Guidance Notes for the Assessed Coursework

Read the specification carefully to understand what is expected in this coursework and what files should be submitted to NESS by its deadline.

Task 1 of the specification suggests that you need to develop a `SortedArrayList<E>` class as a sub-class of the standard library class `ArrayList<E>` from the `java.util` package. Observe that your `SortedArrayList<E>` class should be implemented as a generic class that uses a type variable `E`. This class should contain only one method for inserting a new element into a sorted array list. This insertion method should not use the `Collections.sort` method. Furthermore, this method should be a non-static method and therefore work as an operation on sorted array lists. We know that a "future" sorted array list, on which this method will be invoked, is represented inside the method by `this` reference variable. Into this sorted array list the method is supposed to insert a new element of type `E` in the right place, so the ascending order of the elements in the list is preserved. You need to think carefully about the heading of your `SortedArrayList` class. Will it work with any type substituted for `E`, or should its type variable be constrained in any way? You should be able to find some clues to help you answer this question by looking at slide 41 in the second set of lecture notes.

Task 2 of the specification suggests that a solution to this coursework should include two further classes: one for book objects and the second one for user objects. Each of these classes should work as a new type for book objects and user objects. Each of these classes should have fields, constructor(s) and methods (non-static methods as they should work as operations on objects of a given type/class). You need to decide first on the fields of each class: the attributes of objects of a given class. The clues for the fields for both classes can be found at the top of page 2 of the specification (see points 1 and 2). Then, you need to decide on types of your fields. Sometimes the types are obvious. For example, the field to store the title of a book should be of `String` type. However, for some other fields, you might need to make a choice between a few possible types. Further important information for implementing the classes for books and users can be found in the point 2 of the "**Additional assumptions**" on page 2 of the specification, where it is mentioned that these objects should be stored in sorted array lists. This point also explains how the objects of both classes should be ordered. You are advised to look at slide 38 in the second set of lecture notes, to see an example of a class whose objects can be ordered (`Person` class). Its `compareTo` method defines how the `Person` objects are ordered.

Apart from these three classes, your program should contain a driver class, the class that does not work as a new type for objects in your program but provides a framework for your `static main` method representing your sequential program that you will run at the testing stage of this project. For some ideas on how to structure your `main` method look into the first set of lecture notes at the interactive program about a sports club (slides 23 – 41). Your experience of developing a solution for the Formative Coursework should be helpful here as well.

You might decide to have additional classes in your program, but this is up to you. What about having a `Library` class, for example? Would this help with your overall design? As a software engineer you need to make such decisions. In the first instance, your solution should satisfy the specification. However, this shouldn't be your only concern. You want your program to be readable and maintainable. So, its overall structure is important.

Reading the **Marking Scheme** provided at the end of the coursework specification, you can see that up to 6 marks out of 50 will be assigned for the input and output operations. You have learned how to use I/O files in Java programs by doing the Formative Coursework. The specification of the Assessed Coursework tells you that your program should use input and output files. On the first page of the specification (and continuing on the second page) there is an example content of the input file. You are advised to use it for your program. It follows from the specification that the lists of users and books won't change after the input file has been read by the program (new users or books won't be added; no books or users will be removed). Furthermore, as an interactive program, it should retrieve the information that the librarian will type on the keyboard and display some of the information on the computer screen for the librarian to see, when she/he is talking to the users borrowing and returning books. The users will need to tell the librarian their names and surnames and the details of the books they want to borrow/return. The librarian will be entering these details by using the keyboard. See the descriptions of operations `i` and `r` in the points 3 and 4 of the "Additional assumptions" on page 2 of the specification. In the **Submission Notes**, on page 3 of the specification, it is written that you should submit "the output file used by your program". Most information produced by the program should be displayed on the computer's screen. The only information that will be directed to the program's output file is described in point 3 of the "Additional assumptions": "**If the book is currently on loan, a note to the user who is holding the book should be printed to a file informing that the book was requested by another user and should be returned as soon as possible.**" If you create only one object of the `PrintWriter` type in your program all such notes will be appended to the same output file, which will contain at the end the full record of the printed notes. This record should be consistent with the content of the second required text file containing the record of librarian's interactions with the program displayed in the Terminal Window/Console. These two text files will be assessed in the section "**Evidence of testing**" of the Marking Scheme. You will test your program by running the `main` method of your program.