**Concordia University**
**Comp 249 – Winter 2017**
**Object-Oriented Programming II**
**Assignment 3 – Due 11:59 PM - Friday March 10, 2017**

---

**<u>Purpose:</u>** The purpose of this assignment is to allow you practice Exception Handling, and File I/O, as well as other previous object oriented concepts.

---

This assignment contains two parts. You need to complete part I before doing part II.

## <u>Part I</u>

The **Book** class has the following attributes: an *ISBN* (long type), a *title* (String type), an *issue year* (int type), an *author(s) name* (String type), a *price* (double type) and a *number of pages* (int type). It is assumed that both the title and the author(s) name are recorded as a continuous string with "_" to separate the different words if any (i.e. a title can be Java_Applications_for_Engineers, and author name can be Mike_Westman_and_Jack_Bartlett).

The file ***Initial_Book_Info.txt***, which one of its versions is provided with this assignment, has the information of various books that a book store carries. However, this file is always created by the owner of the book store, who is not so careful, so errors in the ISBN numbers are expected to exist in this file. Specificaly, due to a cut-and-paste practice by the owner, some ISBN numbers of some books are usually re-recorded later in the file as the ISBN number of other following books in the file. Consequently, an ISBN 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 *Initial_Book_Info.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 of possible versions of the file, and must not be considered as the general case when writing your code.

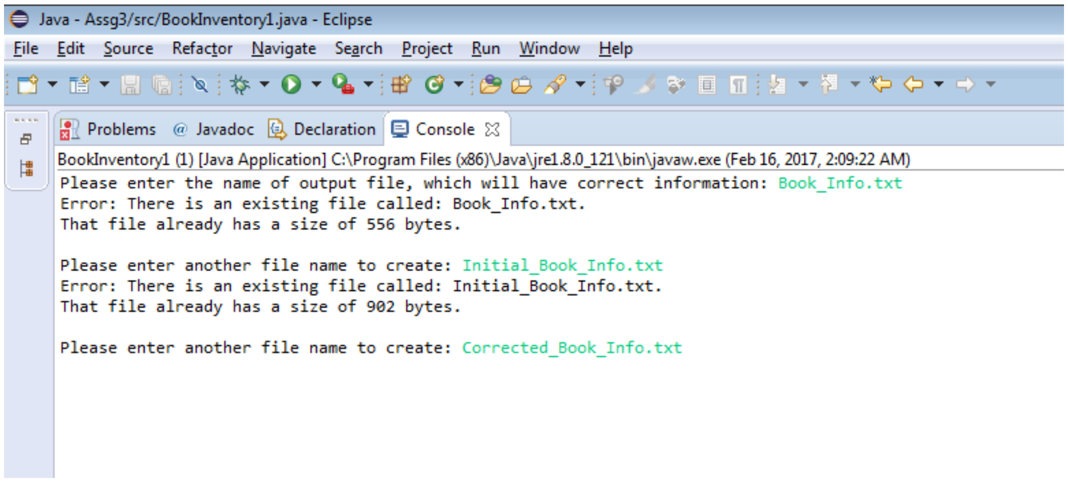
<u>For this part, you are required to:</u>

1. Write the implementation of the **Book** class according to the above given specification.
2. Write the implementation of an exception class called **DuplicateISBNException**, which extends the **Exception** class. Should a duplicate ISBN number 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 **BookInventory1**, which will utilize the **Book** class and the *Initial_Book_Info.txt*, as explained below. The class has a static array of books, called *bkArr[]*, and few methods. Beside the main() method, the class must have the following methods:
   - A method called ***fixInventory()***, which accepts two parameters, an input file stream name and an output file stream name. The first parameter is the stream related to the *Initial_Book_Info.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 inventory. More information on this will follow below.
   - A method called ***displayFileContents()***, which accepts an input file stream name, then displays the contents of this file to the standard output (the screen).
   - You can add any other methods to this 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 *Initial_Book_Info.txt*, but with all the duplicate ISBN numbers 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 Figure1 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 *Initial_Book_Info.txt* and that output file accordingly. This process may surely throw specific exceptions, and your program must handle all these exceptions properly.
- The *fixInventory()* method (see details below) will utilize the *BkArr[]* array as follows: All book objects recorded in the *Initial_Book_Info.txt* file must first be copied into that array. All detections and corrections of ISBN numbers 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_Book_Info.txt* file, you must first find this information. You may, and should, add a private helping method to the **BookInventory1** class to do so. Once the number of records is know, the array must be set to that size.
- If the number of records in the *Initial_Book_Info.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 *Initial_Book_Info.txt* has more than one record, then finally the *fixInventory()* 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 book record from the input file and creates a book object in the array *bkArr[]* based on that record,
  - Once the entire input file is read into the array (notice that the array has real book objects, and not just information), the method starts to trace that array from start to end looking for any ISBN duplicates,
  - If an ISBN duplicate is detected, then the method displays a message to the user indicating that, and prompt the user to enter a new ISBN number,
  - You should notice that this new number must not be a duplicate of any other existing record, and your program must guarantee that. Should the user enters an ISBN number that is still a duplicate, the program must throw a **DuplicateISBNException**, which must be catched to display a message indicating that, then user must be prompted again to enter a new ISBN. Further bad entries will result in the same action; that is throwing of the **DuplicateISBNException** 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 ISBN number. Again, you may, and should, create a private helping method to find if a duplicate exists in the array. See Figure 2 for illustration.
  - Finally, once all the duplicate ISBN numbers are removed, the new information of the objects in the *bkArr[]* must be copied to the output file. This output file has now the correct information.
- Upon the return of the *fixInventory()* call in the main() method, the program must use the *displayFileContents()* method to display the information of both the *Initial_Book_Info.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.

Below are sample snapshots of the program behavior for further illustrations:

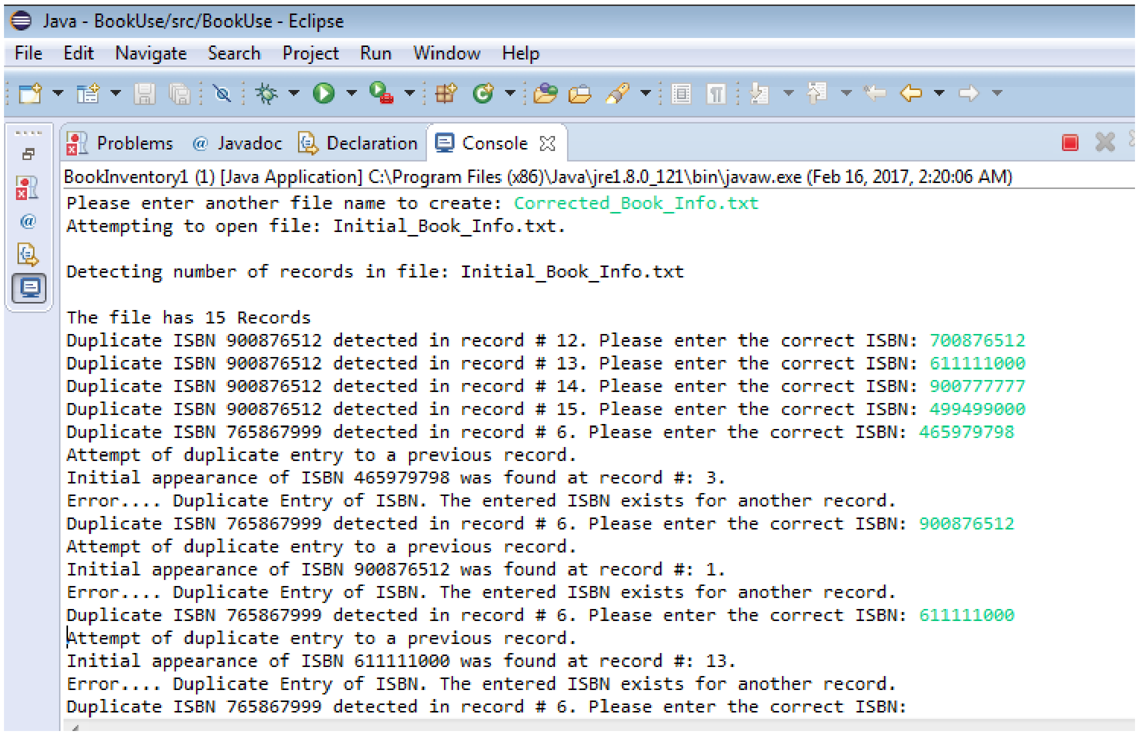

Java - Assg3/src/BookInventory1.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems @ Javadoc Declaration Console

```
BookInventory1 (1) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_121\bin\javaw.exe (Feb 16, 2017, 2:09:22 AM)
Please enter the name of output file, which will have correct information: Book_Info.txt
Error: There is an existing file called: Book_Info.txt.
That file already has a size of 556 bytes.

Please enter another file name to create: Initial_Book_Info.txt
Error: There is an existing file called: Initial_Book_Info.txt.
That file already has a size of 902 bytes.

Please enter another file name to create: Corrected_Book_Info.txt
```

Figure 1: Detecting Existing Files

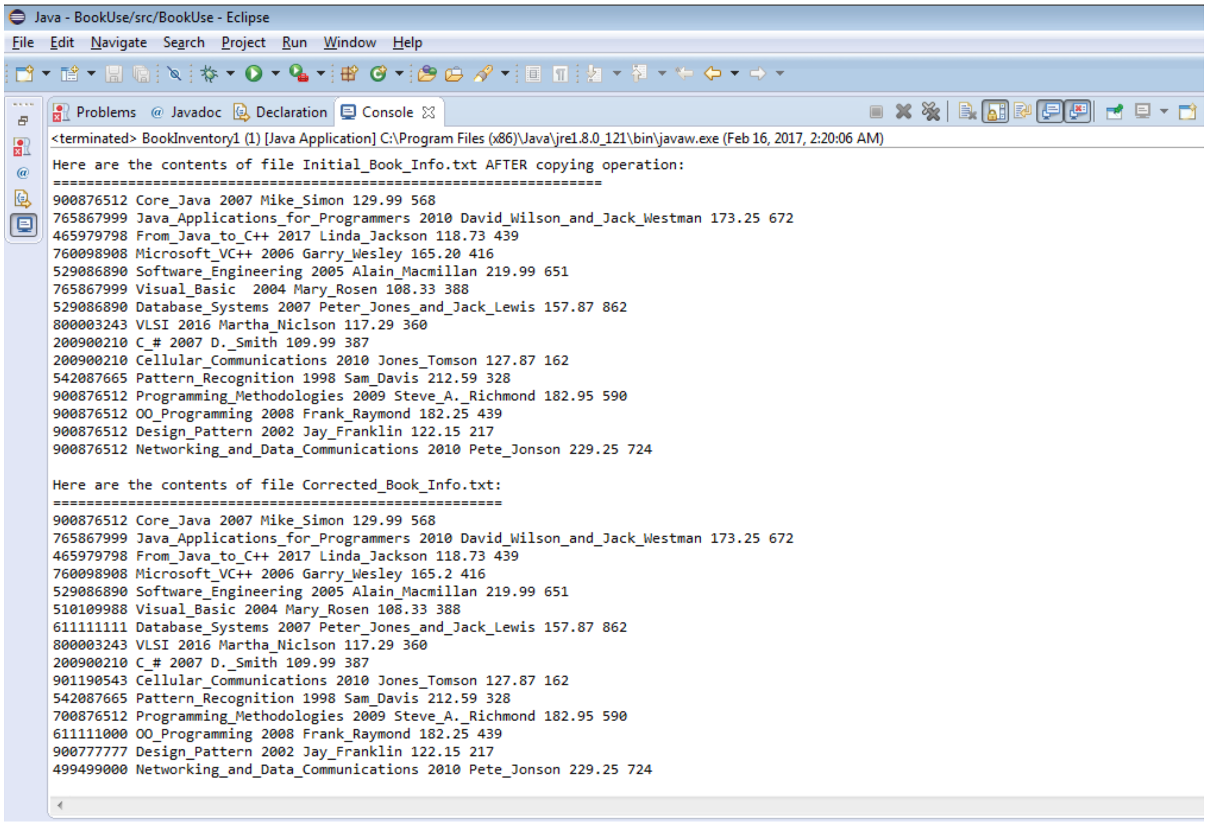

Java - BookUse/src/BookUse - Eclipse
File Edit Navigate Search Project Run Window Help
Problems @ Javadoc Declaration Console

```
BookInventory1 (1) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_121\bin\javaw.exe (Feb 16, 2017, 2:20:06 AM)
Please enter another file name to create: Corrected_Book_Info.txt
Attempting to open file: Initial_Book_Info.txt.

Detecting number of records in file: Initial_Book_Info.txt

The file has 15 Records
Duplicate ISBN 900876512 detected in record # 12. Please enter the correct ISBN: 700876512
Duplicate ISBN 900876512 detected in record # 13. Please enter the correct ISBN: 611111000
Duplicate ISBN 900876512 detected in record # 14. Please enter the correct ISBN: 900777777
Duplicate ISBN 900876512 detected in record # 15. Please enter the correct ISBN: 499499000
Duplicate ISBN 765867999 detected in record # 6. Please enter the correct ISBN: 465979798
Attempt of duplicate entry to a previous record.
Initial appearance of ISBN 465979798 was found at record #: 3.
Error.... Duplicate Entry of ISBN. The entered ISBN exists for another record.
Duplicate ISBN 765867999 detected in record # 6. Please enter the correct ISBN: 900876512
Attempt of duplicate entry to a previous record.
Initial appearance of ISBN 900876512 was found at record #: 1.
Error.... Duplicate Entry of ISBN. The entered ISBN exists for another record.
Duplicate ISBN 765867999 detected in record # 6. Please enter the correct ISBN: 611111000
Attempt of duplicate entry to a previous record.
Initial appearance of ISBN 611111000 was found at record #: 13.
Error.... Duplicate Entry of ISBN. The entered ISBN exists for another record.
Duplicate ISBN 765867999 detected in record # 6. Please enter the correct ISBN:
```

Figure 2: Detecting Existing ISBN


Java - BookUse/src/BookUse - Eclipse
File Edit Navigate Search Project Run Window Help
Problems @ Javadoc Declaration Console

```
<terminated> BookInventory1 (1) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_121\bin\javaw.exe (Feb 16, 2017, 2:20:06 AM)
Here are the contents of file Initial_Book_Info.txt AFTER copying operation:
==================================================================
900876512 Core_Java 2007 Mike_Simon 129.99 568
765867999 Java_Applications_for_Programmers 2010 David_Wilson_and_Jack_Westman 173.25 672
465979798 From_Java_to_C++ 2017 Linda_Jackson 118.73 439
760098908 Microsoft_VC++ 2006 Garry_Wesley 165.20 416
529086890 Software_Engineering 2005 Alain_Macmillan 219.99 651
765867999 Visual_Basic  2004 Mary_Rosen 108.33 388
529086890 Database_Systems 2007 Peter_Jones_and_Jack_Lewis 157.87 862
800003243 VLSI 2016 Martha_Niclson 117.29 360
200900210 C_# 2007 D._Smith 109.99 387
200900210 Cellular_Communications 2010 Jones_Tomson 127.87 162
542087665 Pattern_Recognition 1998 Sam_Davis 212.59 328
900876512 Programming_Methodologies 2009 Steve_A._Richmond 182.95 590
900876512 OO_Programming 2008 Frank_Raymond 182.25 439
900876512 Design_Pattern 2002 Jay_Franklin 122.15 217
900876512 Networking_and_Data_Communications 2010 Pete_Jonson 229.25 724

Here are the contents of file Corrected_Book_Info.txt:
======================================================
900876512 Core_Java 2007 Mike_Simon 129.99 568
765867999 Java_Applications_for_Programmers 2010 David_Wilson_and_Jack_Westman 173.25 672
465979798 From_Java_to_C++ 2017 Linda_Jackson 118.73 439
760098908 Microsoft_VC++ 2006 Garry_Wesley 165.2 416
529086890 Software_Engineering 2005 Alain_Macmillan 219.99 651
510109988 Visual_Basic 2004 Mary_Rosen 108.33 388
611111111 Database_Systems 2007 Peter_Jones_and_Jack_Lewis 157.87 862
800003243 VLSI 2016 Martha_Niclson 117.29 360
200900210 C_# 2007 D._Smith 109.99 387
901190543 Cellular_Communications 2010 Jones_Tomson 127.87 162
542087665 Pattern_Recognition 1998 Sam_Davis 212.59 328
700876512 Programming_Methodologies 2009 Steve_A._Richmond 182.95 590
611111000 OO_Programming 2008 Frank_Raymond 182.25 439
900777777 Design_Pattern 2002 Jay_Franklin 122.15 217
499499000 Networking_and_Data_Communications 2010 Pete_Jonson 229.25 724
```

Figure 3: Displaying File Contents

## Part II

For this second part, you will still use the same **Book** class from Part I (some little modifications may be needed, and you should find that out). Other code segments from Part I can still be used when appropriate.

Implement a public class called **BookInventory2**, which will utilize the **Book** class and the file called *Sorted_Book_Info.txt*. This file includes information about books inventory in a book store. The records in this file are sorted by ISBN, and the information in this file is assumed to always be correct.

The class has a static array of books, called *bkArr[]*, and few methods. Beside the main() method, the class must have the following methods:

- A method called ***addRecords()***, which accept one parameter: an output file stream name; further details of this method are given below.
- A method called ***displayFileContents()***, 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.
- A method called ***binaryBookSearch()***, which accepts four parameters: any array of books, a start index, an end index, and an ISBN number. The method then uses *binary search* to search the array segment (from start index to end index) for that ISBN. The method must also keep track and display how many iterations it needed to perform the search.
- A method called ***sequentialBookSearch()***, which accepts four parameters: any array of books, a start index, an end index, and an ISBN number. The method would then search the array, using *sequential search*, for that ISBN. You must analysis the performance aspect of your solution. Hence, the method must also display how many iterations it needed to perform the search.
- 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 *Sorted_Book_Info.txt* file to append few records to it.
- Call the *addRecords()* method to add 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 limiting the numbers of records that can be added). To simplify your task, you can assume that the user will always keep the file sorted; that is, no new record will have an ISBN# that is smaller than any of the previous records in the file., in a nother way a reference of the last record is kept to accept or reject the record you want to add.
- Call the *displayFileContents()* method to show the contents of the file after modifications.
- Now, the array *bkArr[]* is going to be used in a similar fashion to Part I. So, you need to find out what is the current number of records in the *Sorted_Book_Info.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 ISBN number then use *binaryBookSearch()* to search for that ISBN.
- Repeat the same search using *sequentialBookSearch()*.
  - ➢As an important exercise, you should run your code for different searches (i.e. attempt to find best and worst cases) of the two searches.
- Finally, store all the objects from the *bkArr[]* into a binary file called ***Books.dat***. You can have a separate method to do so. If writing to the binary file requires any specific modifications to your 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.

**General Guidelines When Writing Programs**

- Include the following comments at the top of each class you are writing.

```
// -----------------------------------------------------
// Assignment #3
// Part: (include Part Number)
// Written by: (include your name(s) and student ID(s))
// -----------------------------------------------------
```

- When commenting your code provide on the top a general and clear explanation of what the piece of code is doing; and within that piece of code if there is any method or any loop specify briefly what is doing. Include comments as needed.
- Display clear prompts for the user whenever you are expecting the user to enter data from the keyboard.
- All outputs 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.

## Submission & Demo

When you have finished the program, you must submit the assignment online to the EAS system.

1) Create **one** zip file, containing the two parts: I and II separately, where each part contains the files (.java and .html and test cases).

Please name your file following this convention:
If the work is done by 1 student: Your file should be called *a#_studentID*, where # is the number of the assignment *studentID* is your student ID number.
If the work is done by 2 students, ONLY ONE of the members can upload the file (do not upload twice): The zip file should be called *a#_studentID1_studentID2*, where # is the number of the assignment *studentID1* and *studentID2* are the student ID numbers of each student.

2) Assignments must be submitted in the right folder of the assignments. Upload your zip file at the URL: https://fis.encs.concordia.ca/eas/ as *Programming Assignment* ***3.*** **Assignments uploaded to an incorrect folder will not be marked and result in a zero mark. No resubmissions will be allowed.**

**3) Finally, notice that a demo is required for the assignment. Failure to do your demo, or missing you reserved demo time, will result in a zero mark for the assignment regardless of your submission.** ***There will be no substitution for a missed demo time.*** **Please see course outline for further details.**

## Evaluation Criteria for Assignment 3 (10 points)

| Part I (5 points) | |
|---|---|
| Book & DuplicateISBNException classes | 0.5 pt |
| Correct implementation to fix the inventory | 4 pt |
| *displayFileContents()* method | 0.5 pt |
| Part II (5 points) | |
| *addRecords()* method | 1 pt |
| Search Methods [Binary and sequential] | 2 pt |
| *displayFileContents()* method | 1 pt |
| Writing to the binary file | 1 pt |