMP 248 – Fall 2021
Assignment 4

Due Date:	By 11:59 PM, December 3rd , 2021
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 have you manipulate classes, objects and arrays of objects
CEAB/CIPS ATTRIBUTES:	Design/Problem analysis/Communication Skills

Please note: *you are NOT allowed to post the assignment/solution anywhere on the Internet. Intellectual Property rights are reserved. If any similar cases are found via your account or IP, your submission will NOT be considered and will be reported immediately.*

General Guidelines When Writing Programs:
Refer to assignment #1 handout.

Teams: The assignment can be done individually or in teams of 2 Max. Team members must be in the same section. Submit one assignment per team; be sure to have both team members' name in the comments at the top of the assignment.



Ticketbooth Simulator

A ticketbooth handles typically tickets and OPUS cards among other things. In this assignment, you will write a program to simulate a ticketbooth which holds tickets and OPUSCard. To do so you will write a **Tickets** class, a **OPUSCard** class, a **Ticketbooth** class and a **driver**.

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

Class *`Tickets`*:

Attributes: you want to keep track of the number of the following tickets type:

- Regular ticket (3.50$)
- Junior ticket (2.50$)
- Senior ticket (1$)
- Daily ticket (10$)
- Weekly ticket (40$)

You should also have static constants for the values of each of these tickets which you will need to use to evaluate the inventory of the tickets you have in a ticketbooth.

Methods:

- Constructors:
  - Default constructor
  - Constructor with 5 integer parameters to set the number of each ticket in the ticketbooth.
  - Copy constructor with one parameter of type *Tickets*.
- Accessor and mutator methods for non-constant attributes.
- `addTickets()` method with 5 integer parameters which allows you to increase the number of each tickets by the indicated number.
- `ticketsTotal()` method which returns a double indicating the total value of the tickets in the ticketbooth. For example, means the total value of the tickets adds up to $56, means that the total value of the tickets adds up to $40, $10, 3.50 and $2.50.
- `toString()` method which will return a string clearly indicating the count of each ticket in the ticketbooth. You decide on the format.
- `equals()` method which will return *true* if the two objects of type *Tickets* being compared have the same breakdown of tickets and *false* otherwise.

Class *`OPUSCard`*:

Attributes: you want to keep track of the OPUS card type (STM, RTL, STL, REM, 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 *OPUSCard*.
- 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 OPUS 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 an OPUS card that expires in January 2022 should show an expiry of 01/2022.
- `equals()` which will return *true* if the two objects of type *OPUSCard* are identical in other words have exactly the same information (in which case one is an illegal copy) and *false* otherwise.

Class *`Ticketbooth`*:

Attributes: your ticketbooth will contain tickets and OPUS cards (typically more than one) which will be represented by an object of type *Tickets* and an array of objects of type *OPUSCards*.

Methods:

- Constructors:
  - Default constructor
  - Constructor with 2 parameters to set the initial value of each attribute. One attribute is of type *Tickets* and the other is an array of type *OPUSCard*. (***Warning***: Be sure to set the attributes of *Ticketbooth* in such a way so as to avoid the risk of any privacy leaks). A ticketbooth may contain no OPUS cards in which case the reference to the array of *OPUSCard* objects is set to *null*.
  - There is **no** copy constructor.
- A method which will return *true* if the total value of the tickets of two *Ticketbooth* objects are equal, and *false* otherwise.
- A method which will return *true* if the number of each type of tickets of two *Ticketbooth* objects are equal, and *false* otherwise.
- A method which will return the total value of the tickets in a ticketbooth.
- A method which will return the number of OPUS cards in a ticketbooth.
- A method which will add a new OPUS card to a ticketbooth. You will need to count for a ticketbooth not having any OPUS cards before the addition. (Reminder how do you increase the number of elements in an array from arrays chapter). This method returns the number of OPUS cards in the ticketbooth after the addition.
- A method which will remove an OPUS card from a ticketbooth. You will need to account for a ticketbooth that has no OPUS cards. (Reminder how do you reduce the number of elements in an array). This method returns true if the removal of the OPUS card was successful and false if it was not.
- A method which will update the expiry month and year of an OPUS card.
- A method which will add tickets to a ticketbooth. This method should have 5 parameters, one for each type of ticket and returns the new total value of the tickets' amount in the ticketbooth.
- *equals()* method which will return *true* if the total value of tickets and the number of OPUS cards tickets of two *Ticketbooth* objects are equal, and false otherwise.

- *toString()* which will return a string clearly indicating the number of each ticket as well as the details of each OPUS card in the ticketbooth. It is possible for a ticketbooth to have no OPUS cards, in which case "No OPUS cards" should be indicated.
- A method which will return a string with just the breakdown of the tickets.

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 ticketbooths such that
  - 2 ticketbooths have exactly the same ticket distribution and the same number of OPUS cards.
  - 1 ticketbooth with the same total value of tickets of another ticketbooth but with a different configuration of tickets and this ticketbooth should have at least 3 OPUS cards
  - The last 2 ticketbooths have the same breakdown of tickets but different from the other 3 and both have no OPUS cards.
  - You can hardcode the ticketbooths 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 Ticketbooths
   2. See the content of one Ticketbooth
   3. List Ticketbooths with same amount of tickets' values
   4. List Ticketbooths with same Tickets amount
   5. List Ticketbooths with same amount of tickets values and same number of OPUS cards
   6. Add a OPUS card to an existing Ticketbooth
   7. Remove an existing OPUS card from a Ticketbooth
   8. Update the expiry date of an existing OPUS card
   9. Add Tickets to a Ticketbooth
   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 tickets and OPUS cards of each ticketbooth. Make sure all output is clearly labelled.
2. Ask the user which ticketbooth they wish to see the content of, and display the tickets and OPUS card(s) info for that ticketbooth.
3. Compare all ticketbooths and display those pairs that have the same total tickets' amount 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 ticketbooths and display the pairs that have the same ticket 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 ticketbooth pairs that are equal based on the definition of equal in the class *Ticketbooth*. Avoid repetitive displays.
6. Ask the user which ticketbooth they want to add an OPUS card to, as well as the OPUS card information, create the new OPUS card and add it to the ticketbooth in question.
7. Ask the user which ticketbooth they want to remove an OPUS card from, as well as the card index they want removed, and remove it from the ticketbooth.
8. Ask the user which card from which ticketbooth they want to update. Ask the user for the new expiry date and update the OPUS card.
9. Ask the user which ticketbooth's tickets they want to add to and the number of each tickets they want to add. Then add these tickets to the tickets in the ticketbooth.

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 file Sample RunA#4.pdf.

Submitting Assignment 4

What to submit:

Zip the source codes (the .java files only please, **not** the entire project) of this assignment as a .ZIP file (**NOT** .RAR) using the following naming convention: *a#_studentID*, where # is the number of the assignment and *studentID* is your student ID number. For example, for this third assignment, student 123456 would submit a zip file named a4_123456.zip. If the assignment is done by two students it must be named a4_IDStudent1_IDStudent2.zip.

How to submit:

Submit from Moodle course page. If you are in an eConcordia course, please check your eConcordia webpage for instructions on how to submit your assignment.

Evaluation Criteria for Assignment 4 (20 points)

Source Code	
Comments for all 2 questions (1.5 pts.)	
Description of the program (authors, date, purpose)	0.5 pts.
Description of variables and constants	0.5 pts.
Description of the algorithm	0.5 pts.
Programming Style (1.5 pts.)	
Use of significant names for identifiers	0.5 pts.
Indentation and readability	0.5 pts.
Welcome Banner or message/Closing message	0.5 pts.
Implementation of Tickets class (3 pts.)	
Attributes	0.5 pts.
Methods	2.5 pts.
Implementation of OPUSCard class (3 pts.)	pts
Attributes	0.5 pts.
Methods	2.5 pt.
Implementation of Ticketbooth class (4 pts.)	
Attributes	1.0 pt.
Methods	3 pts.
Implementation of Driver class (7 pts.)	pts.
Creation of all objects	2.0 pts.
Menu Algorithm	5.0 pt.
TOTAL	**20 pts.**