

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)
// Question: (include question number)
// Written by: (include your name and student id)
// For Comp 249 Section: (Substitute 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.

Part 1

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. In specific, 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 version of the file, and must not be considered as the general case when writing your code.

For that part, you are required to:

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. 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 that will follow below.
- b. A method called ***displayFileContents()***, which accepts an input file stream name, then displays the contents 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 *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 known, 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 caught 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:

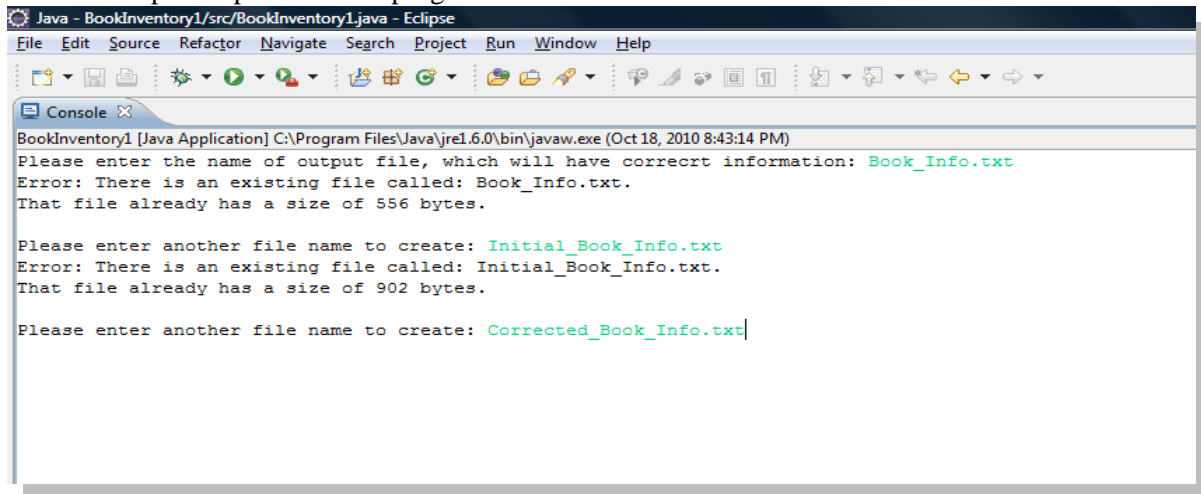


Figure 1: Detecting Existing Files

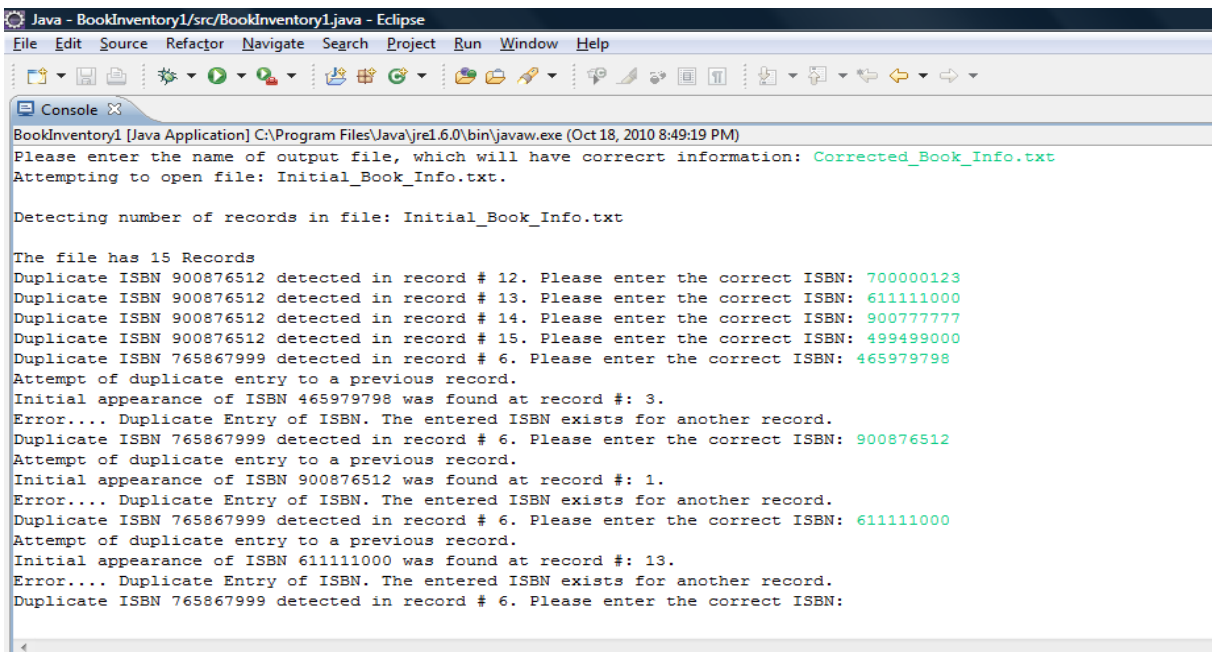


Figure 2: Detecting Existing ISBN

Figure 3: Displaying File Contents (*partial image*)

Part 2

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

1. For that part you need to implement a public class called **BookInventory2**, which will utilize the **Book** class and file called *Sorted_Book_Info.txt*. That 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.
2. 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. A method called **addRecords()**, which accept one parameter, an output file stream name; further details of this method are given below.
 - b. 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.
 - c. 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 (that is $O(\)$ complexity).
 - d. 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 yur 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 *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). 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.
- 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 part1. 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 an 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 (attempt to find best and worst cases) and compare the $O()$ complexity of the two searches.
- Finally, store all the objects on 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.

Submitting Assignment 3

- Zip the source code for part 1 along with all files used/created in that part1 (Please use WINZIP).
- Zip the source code for part 2 along with all files used/created in that part1 (Please use WINZIP).
- Naming convention for zip files:
The zip files should be called *a3_part#_studentID*, where # is part1, or part2, and *studentID* is your student ID number. For example, for the first part, student 123456 would submit a zip file named *a3_part1_123456.zip*
- Finally, zip the above two files in one zip file called *a3_studentID*, where *studentID* is your student ID number.
- Submit your final zip file at: <https://fis.encs.concordia.ca/eas/> as **Programming Assignment and submission #3**. 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 for Assignment 3 (10 points)

Part 1 (8 points)	
Book & DuplicateISBNException classes	0.5 pt
Correct implementation to fix the inventory	4 pt
<i>displayFileContents()</i> method	0.5 pt
Part 2 (2 points)	
<i>addRecords()</i> method	1 pt
Search Methods & <i>Big O()</i>	2 pt
<i>displayFileContents()</i> method	1 pt
Writing to the binary file	1 pt