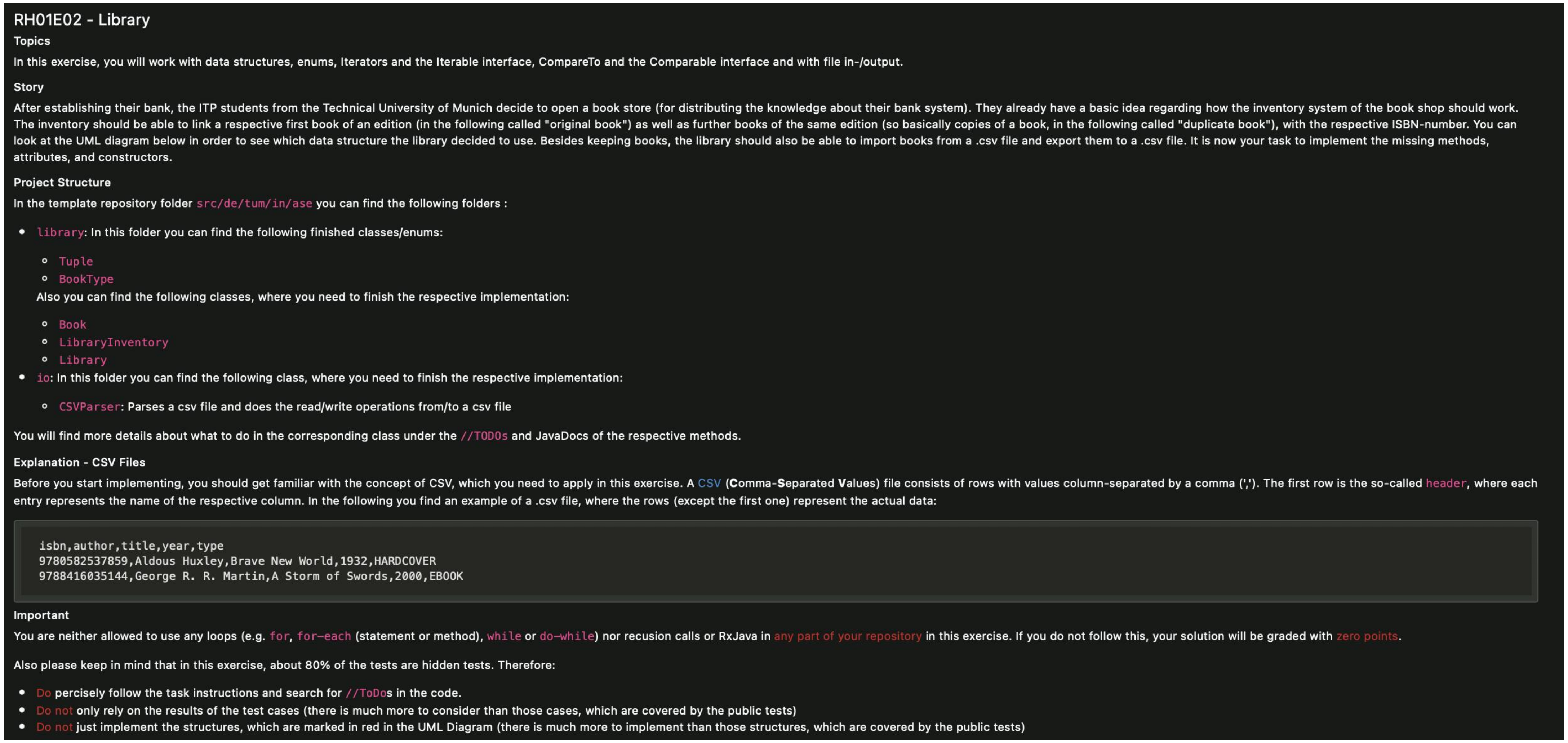


RH01E02 - Library

Topics

In this exercise, you will work with data structures, enums, Iterators and the Iterable interface, CompareTo and the Comparable interface and with file in-/output.

Story

After establishing their bank, the ITP students from the Technical University of Munich decide to open a book store (for distributing the knowledge about their bank system). They already have a basic idea regarding how the inventory system of the book shop should work. The inventory should be able to link a respective first book of an edition (in the following called "original book") as well as further books of the same edition (so basically copies of a book, in the following called "duplicate book"), with the respective ISBN-number. You can look at the UML diagram below in order to see which data structure the library decided to use. Besides keeping books, the library should also be able to import books from a .csv file and export them to a .csv file. It is now your task to implement the missing methods, attributes, and constructors.

Project Structure

In the template repository folder `src/de/tum/in/ase` you can find the following folders :

- `library`: In this folder you can find the following finished classes/enums:
    - `Tuple`
    - `BookType`

  Also you can find the following classes, where you need to finish the respective implementation:
    - `Book`
    - `LibraryInventory`
    - `Library`
- `io`: In this folder you can find the following class, where you need to finish the respective implementation:
    - `CSVParser`: Parses a csv file and does the read/write operations from/to a csv file

You will find more details about what to do in the corresponding class under the `//TODOs` and JavaDocs of the respective methods.

Explanation - CSV Files

Before you start implementing, you should get familiar with the concept of CSV, which you need to apply in this exercise. A CSV (**C**omma-**S**eparated **V**alues) file consists of rows with values column-separated by a comma (','). The first row is the so-called header, where each entry represents the name of the respective column. In the following you find an example of a .csv file, where the rows (except the first one) represent the actual data:

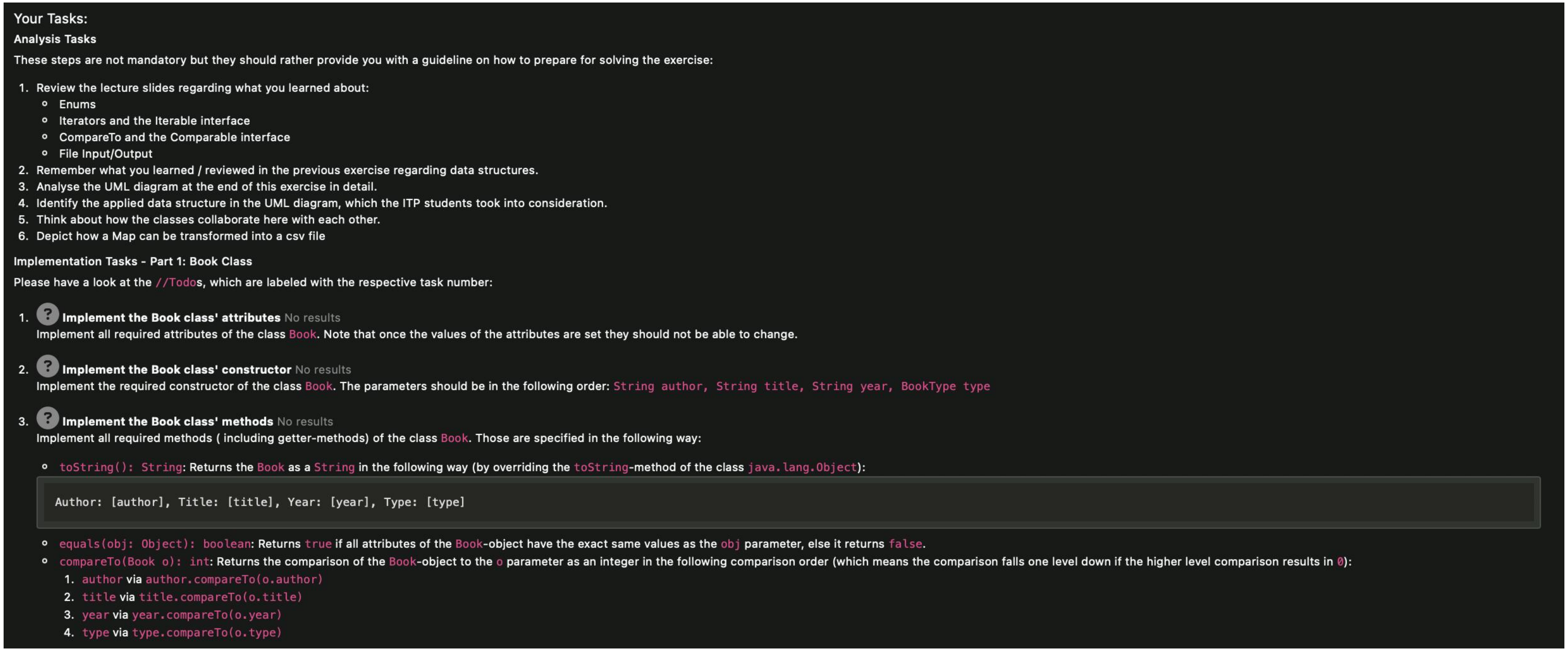
```
isbn,author,title,year,type
9780582537859,Aldous Huxley,Brave New World,1932,HARDCOVER
9788416035144,George R. R. Martin,A Storm of Swords,2000,EBOOK
```

Important

You are neither allowed to use any loops (e.g. `for`, `for-each` (statement or method), `while` or `do-while`) nor recusion calls or RxJava in any part of your repository in this exercise. If you do not follow this, your solution will be graded with zero points.

Also please keep in mind that in this exercise, about 80% of the tests are hidden tests. Therefore:

- Do percisely follow the task instructions and search for `//ToDo`s in the code.
- Do not only rely on the results of the test cases (there is much more to consider than those cases, which are covered by the public tests)
- Do not just implement the structures, which are marked in red in the UML Diagram (there is much more to implement than those structures, which are covered by the public tests)



Your Tasks:

Analysis Tasks

These steps are not mandatory but they should rather provide you with a guideline on how to prepare for solving the exercise:

1. Review the lecture slides regarding what you learned about:
   - Enums
   - Iterators and the Iterable interface
   - CompareTo and the Comparable interface
   - File Input/Output
2. Remember what you learned / reviewed in the previous exercise regarding data structures.
3. Analyse the UML diagram at the end of this exercise in detail.
4. Identify the applied data structure in the UML diagram, which the ITP students took into consideration.
5. Think about how the classes collaborate here with each other.
6. Depict how a Map can be transformed into a csv file

Implementation Tasks - Part 1: Book Class

Please have a look at the `//Todo`s, which are labeled with the respective task number:

1. **Implement the Book class' attributes** No results
   Implement all required attributes of the class `Book`. Note that once the values of the attributes are set they should not be able to change.

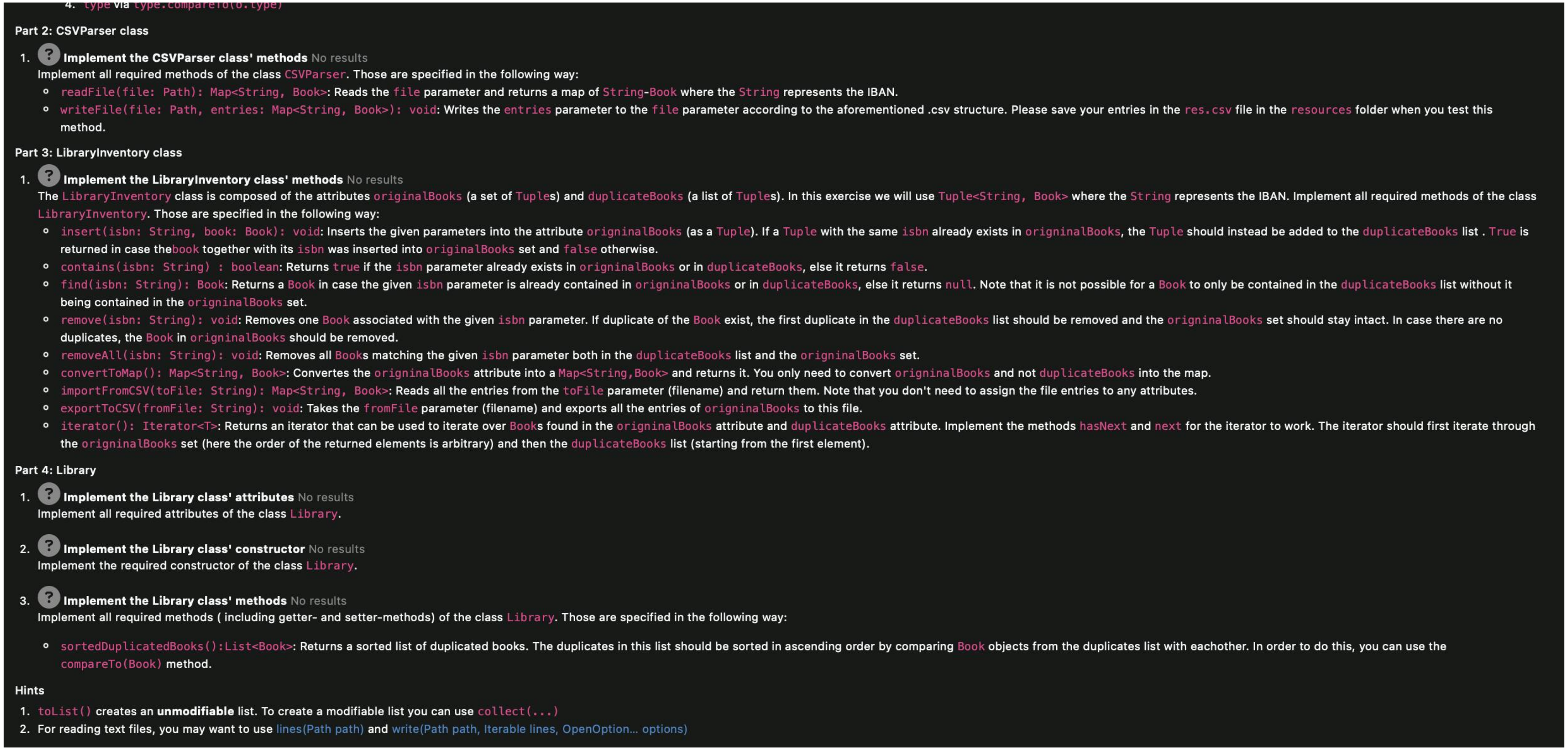
2. **Implement the Book class' constructor** No results
   Implement the required constructor of the class `Book`. The parameters should be in the following order: `String author, String title, String year, BookType type`

3. **Implement the Book class' methods** No results
   Implement all required methods (including getter-methods) of the class `Book`. Those are specified in the following way:

   - `toString(): String`: Returns the `Book` as a `String` in the following way (by overriding the `toString`-method of the class `java.lang.Object`):

   ```
   Author: [author], Title: [title], Year: [year], Type: [type]
   ```

   - `equals(obj: Object): boolean`: Returns `true` if all attributes of the `Book`-object have the exact same values as the `obj` parameter, else it returns `false`.
   - `compareTo(Book o): int`: Returns the comparison of the `Book`-object to the `o` parameter as an integer in the following comparison order (which means the comparison falls one level down if the higher level comparison results in `0`):
     1. `author` via `author.compareTo(o.author)`
     2. `title` via `title.compareTo(o.title)`
     3. `year` via `year.compareTo(o.year)`
     4. `type` via `type.compareTo(o.type)`



4. `type` via `type.compareTo(o.type)`

Part 2: CSVParser class

1. **Implement the CSVParser class' methods** No results
   Implement all required methods of the class `CSVParser`. Those are specified in the following way:
   - `readFile(file: Path): Map<String, Book>`: Reads the `file` parameter and returns a map of `String`-`Book` where the `String` represents the IBAN.
   - `writeFile(file: Path, entries: Map<String, Book>): void`: Writes the `entries` parameter to the `file` parameter according to the aforementioned .csv structure. Please save your entries in the `res.csv` file in the `resources` folder when you test this method.

Part 3: LibraryInventory class

1. **Implement the LibraryInventory class' methods** No results
   The `LibraryInventory` class is composed of the attributes `originalBooks` (a set of `Tuple`s) and `duplicateBooks` (a list of `Tuple`s). In this exercise we will use `Tuple<String, Book>` where the `String` represents the IBAN. Implement all required methods of the class `LibraryInventory`. Those are specified in the following way:
   - `insert(isbn: String, book: Book): void`: Inserts the given parameters into the attribute `origninalBooks` (as a `Tuple`). If a `Tuple` with the same `isbn` already exists in `origninalBooks`, the `Tuple` should instead be added to the `duplicateBooks` list . `True` is returned in case the`book` together with its `isbn` was inserted into `originalBooks` set and `false` otherwise.
   - `contains(isbn: String) : boolean`: Returns `true` if the `isbn` parameter already exists in `origninalBooks` or in `duplicateBooks`, else it returns `false`.
   - `find(isbn: String): Book`: Returns a `Book` in case the given `isbn` parameter is already contained in `origninalBooks` or in `duplicateBooks`, else it returns `null`. Note that it is not possible for a `Book` to only be contained in the `duplicateBooks` list without it being contained in the `origninalBooks` set.
   - `remove(isbn: String): void`: Removes one `Book` associated with the given `isbn` parameter. If duplicate of the `Book` exist, the first duplicate in the `duplicateBooks` list should be removed and the `origninalBooks` set should stay intact. In case there are no duplicates, the `Book` in `origninalBooks` should be removed.
   - `removeAll(isbn: String): void`: Removes all `Book`s matching the given `isbn` parameter both in the `duplicateBooks` list and the `origninalBooks` set.
   - `convertToMap(): Map<String, Book>`: Convertes the `origninalBooks` attribute into a `Map<String,Book>` and returns it. You only need to convert `origninalBooks` and not `duplicateBooks` into the map.
   - `importFromCSV(toFile: String): Map<String, Book>`: Reads all the entries from the `toFile` parameter (filename) and return them. Note that you don't need to assign the file entries to any attributes.
   - `exportToCSV(fromFile: String): void`: Takes the `fromFile` parameter (filename) and exports all the entries of `origninalBooks` to this file.
   - `iterator(): Iterator<T>`: Returns an iterator that can be used to iterate over `Book`s found in the `origninalBooks` attribute and `duplicateBooks` attribute. Implement the methods `hasNext` and `next` for the iterator to work. The iterator should first iterate through the `origninalBooks` set (here the order of the returned elements is arbitrary) and then the `duplicateBooks` list (starting from the first element).

Part 4: Library

1. **Implement the Library class' attributes** No results
   Implement all required attributes of the class `Library`.

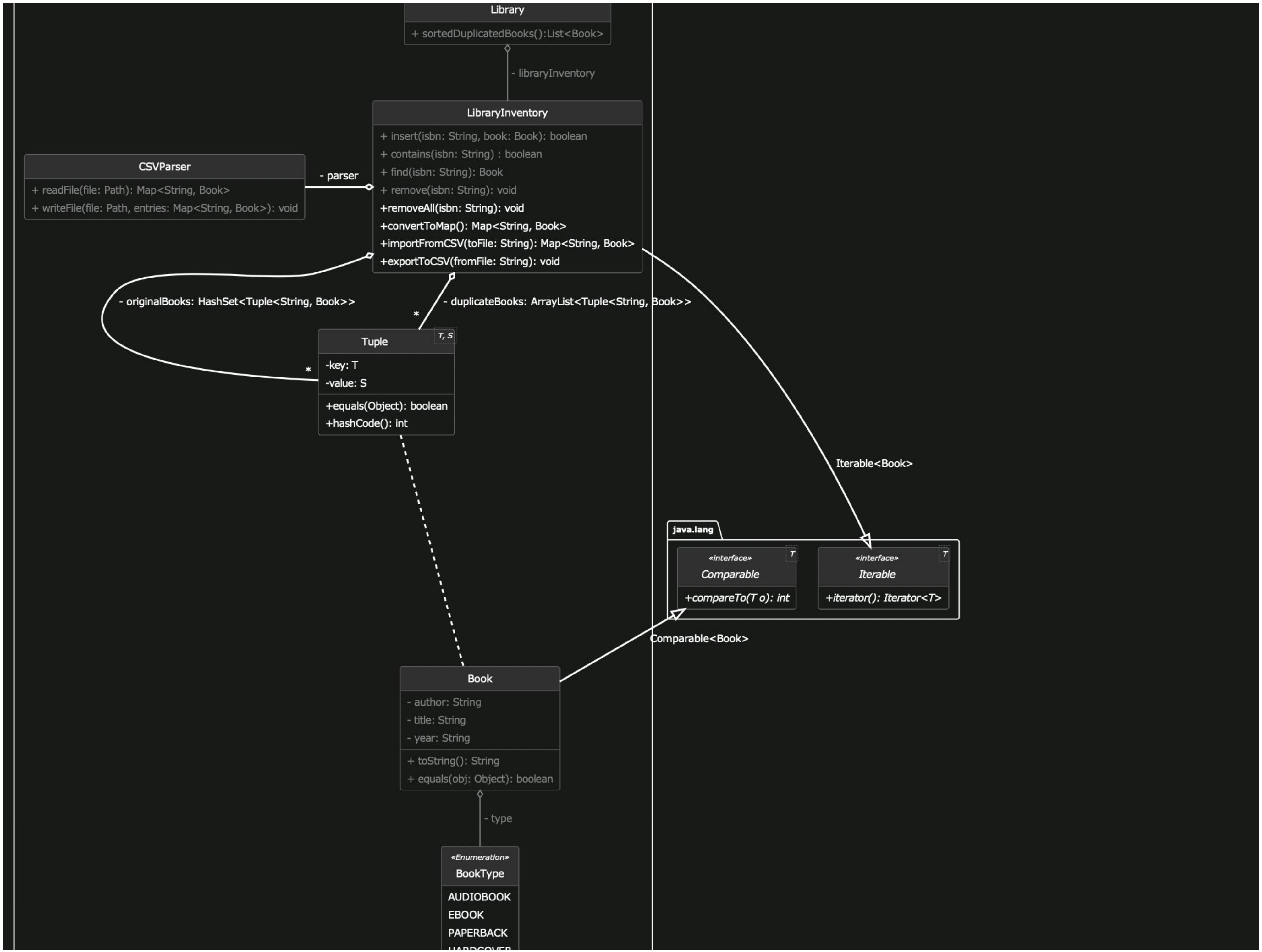
2. **Implement the Library class' constructor** No results
   Implement the required constructor of the class `Library`.

3. **Implement the Library class' methods** No results
   Implement all required methods (including getter- and setter-methods) of the class `Library`. Those are specified in the following way:

   - `sortedDuplicatedBooks():List<Book>`: Returns a sorted list of duplicated books. The duplicates in this list should be sorted in ascending order by comparing `Book` objects from the duplicates list with eachother. In order to do this, you can use the `compareTo(Book)` method.

Hints

1. `toList()` creates an **unmodifiable** list. To create a modifiable list you can use `collect(...)`
2. For reading text files, you may want to use lines(Path path) and write(Path path, Iterable lines, OpenOption... options)


Library
+ sortedDuplicatedBooks():List<Book>
- libraryInventory
LibraryInventory
+ insert(isbn: String, book: Book): boolean
+ contains(isbn: String) : boolean
+ find(isbn: String): Book
+ remove(isbn: String): void
+removeAll(isbn: String): void
+convertToMap(): Map<String, Book>
+importFromCSV(toFile: String): Map<String, Book>
+exportToCSV(fromFile: String): void
CSVParser
+ readFile(file: Path): Map<String, Book>
+ writeFile(file: Path, entries: Map<String, Book>): void
- parser
- originalBooks: HashSet<Tuple<String, Book>>
- duplicateBooks: ArrayList<Tuple<String, Book>>
*
*
Tuple
T, S
-key: T
-value: S
+equals(Object): boolean
+hashCode(): int
Iterable<Book>
java.lang
«interface»
Comparable
T
+compareTo(T o): int
«interface»
Iterable
T
+iterator(): Iterator<T>
Comparable<Book>
Book
- author: String
- title: String
- year: String
+ toString(): String
+ equals(obj: Object): boolean
- type
«Enumeration»
BookType
AUDIOBOOK
EBOOK
PAPERBACK