Data Structures and Representation (ZEIT2103) – 2020

Assignment 1 – Collections – eBooks

100 Marks = 16%

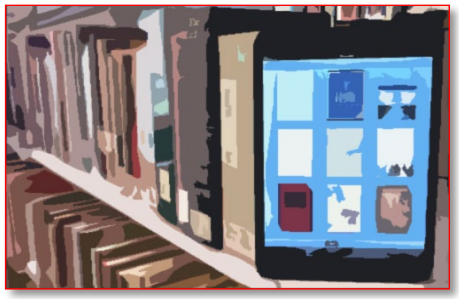
Due Date: 11:59 pm, Fri 3rd April 2019

Assignment Summary

This assignment aims to familiarise you with different collections and sorting algorithms as well as deal with multiple interacting objects. A central theme is designing your own software and appropriate test cases *before* you begin programming.

Problem Description

This assignment aims to manage details of a collection of books, including their titles and authors. The program you will develop should enable the addition and removal of books, tracking those a user has read and sorting them based on any of their attributes. Note that you will not be implementing an actual reading interface, only the parts of the program related to manipulating the list of book details.



Getting Started – Resources

For this assignment, the resources provided in the assignment area of the course's Moodle website is a draft `Harness.java` class which you will update by implementing your test cases in it. Make sure that all the methods you implement comply with the `Harness` class, otherwise, you will get a zero mark for that method.

Assignment Tasks

This assignment involves several tasks: implementing some classes; documenting design decisions; and reflecting on your programming processes. The latter two should be included in the pdf file `ass1.pdf` which should be structured with a heading for each task and sub-headings for the different activities within it.

Task 1. Implementing Book Class (15 Marks)

Your first task is to implement a class to store all the relevant information about a single book, including its title and author, which should have a constructor of the form `public Book(String title, String author)`. You will need the following methods (you can, of course, include additional ones, attributes and constants that you determine are necessary).

- getter methods, i.e., `getTitle` and `getAuthor` **(2 marks)**
- setter methods, i.e., `setTitle` and `setAuthor` **(2 marks)**

  > Always ensure that both the titles and authors are not null or empty and the authors do not contain any non-alphabetic characters; otherwise, you MUST throw an error.
  > **N.B:** there are different ways of doing this, one of which is to use the `match` method in the String class.

- As you will need to search for particular books, this class has to have the method `public boolean equals(Object obj)` to check if two objects are equal. **(3 marks)**
- `String toString()` returns a string representation of a book object. **(2 marks)**

As a rule, whenever you write a new class (or method) that 'models' some aspect of a problem, although it is intended to be used as part of a larger program written in multiple classes, you should always include a 'self-test' `main` method that calls the various other ones in this class and validates that they function correctly. You may do so in the Harness class provided to you. In `ass1.pdf`, you should provide a UML diagram that lists all the attributes and methods you implemented in your `Book` class. Do not forget to also document your java code. **(6 marks)**

Task 2. Implementing `BooksCatalogue` (15 Marks)

In this task, you will need to create the `BooksCatalogue` class and implement the following.

Firstly, the `public void addBook(Book book)` method which adds a book to the collection if it does **NOT** exist, otherwise an exception should be thrown. Obviously, the first decision regards the selection of the data structure you will use to store the collection of dictionary words your class will be given. At this stage, you should use a **`List`** although there are other alternatives. Remember, you will need to add code to declare and initialise the class attribute used to implement the collection. (**5 Marks**)

Secondly, (a) the `public void removeBook(Book book)` method which removes a book from the collection and (b) the `public void removeByAuthor(String author)` one that removes all books authored by `author`. (**5 Marks**)

Then, you must write your 'self-test' `main` method for testing these additional ones. Under the heading 'Task 2' in your `ass1.pdf` documentation file, note any additional design decisions you made or problems you encountered while implementing these methods. Discuss your selection of suitable test values and expected results, and confirm that all tests ran as expected. **(5 Marks)**

Task 3. Search by Name or Title (15 Marks)

This task aims to allow users to search for a specific book by either part of a title or author's name and then return the number of books that match the search. To do this, you will need to implement the following in the `BooksCatalogue` class.

- **`public List<Book> searchByAuthor(String name)`** method which returns a list of the books where `name` is the author's name or part of it. To clarify, if the name of the author of a book is "Saber Elsayed", you can search by "Saber", "Elsayed" or "Saber Elsayed" or even by part of the name, i.e., "Els". The search should be case-insensitive. Then, you will need to implement the **`public int getNumberOfBooks(String name)`** method which returns the number of books returned. **(5 Marks)**
- Similar to the point above, you will need to implement **`public List<Book> searchByTitle(String title)`** and **`public int getNumberOfBooksTitle(String title)`** `methods` which return the number of books matching a title (or part of it). Similarly, the search should be case case-insensitive. **(5 Marks)**

> N.B. I expect one line of code for both the `getNumberOfBooks` and `getNumberOfBooksTitle` methods.

In your documentation file, you should discuss any additional design decisions you made, and remember to test your code. **(5 Marks)**

Task 4. Reading History Reader (10 Marks)

Your next task is to update the `BooksCatalogue` class by adding a new method (`public void addReadingBooks(Book b)`) that keeps track of all the books read (or being read) (stored in the `readingHistory` variable). Note that the latest book read (`x`) should be at the front, that is, if it exists in `readingHistory` but is not the first item, it should be removed from its current location and inserted at the front of the collection. As there are different ways of selecting the type of collection for `readingHistory`, you must justify your selection in the `ass1.pdf` document.

Then, implement `public String getReadingHistory()` that returns a string with all books in `readingHistory`.

Marking scheme: 4 marks for `addReadingBooks`*, 2 marks for* `getReadingHistory` *and 4 marks for the justification of which collection to use (no more than one paragraph (or half a page), testing and java documentation*.

Task 5. Sorting Books (15 Marks)

In this task, you need to implement four methods which should sort books based on either the author's name or book title, in both ascending and descending orders, that is,

- `public List<Book> sortByAuthorAscending()`,
- `public List<Book> sortByAuthorDescending()`,
- `public List<Book> sortByTitleAscending()`, and
- `public List<Book> sortByTitleDescending()`.

As discussed in lectures (week 3), there are different sorting algorithms. You need to justify which one you used in this task and discuss your implementation design.

Marking scheme: 3 marks for each method; and 3 marks for justification, testing and Java documentation.

Task 6. Hashing (25 Marks – no assistance given)

So far, you have tested the program on only a small number of books whereas, in reality, there may be millions of books. This task aims to familiarise you with some basic functions of `HashMap` by implementing a few methods.

- Firstly, create a new class called `BookWithISBN` that extends the Book class, but has an extra attribute, that is, `Integer ISBN` (which represents a unique code for each book). You will need to set it in the constructor and implement the `int getISBN()` method, which returns the ISBN. You may need to override the equals method as well. **(5 Marks)**
- Secondly, create a `BooksCatalogueISBN` class that has one attribute called books of type `HashMap`. You will need to define its key and values and then implement the following simple methods:
  - `public void addBook(BookWithISBN book)` which adds a book to `HashMap` if it does not exist; **(5 Marks)**

- `public void removeBook(BookWithISBN book)` which removes a book given its ISBN; **(5 Marks)** and
- `public BookWithISBN getBook(Integer ISBN)` which returns all details of a book given its ISBN. **(5 Marks)**

The remaining 5 marks are for design justification, UML, testing and Java documentation.

Please note: although you may discuss your decision with your lab tutor, for this task, no help with coding is given.

Final note: for each of the above methods/classes you implemented, you should design, execute and document a suitable test strategy that demonstrates that it functions correctly when used and provide a UML diagram for it.

Task 7. Reflecting (5 Marks)

Reflect on the usefulness of designing tests before implementing them. Did designing tests lead you to re-design your code? Did the tests detect errors in your code as you developed it? Describe your thoughts in your design document.

Marking and Submission

You are required to refer to and follow the '**Requirements and Expectations for Programming Assignments'** document. If you do not, you are likely to be seriously disappointed by your final mark!

If you are aiming for a basic pass/credit result, you need to show your mastery of the basics by completing the following tasks:

- provide and discuss the required designs and specifications, with appropriate justifications and references, and have them noted by your lab tutor;
- ensure that your Java code is well designed, with design decisions documented and the requested UML diagrams provided;
- correctly and efficiently use your Java code to implement the classes and methods specified in the relevant tasks;
- add suitable Javadoc comments to your Java code for each class, attribute and method, and include inline comments on methods where necessary to clarify the intent of the code;
- design your Java code in a modular way using a sensible set of methods;
- ensure that the layout and indentation of your Java code is consistent and readable;
- discuss the design and validation of a test strategy for the code you developed; and
- correctly use proper Java name conventions

To aim for a higher mark, you need to complete all the specified tasks to the basic standard (above) and also discuss the extensions you tackled, the research you conducted, your designs, evidence of your rigorous testing, such as an error-handling code, and big O calculations of the approaches you used.

When you have completed your work, you need to submit the files you have written through Moode.