COMP249/4 – Winter 2016

ASSIGNMENT #2

Due: Wednesday March 2, 2016

Purpose: The purpose of this assignment is to allow you practice File I/O, Exception Handling, Searching, as well as other previous topics.

General Guidelines When Writing Programs:

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

```
// -------------------------------------------------------------------------
// Assignment (include number)
// Questions: (include part number)
// Written by: (include you name and student id)
// For COMP249 Section: (include your section letter(s))
// -------------------------------------------------------------------------
```

- 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.
- Display a welcome message which includes your name(s).
- 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.
- CEAB/CIPS Attributes: Design - Problem analysis.

Part 1

The Publication class has the following attributes: a *publication_code* (long type), a *publication_name* (String type), a *publication_year* (int type), a *publication_authorname* (String type), a publication_*cost* (double type), and a publication_*nbpages* (int type). It is assumed that both the publication name and the author(s) name are recorded as a continuous string with "_" to separate the different words if any (i.e. a publication name can be Absolute_Java, and author name can be Walter_Savitch).

The file *PublicationData_Input.txt*, which one of its versions is provided with this assignment, has the information of various publications that a publication store carries. However, this file is always created by the owner of the publication store, who is not so careful, so errors in the publication codes are expected to exist in this file. In specific, due to a cut-and-paste practice by the owner, some publication codes of some publications are usually re-recorded later in the file as the publication code of other following publications in the file. Consequently, a publication code can either appears once in the file, which is the correct case, or appears multiple times, in which case the second, and following, appearances are in error.

> You must notice that the file *PublicationData_Input.txt* changes regularly, and it may have many records, or no records at all, depending on the current inventory of the store. The file provided with this assignment is only one version of the file, and must not be considered as the general case when writing your publication code.

For that part, you are required to:

1. Write the implementation of the publication class according to the above given specification.
2. Write the implementation of an exception class called CopyCodeException, which extends the Exception class. Should a copy publication code is detected at any point, an object from this class will be thrown. More details will follow below
3. Write the implementation of a public class, called PublicationListingProcess1, which will utilize the publication class and the *PublicationData_Input.txt*, as explained below. The class has a static array of publications, called *PublicationArray[]*, an enumerated type called PublicationTypes containing the following values : *PUBLICATIONCODE*, *PUBLICATIONNAME*, *PUBLICATIONYEAR*, *PUBLICATIONAUTHORNAME*, PUBLICATION*COST*, PUBLICATION*NBPAGES*, and few methods. Beside the main method, the class must have the following methods :
    a. A method called *correctListOfItems()*, which accepts two parameters, an input file stream name and an output file stream name. The first parameter is the stream related to the *PublicationData_Input.txt*. The second parameter is related to an output file name, which will be entered by the user prior to calling this method (at the main method). This output file will eventually store a correct version of the items list. More information on that will follow below;

b. A method called *printFileItems()*, which accepts an input file stream name, then prints the data of this file to the standard output (the screen).

c. You can add any other methods to that class if you wish to.

Here are the details of how your program must behave:

- In the main method, your program must prompt the user to enter the name of the output file that will be created to hold the modified/correct inventory. This output file is theoretically a copy of the *PublicationData_Input.txt*, but with all the copy publication codes corrected.
- If the entered name by the user matches the name of an existing file, then the program must reject that name indicating to the user that a file with that name already exists, display the size of this existing file (in bytes) to the user, then prompt the user to enter a new name. This process would repeat indefinitely, until the user finally enters a name for a non-existing file. See *Figure 1* for illustration.
- Once the user finally enters a correct name for the output file, the program will attempt to establish an input and output streams for the *PublicationData_Input.txt* and that output file accordingly. This process may surely throw specific exceptions, and your program must handle all these exceptions properly.
- The *correctListOfItems()* method (see details below) will utilize the *PublicationArray[]* array as follows: All publication objects recorded in the *PublicationData_Input.txt* file must first be copied into that array. All detections and corrections of publication codes will be conducted on that array. Once the array has finally all correct information, the objects will be recorded to the output file. However since it always unknown at the time the program starts how many records are in the *Initial_Publication_Info.txt* file, you must first find this information. You may, and should, add a private helping method to the PublicationListingProcess1 class to do so. Once the number of records is known, the array must be set to that size.
- If the number of records in the *PublicationData_Input.txt* was detected to be zero or one; the case when the file is empty or has only one record, then the program must display a message indicating that, performs any needed operations (such as closing files), then exits since there is nothing to be fixed.
- If the *PublicationData_Input.txt* has more than one record, then finally the *correctListOfItems()* method will be called to create a the new output file with the correct information. The exact details of this method are as follows:
    - The method will accept two stream names for the input and output files,
    - The method must read each publication record from the input file and creates a publication object in the array *PublicationArray[]* based on that record,

- Once the entire input file is read into the array (notice that the array has real publication objects, and not just information), the method starts to trace that array from start to end looking for any publication code copies,
- If a publication code copy is detected, then the method displays a message to the user indicating that, and prompt the user to enter a new publication code,
- You should notice that this new number must not be a copy of any other existing record, and your program must guarantee that. Should the user enters a publication code that is still a copy, the program must throw a CopyCodeException, which must be catched to display a message indicating that, then user must be prompted again to enter a new publication code. Further bad entries will result in the same action; that is throwing of the CopyCodeException object, catching it to display the message, and the user is prompted again. Effectively, this process will repeat indefinitely until the user enters a correct publication code. Again, you may, and should, create a private helping method to find if a copy exists in the array. See Figure 2 for illustration.
- Finally, once all the copy publication codes are removed, the new information of the objects in the *PublicationArray[]* must be copied to the output file. This output file has now the correct information.

- Upon the return of the *correctListOfItems()* call in the main() method, the program must use the *printFileItems()* method to display the information of both the *PublicationData_Input.txt* file, as well as the created output file. See Figure 3 for illustration. Hint: You must carefully keep track of the opening and closing of these files.

Part 2

For that part, you will still use the same Publication class from Part1 (some little modifications may be needed, and you should find that out). Other publication code segments from part1 can still be used when appropriate.

1. For that part you need to implement a public class called PublicationListingProcess2, which will utilize the publication class and file called *PublicationData_Ouput.txt*. That file includes information about publications list of items in a Publication store. The records in this file are sorted by publication code, and the information in this file is assumed to always be correct.
2. The class has a static array of publications, called *PublicationArray[]*, an enumerated type called PublicationTypes containing the following values : *PUBLICATIONCODE, PUBLICATIONNAME, PUBLICATIONYEAR, PUBLICATIONAUTHORNAME,* PUBLICATION*COST,* PUBLICATION*NBPAGES*, and few methods. Beside the main() method, the class must have the following methods :
    a. A method called *insertRowsToFile(),* which accept one parameter, an output file stream name; further details of this method are given below.
    b. A method called *printFileItems(),* which accepts an input file stream name, then displays the contents of this file to the standard output (the screen). This method however must use the BufferedReader class to read the file.
    c. A method called *binaryPublicationSearch(),* which accepts four parameters, any array of publications, a start index, an end index, and a publication code. The method then uses binary search to search the array segment (from start index to end index) for that publication code. The method must also keep track and display how many iterations it needed to perform the search (that is *O( )* complexity).
    d. A method called *sequentialPublicationSearch(),* which accepts four parameters, any array of publications, a start index, an end index, and a publication code. The method would then search the array, using sequential search, for that publication code. You must analysis the performance aspect of your solution. Hence, the method must also display how many iterations it needed to perform the search (that is *O( )*).
    e. You can add any other methods to that class if you wish to.

Here are the details of how your program must behave:

- In the *main* method, your program must open the *PublicationData_Ouput.txt* to append few records to it.
- Call the *insertRowsToFile(),* method to insert few records to the file. The information of each of the new records must be entered by the user. The user can add as many records as he/she wishes (you must surely allow the user to have a

stopping condition). To simplify your task, you can assume that the user will always keep the file sorted; that is, no new record will have a publication code that is smaller than any of the previous records in the file.

- Call the *printFileItems()* method to show the contents of the file after modifications.
- Now, the array *PublicationArray[]* is going to be used in a similar fashion to part1. So, you need to find out what is the current number of records in the *PublicationData_Ouput.txt* file, and set the size of the array to that number. You may, and should, have a private helping method to do so.
- After setting the array size properly, the program must read the contents of the file to the array objects. You can use a separate method to do so.
- Once the array has the objects information from the file, prompt the user to enter and publication code then use binaryPublicationSearch() to search for that publication code.
- Repeat the same search using sequentialPublicationSearch().

  ➔ As an important exercise, you should run your publication code for different searches (attempt to find best and worst cases) and compare the *O()* complexity of the two searches.

- Finally, store the all the objects on the *PublicationArray[]* into a binary file called *Publications.dat*. You can have a separate method to so. If writing to the binary file requires any specific modifications to your publication code, you must apply these modifications.

  ⇨ As always, you must handle all exceptions as needed and you must keep track of the opening and closing of your files.

Submitting Assignment #2

- Zip the source codes for part 1 and part 2 along with all files used/created in one zip file. (Please use ARCHIVE tool).
- Naming convention for zip file:

  The zip files should be called *a2_part1-2_studentID*. For example, for the file, student 123456 would submit a zip file named *a2_part1-2_123456.zip*
- For the professor Acemian's section, submit your zip file via Moodle.
- For Professors Clark and Taleb sections respectively, submit your zip file at: https://fis.encs.concordia.ca/eas/ as programming assignment and submission #2. Assignments submitted to the wrong directory would be discarded and no replacement submission will be allowed.
- 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 or Assignment #2 (5 points)

Part 1 (3 points)	
Publication & CopyCodeException classes	0.5 pt
Correct implementation to publication listing process	2 pts
printFileItems() method	0.5 pt
Part 2 (2 points)	
insertRowsToFile() method	0.5 pt
Search Methods & *Big O()*	0.5 pt
printFileItems() method	0.5 pt
Writing to the binary file	0.5 pt