CmpE 160 - Introduction to Object Oriented Programming

Project #1 - Simulation of Library Operations
Due Date: 04.04.2019 23:55 PM

1.Introduction

In this project, you are going to implement a simple library management system, resembling the typical daily operations within the library in our university. The main objective of the program is analyzing the incoming requests provided as a list of input, and carrying out the necessary actions.

The members of the library, students and academicians in particular, are able to borrow books and return them before the specified deadline, otherwise they should pay the fee in order to be able to borrow another book. It is also possible for a member to extend the deadline only once for each book (the requirements and details of a possible extension is presented in the following sections of this document). Additionally, a certain set of books cannot be taken out of the library, it is only possible to read them within the library since they are rare to find a copy, and only academicians could reach them. Of course, before such operations, the members should be registered to the system, and so they can begin to benefit from the services provided by the library.

Besides of the typical daily operations triggered by the members, the book collection can be enlarged by adding a new book.

Please note that the program will not require a user interaction during the execution: it parses the input file composed of the sequential library operations each of which occurs at consecutive unit of time. The following section provides the necessary information for you to implement the requirements of the project.

Also, as a significant remark, the signature of the methods and the field names that you are going to implement should be identical with the ones that are specified in this document. As a brief example, the `timerTick` method is implemented in `LibrarySimulator` class with the signature `void timerTick(Action action)` with an appropriate access modifier to be determined according to the inheritance and encapsulation principles. If a specific method signature is not enforced by the project description document, you can feel free to implement in your own way (i.e. you can implement extra methods considering your own design choice). However, you should also consider the proper usage of access modifiers for preserving the desired visibility and accessibility. In other words, you should not define everything as “public”, which results in a penalty if done so.

2. Class Hierarchy and Implementation Details

You are provided with the following classes:

- `Main`
- `Action`
- `LibrarySimulator`

You are also given the following interfaces:

- `Borrow`
- `ReadInLibrary`

Do not modify the code in these files! In addition, you are given a Main class. Read that class and try to understand the operations of the simulator. You can play with the main class to test your code. During testing and grading phase, all changes made in these classes will be reset. You are responsible for all the compilation errors originated from the changes made in any of these classes including the addition or removal of libraries.

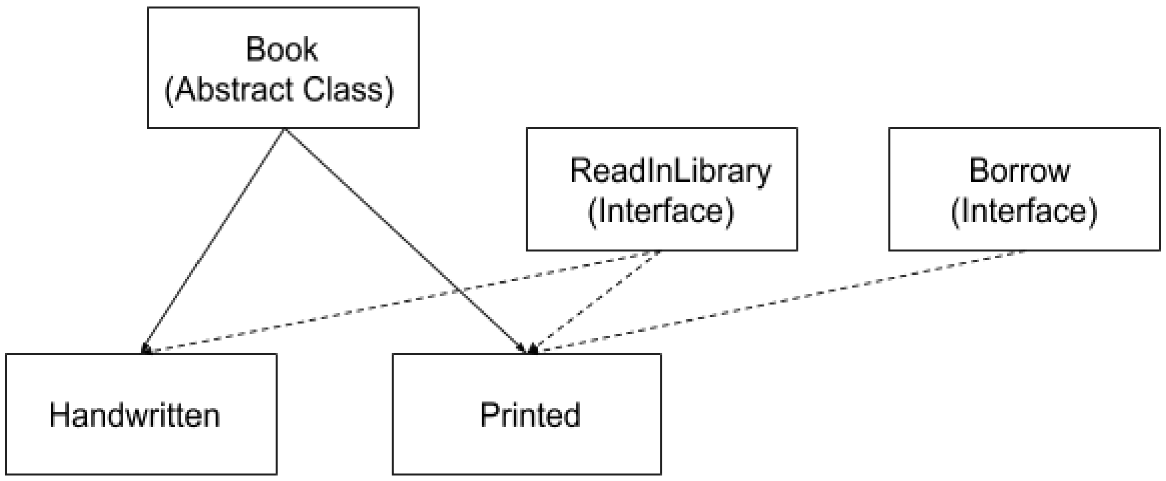
The other files are just empty. Using the fundamentals of the object-oriented programming methodology and the inheritance principles, you are expected to fill in these files in a suitable manner to complete the library simulator. Do not forget that you are free to create new packages, classes or interfaces. However, the class names and properties that are given below should be exactly the same in your projects. **Please note that,** if you fail to follow this instruction, you do not get any points as your projects will be automatically graded.

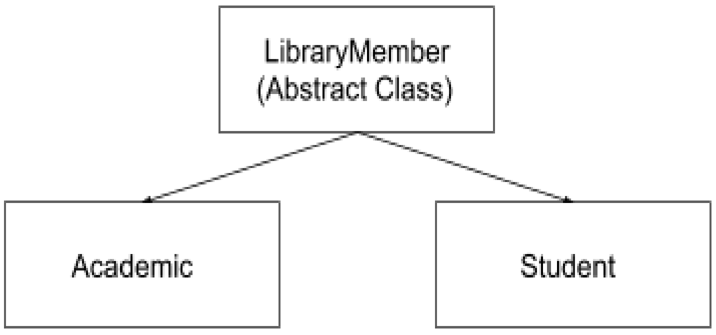
Before Cmpe 160, there was a single source file and all the functions were implemented in that file. However, in this project, there will be multiple classes that interact with each other during execution. Therefore, before starting the implementation, you are advised to allocate some time to understand the main logic behind the design and the relation among the code pieces.

Within the scope of this project, you are expected to implement these classes (which will be explained in detail in the rest of this document):

1. `Library.java`
2. `Book.java`
3. `Handwritten.java`
4. `Printed.java`
5. `LibraryMember.java`
6. `Academic.java`
7. `Student.java`

The class hierarchy that we have designed is illustrated in the class diagram below which we expect you to follow:





You need to consider the rules designated by the class diagram and create the classes considering the hierarchy and class properties.

Main.java:

The main method simply reads an input file, that is composed of sequential commands related to the library operations. As stated above, you can use this class to test your code. The set of instructions within this class is already given for you to understand the operation.

Main method creates an instance of `LibrarySimulator` and for each line of the input file, it calls the `void timerTick(Action action)` method of the `LibrarySimulator`. `tick` is the time unit of the project, which is stored in the `LibrarySimulator`. In each `Action`, the time goes by one `tick`. No two actions can happen at the same `tick`.

The main method just reads the type of the `Action` and calls the `timerTick` immediately. Therefore, the remaining parts of the input line must be read in the `Library.java` and not in elsewhere. For this reason, scanner of the main method is given as an argument to the constructor of the `LibrarySimulator.java` which in turn, transfers this scanner to the `Library.java`.

Before going into the details of the `LibrarySimulator` class, let's define the format of the input file in a clear manner.

INPUT FORMAT:

- In the first line, there will be number **"n"**, which represents the total number of events that will occur during the simulation.
- In the each of the following **"n"** lines, the `Actions` in the library will be specified in their customized format as described below. In other words, each distinct line represents an event with a specific action in the library.

The possible actions in the library is encoded through separate numbers:

1 => `void addBook()`
2 => `void addMember()`
3 => `void borrowBook(int tick)`
4 => `void returnBook(int tick)`
5 => `void extendBook(int tick)`
6 => `void readInLibrary()`

<u>Adding a new book:</u>

While adding a new book to the library collection, only the type of the book will be given and the book id will be determined by `Library`. The id of a book is at most a six digit number and an id should be unique to a book. In other words, no two books can have same id, even if one of them is a printed book and one of them is a handwritten book. Successive numbers should be assigned as id values of the books according to the order of adding them to the library collection starting indexing from 1, such that the first book in the library must have the id equals to 1 and the second book has id 2 and so on.

1 P => 1 means addBook, P means the type of the book is Printed
1 H => The type of the book that will be added is Handwritten

P => printed, H => Handwritten

Adding a new member:

While adding a member, only the type of the member will be given and the member id will be automatically determined by the library. The id of a member is at most a six digit number and no two members can be assigned with the same id. The procedure of adding a member is the same as with the procedure of adding a new book.

2 S
2 A

S => Student, A => Academic

Borrowing a book:

3 Id_Of_Book Id_Of_Library_Member

Returning a book:

4 Id_Of_Book Id_Of_Library_Member

Extending a book:

5 Id_Of_Book Id_Of_Library_Member

Read In library:

6 Id_Of_Book Id_Of_Library_Member

LibrarySimulator.java :

`LibrarySimulator` is like a general manager in a Library. A manager does not accomplish tasks by itself, instead it connects the different departments of the library and assign them these tasks. That is what `LibrarySimulator` implements.

In `LibrarySimulator` there is a constructor and a method whose signature is `void timerTick(Action action).`

Each event in the library represents one call for `void timerTick(Action action). tick` is incremented by 1 in every event. Also, no two events can occur at the same time. In each `timerTick` call, `LibrarySimulator` receives one action specified by the corresponding event and requests to the `Library.java` to execute the necessary instructions for realizing the event. It is entirely up to the `Library` class executing the actions or dismissing the incoming request, because the requested action should be validated. For example, if a member desires to borrow a book by specifying its ID, which is not available within the library at that time instant, the `Library` class should invalidate the event and the program continues with the following event and `timerTick`. In case of an invalid request, you have to just skip that action and do nothing else. All the check operations should be carried out in the `Library.java`, not in elsewhere. You should not modify `LibrarySimulator.java.`

Action.java :

In the `Action` class, the possible events are listed. You should not modify this class!

Library.java :

You are expected to design and code `Library.java` from scratch, considering the requirements and rules as follows.

- In `Library.java` you must read the remaining parts of the input line according to the type of the `Action`. Because in the `main.java` only the type of the action has read.
- The Library must have two separate Arrays: one for storing books that are registered in the library system and one for the members. In order to access these arrays, you should implement the getter methods with the signatures `Book[] getBooks()` and `LibraryMember[] getMembers().`
- The library must have a variable `int totalFee,` which holds the total amount of fee that is paid to the library.
- Implement the default constructor with taking a Scanner as a parameter. As explained above, you should use this Scanner object to read the remaining of the input related to the type of the Action. Also, you should initialize the book and member arrays with the size of 10^6.
- Implement the `void addBook()` method
  - This method should assign an id to the book incrementally, check the type of the book to be added, and add the corresponding book object to the library collection (the books array).
- Implement the `void addMember()` method
  - Before adding the member, this method should initially assign an id to the member incrementally. Check the type of the member, either student or academician, and store the corresponding member object in the library system (the members array).
- Implement the `void borrowBook(int tick)` method
  - After the necessary checks for the validity of a borrowing event, if validated, this method completes the borrowing operation by making necessary function calls that makes the corresponding modifications in the fields of the book.
- Implement the `void returnBook(int tick)` method
  - Following the necessary checks for the validity of returning a book, the member returns the book to the library. However, if the deadline is missed, there is a penalty and the member should pay the fee for being able to borrow another book.
  - Fee should be paid = (the current `tick)` - (the deadline of the book).

    - Each fee paid by the members must be added on the variable that holds the total amount of fees paid. One should be able to access the total fee through `int getTotalFee()` method.
- Implement the `void extendBook(int tick)` method
    - Before the deadline, the member can extend the deadline further in this method. The necessary check should be carried out for a possible extension.
    - After the extension, the deadline = the current deadline + time limit for the `LibraryMember`.
- Implement the `void readInLibrary()` method
    - If a member desires to read the book instead of borrowing it, the necessary instructions should be executed by this method. Contrary to `extendBook` method, there is no deadline for returning the book in reading in the library.

Besides these methods, as described before, this class should implement the necessary functions for validating the operations requested by the `timerTick`. **You are responsible to check for all possible erroneous requests from timerTick.**

Book.java :

It is an abstract class. The `Printed` and `Handwritten` classes inherits from the `Book.java`.

In Book.java, the common properties of book that is independent from its type will be implemented.

Book must have this variables, exactly as named below:

`int bookID` : At most a 6-digit ID number

`String bookType` : One character String, “P” or “H”, represents the type of the book.

`boolean isTaken` : The variable that holds whether the Book is taken or not

`LibraryMember whoHas` : The variable that holds the library member who took the book if it is taken. If it is not taken, then it must be set to the null value.

The book must have these methods, exactly as named below:

- A constructor with two parameters, `bookID` and `bookType`.
- An abstract method `void returnBook(LibraryMember member)`

Additionally, the choice of defining the variables private, protected or public may require or may not to define additional getter and setter methods.

Handwritten.java :

The class holds the properties for the Handwritten books and related methods.

- It must have constructor with one parameter, namely `bookID`.

- It must override the methods that are inherited from `Book.java` .
- It must implement the `ReadInLibrary` interface.
- Since the handwritten books can be only read in the library without borrowing, overriding the `void returnBook(LibraryMember member)` method should just modify the states of the `boolean isTaken` and `LibraryMember whoHas` fields.

Printed.java :

The class holds the properties for the printed books and related methods.

Printed must have this variables, exactly as named below:
`int deadLine` : Holds the deadline for the printed book if it is taken. If it is not taken, then it must be set to 0.
`boolean isExtended` : Holds the borrower of this printed book whether extended the deadline of it previously, or not.

- It must have constructor with one parameter, namely `bookID`.
- It must override the methods that are inherited from `Book.java` .
- It must implement the `ReadInLibrary` and `Borrow` interface.

LibraryMember.java :

`LibraryMember` class holds the properties for the members and related methods.

- Each `LibraryMember` must have an ID defined by `int id`. This ID is up to 6-digit integer.
- Each `LibraryMember` has a book limit defines the quantity of the books that could simultaneously be borrowed, as `int maxNumberOfBooks`. This book limit depends on the type of the `LibraryMember`.
- `LibraryMember` have a constant time limit to return the borrowed book for each book which is defined through `int timeLimit`. This time limit also depends on the type of the `LibraryMember.` If the Book is not returned in time, then the `LibraryMember` must have to pay a fee.
- The history of the books that are either read or borrowed by a library member should be stored in an `ArrayList`, and the abstract method that returns this instance of `ArrayList` must be declared as `abstract ArrayList<Book> getTheHistory()` to be implemented by the subclasses of the `LibraryMember` class.

Academic.java :

The first class that extends the `LibraryMember` abstract class is the `Academic`. The instances of the `Academic` class represents the academicians to utilize the services provided by the library. The properties and possible behaviors of the `Academic` class are defined as follows:

- An academician can have 20 books at most simultaneously.
- The time limit to return a borrowed book should be defined as 50 unit of time.
- Only the academicians have an access to the `Handwritten` books
- It must have a constructor with one parameter, namely `int id`.
- It must override the methods that are inherited from `LibraryMember.java` when necessary.

Student.java :

There is another set of library members composed of students, which is defined through the `Student` class that should extend the `LibraryMember` abstract class. Similar to the `Academic` class, the students as members have separate properties and abilities that are permitted by the library:

- A student can borrow 10 books at most simultaneously.
- The time limit to return a previously borrowed book is 20time unit.
- While academicians are defined as “A” , the students should be represented through “S”.
- The students do not have permission to read a `Handwritten` book.
- It must override the methods that are inherited from `LibraryMember.java` when necessary.

Borrow.java :

`Borrow` is an interface that is already given to you, so that you should not modify this interface. The `Printed` class must implement this interface. There are two different methods that are declared by this interface to be implemented by the `Printed` class:

- `void borrowBook(LibraryMember member, int tick):` The body of this method signature should be composed of the instructions for a member, which is passed as an argument `LibraryMember member`, borrowing a desired book. Besides, the second argument, `int tick`, represents the time tick that the event occurs.
- `void extend(LibraryMember member, int tick):` This method provides an opportunity to extend the deadline, for once. If the deadline is not missed yet and the deadline of the borrowed book is not extended before, this method doubles the time limit permitted to the `LibraryMember member`. Similar to the previous method, `int tick` represents the time tick when the event is triggered.

ReadInLibrary.java

ReadInLibrary is another interface that you are already provided with. You should not modify anything within this interface. Both the `Printed.java` and the `Handwritten.java` must implement this interface. This interface defines a simple behavior for the classes that implements. While some of the books are borrowable by the members and can be taken out of the library, the Handwritten books are not permitted to be borrowed by any member. On the other hand, any member can borrow a Printed book, but it is also possible to read it without borrowing. Therefore, the only method declared by this interface, `void readBook(LibraryMember member)`, involves the necessary instructions for a `LibraryMember member` to read the corresponding book. Since it is not borrowed, there is no specific deadline and you do not need to consider the time tick that event occurs.

3. Some Remarks

- The method signatures and the field names should be exactly the same with the ones that are specified by this document. However, you can implement additional methods as you desire.
- Please keep in mind that providing the necessary accessibility and visibility is important: you should not implement everything as public, even though the necessary functionality is implemented. The usage of appropriate access modifiers and other Java keywords (super, final, static etc.) play an important role in this project since there will be a partial credit, specifically for the software design.
- There will be also a partial credit for the code documentation. You need to document your code in Javadoc style including the class implementations, method definitions (including the parameters, return if available etc.) and field declarations. You do not need to create and submit a documentation file generated by Javadoc as the software documentation.
- Please do not make any assumptions about the content or size of the scenarios defined by the input test files. Your project will be tested through different scenarios, so you need to consider the all the possible criteria and implement the code accordingly.