Concordia University
Comp 249 - Winter 2018
Object-Oriented Programming II
Assignment # 1 – Due 11:59 PM – Wednesday, January 31, 2018

Purpose: The purpose of this assignment is to help you review some of the main topics covered in the previous course, including classes, loops, arrays, arrays of objects, static attributes and static methods.

Part I

For this part, you are required to design and implement the **Book** class according to the following specifications:

- A **book** object has four attributes, a book *title* (String), an author *name* (String), an *ISBN* number (long), and a *price* tag (double).
  - Upon the creation of a book object, the object must immediately be initialized with valid values; that is title, author name, ISBN number and price value. (Hint: use constructors.).
  - The design should allow enough flexibility so that the value of any of these attributes can be modified later on. For example, it should be possible to create a book object with a given price then change its price later on. The design should also allow the user to obtain the value of any of the attributes. (Hint: use accessors & mutators.)
  - The design should allow all information of an object to be displayed at once through the utilization of **`System.out.print()`** method. (Hint: use **`toString()`** method).
  - It is required to know how many Book objects have been created. For that, you need to add a method, called **`findNumberOfCreatedBooks`**`()`, to the class. This method must return the number of created Book objects prior to the time this method is called. The method would simply return 0 if no books has been created by the time the method is called. (Hint: use Static – You are allowed to add other attributes to the class.) .
  - It is required to compare two Book objects for equality. Two Book objects are considered equal if they have the same ISBN and price. (Hint: use **`equals()`** method).
  - It is required to display any Book object (all info of that object) using **`System.out.println()`** method. (Hint: use **`toSting()`** method).

Part II

You are hired by a bookstore to write a software application that helps the store staff (users) in keeping track of the books at the store.

Write a driver program that will contain the **`main()`**`method` and it will perform the following:
(Note: You can have the main function in a separate driver file, or in the same file if you prefer)

- Display a welcome message.
- Prompt the user for the maximum number of books (maxBooks) his/her bookstore can contain. Create an empty array, called `inventory`, that will have the potential of keeping track of the created `Book` objects.

- Display a main menu (figure 1) with the following choices and keep prompting the user until they enter a number between 1 and 5 inclusive:

```
What do you want to do?
    1. Enter new books (password required)
    2. Change information of a book (password required)
    3. Display all books by a specific author
    4. Display all books under a certain a price.
    5. Quit
Please enter your choice >
```

figure 1. Main menu

- When **option 1** is entered:
  - Prompt the user for his/her password. (make sure you have a constant variable containing the password "`password`" – do not use any other password as it will be easier for the marker to check your assignments). The user has a maximum of 3 attempts to enter the correct password. After the 3rd wrong attempt entry, the main menu in figure 1 is re-displayed again. Additionally after this process is repeated 4 times (i.e. total failed attempts is 12), the program must displays the following messages: "`Program detected suspicous activities and will terminate immediately!`", then the program must exits.

  - If the correct password is entered, ask the user how many books he/she wants to enter. Check to make sure that there is enough space in the bookstore (array of `Book`) to add that many books. If so, add them; otherwise inform the user that he/she can only add the number of remaining places in the array. (How the book information will be input/entered by the user, is up to you).

- When **option 2** is entered :
  - Prompt the user for his/her password. (make sure you have a constant containing the password "`password`" as a constant – do not use another password). Again the user has 3 attempts to enter the correct password. However, after the 3rd wrong attempt entry, the main menu in figure 1 is simply re-displayed again (notice the different behaviour in that case from the previous one above).

  - Once the correct password was entered, the user is asked which book number he/she wishes to update. The `book` number is the index in the array `inventory`. If there is no `Book` object at the specified index location, display a message asking the user if he/she wishes to re-enter another book, or quit this operation and go back to the main menu. If the entered index has a valid book, display the current information of that book in the following format:

```
Book:  # x              ( index of the book in the inventory array)
Author: name of author
Title: title of book
ISBN: ISBN #
Price: $ price
```

❖ Then ask the user which attribute they wish to change by displaying the following menu.

```
What information would you like to change?
    1. author
    2. title
    3. ISBN
    4. price
    5. Quit
Enter your choice >
```

figure 2. Update menu

- Once the user has entered a correct choice, make the changes to the attribute then display again all of the attributes on the screen to show that the attribute has been changed. Keep prompting the user for additional changes until the user enters 5. Each time the user is prompted for a choice make sure that a number from 1 to 5 is entered, otherwise keep prompting until a valid number is entered. (ensure that the user is able to change any of the choice 1 to 4 on figure 2) .

➢ When **option 3** (in the main menu shown in figure. 1) is entered, prompt the user to enter an author name. You then need to display the information of all books by that requested author. (Hint: You may use a static method, for instance called **`findBooksBy`** , which accepts a string for an author name then performs the needed search).

➢ When **option 4** (in the main menu shown in figure. 1) is entered, prompt the user to enter a value (representing a price). You then need to display all books that have a vlaue smaller than that entered value. (Hint: You may use a static method, for instance called **`findCheaperThan`**, which accepts a double value, for a price, then performs the needed search).

➢ When **option 5** (in the main menu shown in figure. 1) is entered, display a closing message and end the driver.

General Guidelines When Writing Programs:

- Include the following comments at the top of your source code (each .java file)

```
// -------------------------------------------------------
// Assignment# (include number)
// Question: (include question/part number, if applicable)
// Written by: (include student name and id if assignment done
//by one student or both students if assignment done by two
//students
// -------------------------------------------------------
```

- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations might get a bit longer.
- 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.

JavaDoc Documentation:

Documentation for your program must be written in **javaDoc**.
In addition, the following information must appear at the top of each file:

```
Name(s) and ID(s)     (include full names and IDs)
COMP249
Assignment # (include the assignment number)
Due Date     (include the due date for this assignment)
```

Assignment#1 Submission

- For this assignment, you are allowed to work individually, or in a group of **2 students maximum.** You and your teammate must however be in the same section of the course.
- If working in a group, **only one** of the two members submit/upload the assignment.
- Only electronic submissions will be accepted. Zip together the source code files.
- Students will have to submit their assignment (one copy per group) using the Moodle/EAS system (please check for your section submission). Assignments must be submitted in the right DropBox/folder of the assignments. **Assignments uploaded to an incorrect DropBox/folder will not be marked and result in a zero mark. No resubmissions will be allowed. Your instructor will indicate whether you should upload your assignment to EAS (**https://fis.encs.concordia.ca/eas/**) or to Moodle.**
- **Naming convention for zip file:** Create one zip file, containing all source files and produced documentations for your assignment using the following naming convention:
  - The zip file should be called *a#_StudentName_StudentID*, where # is the number of the assignment and *StudentName/StudentID* is your name and ID number respectively. Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. For example, for the first assignment, student 12345678 would submit a zip file named like: *a1_Mike-Simon_123456.zip.* If working in a group, the name should look like: *a1_Mike-Simon_12345678-AND-Linda-Jackson_98765432.zip.*
- 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.

Assignment 1 Evaluation Criteria (10 points)

Documentations	**1 pts**
javaDoc documentations	1 pt
Part I (Class **`Book`**)	**3 pts**
Default & copy constructors	1 pt
Accessor/mutator method for static attribute	1 pt
`equals, toString` and static attributes/methods	2 pts
Part II (Driver & static methods & other methods)	**6 pts**
Handling of password	1 pt
Handling of option 1	1 pt
Handling of option 2	1 pt
Handling of option 3	1 pt
Handling of option 4	1 pt
Handling of option 5	1 pt