Object Oriented Programming - Exercise 0: Library System

Contents

1 Objectives

In this exercise you will practice basic Java programming principles, including the use of Java primitive types, flow control, loops and the creation of basic classes. You will also practice writing Java code using a text editor, and compiling and running Java programs using the command line. You will be required to implement classes by following an API provided to you by us.

> An *API (Application Programming Interface)* is a set of functions, procedures and protocols that determines the way a certain software component can be used without dealing with the way it is implemented, and expresses that software component in terms of its operations, inputs and outputs.
>
> When we use the term in this course, we usually mean a set of documents describing the different parts of the program you are required to write, where each part is described by the set of operations that can be performed by it and on it.

In a very real sense, an API is like a contract that your program is obligated to fulfill; we tell you what operations it should be able to perform, and what will be their exact results, but how to implement those operations is up to you.

We will further discuss the concept of APIs later in the course.

For a short introduction on reading and understanding Javadoc APIs see the *"Reading Javadoc APIs"* guide on the course website.

We recommend you use the text editor *gedit*, which is already installed on CS lab computers, for this exercise; you can find a short tutorial on how to work with *gedit* in a separate document on the course website.

For a reminder on command-line compilation and execution of Java programs, see section 5.

2 The Library System

The classes you will implement will simulate a library system. A library system consists of libraries, patrons and books. Each of these components is represented by a corresponding class.

Libraries hold some number of books - limited by their individual book capacity - and allow each patron to borrow books up to the allowed limit. Once a book is borrowed by some patron, it cannot be borrowed by another patron until it is returned by its current holder.

Besides borrowing limit and book availability, borrowing is also affected by the preferences of the patrons: each book has three characteristics that compose its literary value - its comic value, its dramatic value and its educational value. Different patrons ascribe different levels of importance to these three characteristics; this is reflected by the individual set of weights each patron assigns them. The weighted sum of the above characteristic reflects the total value a patron assigns to some book, and a patron will only borrow a book that she will enjoy - i.e. a book to which she assigns a value beyond some threshold.

For example, let us assume that the book *"Good Omens"* has a comic value of 8, a dramatic value of 5 and an educational value of 3. And let us also assume that the patron Crowley weighs the comic aspects of book with a value of 2, the dramatic aspects with 5 and the educational aspects with 6. Than, the total value Crowley will assign *"Good Omens"* will be $8*2+5*5+3*6 = 59$.

3 Exercise definition

Your task in this exercise is to write three Java classes: Book, Patron and Library, where each is implemented in a separate Java file; `Book.java`, `Patron.java` and `Library.java`. The three classes composing the program, including the two classes you need to implement from scratch, are well documented in a publicly-accessible Javadoc.

Javadoc is a collection of HTML files - which you can read using your internet browser - that display, for each class, the list of public fields and methods it contains, and for each such method - details about the return type and the parameters it receives (this is called the method's signature).

You will have to implement the given API. For the matter of this exercise, the API simply contains all the method signatures that the different classes have to contain. You may add methods and fields not mentioned in the API, but they should only be used to implement the API, not change it. Think carefully before deciding which additional methods your program needs.
For example, the Patron API describes the method willEnjoyBook which has the signature: boolean willEnjoyBook(Book book). This means that the Patron class has to implement this method, or in other words, the file Patron.java has to contain the following code:

```
boolean willEnjoyBook(Bookbook){
        *yourcodegoeshere*
}
```

You are provided with a class skeleton, or a template, for the Book class, and an implementation of its method stringRepresentation(), and are required to implement the rest of its methods. You are also required to create the class files of the other two classes and implement all their required methods from scratch. You working individually.
When writing the code, testing it and submitting your solution, you must be working individually. You cannot look at the code other students wrote or ask them to describe it to you, and you most certainly cannot copy other people's code. You can, however, and are encouraged to, use code examples from the lectures, tirguls and the coding style document as a starting point.

[If you do not know what access modifiers are - that's ok, you shouldn't, so just skip this box] In order to comply with the API, **do not add access modifiers** (public, protected or private) **to any of the methods defined by it**.

4 The classes composing the program

4.1 The Book class

Defined by the Book.java file. The Book class represents a specific copy of a book that a library can hold. A book has a title, author, year of publication and three different measures of its literary value - comic, dramatic and educational.
The Book class should support all methods defined in the Book API (click on the link for the full API specifications); Notice that you receive a (very) partial implementation that already declares seven fields, a constructor and all five required methods, and also implements one method (String stringRepresentation()), but you have to use the arguments of the constructor to initialize the fields of this class, and implement the **four** additional methods.

4.2 The Patron class

Defined by the Patron.java file. Each patron has both first and last names, three weights he assigns to the three respective literary aspects of books and an enjoyment threshold - a minimal (personal) score a book must be assigned by her in order for her to enjoy that book.

The Patron class should support all methods defined in the Patron API; Notice that you must again implement the constructor and all three methods described by the API, with identical signatures.

4.3 The Library class

Defined by the Library.java file. Each library can hold a certain number of books, and manage a certain number of patrons registered to it - both of these numbers are supplied to the constructor when an object is initialized. In the same manner, each library has its own unique limit to the number of books a patron can borrow at the same time - again, determined by another initialization parameter supplied to the constructor.
Whenever a patron attempts to borrow a book, the action will succeed only if that book is available - meaning the library has a book of the given id, and it is not borrowed by some patron **and** if the borrowing patron has not reached the borrowing limit **and** he will enjoy the requested book.

The Library class should support **all ten methods** defined in the Library API.

4.3.1 Java arrays

In your implementation of the Library class you will have to use some representation for the lists of books and registered patrons, and the basic candidates in Java are arrays. We will learn more about Java arrays further along in the course, but for now it is enough to know that they are somewhat similar to the lists you know from Python. The basic principles needed for using arrays in this exercise are:

- Like all variables in Java, arrays are typed, so you need to declare the type of primitives or objects you want to hold in the array. The type of an int array is int[], that of a boolean array is boolean[], and an array of Dog objects is of the type Dog[].

- In Java you declare the size of the array on initialization, and it cannot change afterwards. You can declare an array of 7 integers with int[] myIntArray = new int[7];, and an array of 12 Dog objects with Dog[] dogArray = new Dog[12];. The size can also be determined using a variable, as in: int x = 43;
  Dog[] dogArray = new Dog[x];.

- Access to array elements is done with square brackets holding the index of the desired element, when the first element has index 0 and the i_{th} element has index $i-1$ (so the last element in an array of n elements has index $n-1$). You can access the first element of your int array with int firstElement = myIntArray[0], and the 5_{th} element of your Dog objects array with Dog fifthDog = dogArray[4];.

  Also, if the Dog class has a bark() method, you can 'make' the fifth dog bark by writing dogArray[4].bark(); Also note here you cannot use negative indices (as opposed to Python).

- Assignment to array elements is done using the same syntax, so you can assign 82 as the first element of your int array with myIntArray[0] = 82;, and 'adopt' a new dog as the 7_{th} element of your Dog objects array with dogArray[6] = buster; (assuming Dog buster = new Dog("Buster"); was defined earlier, of course).

You can't remove primitives from an array (so it is common to assign 0 instead of the 'deleted' value in int arrays, for example, although the right value for a deleted cell is contextdependent), but you can remove objects from an object array by assigning null to the corresponding cell (which is a little like pointing that cell at 'nothing') with dogArray[10] = null; (so null is to Java what None is to Python).

5 Testing and Execution

Your code is required to compile and run correctly on the Linux environment that is currently installed on CS lab computers. You can write your code using any tool and in any OS you wish to use, as long as you make sure that it compiles properly and returns the correct output when it runs on CS computers, **and** upon submission to the course *moodle* site - when you submit your code, it will be compiled and executed automatically, and you will receive a report detailing how it went. Nevertheless, we strongly advise that you use the CS computers to compile and execute your code in the CS lab Linux environment (see how in the following subsections).

We also strongly suggest that for the first exercise you don't use any tools other than the javac and java commands, and a text editor such as gedit on Linux or Notepad++ on Windows.

Remember: Check your submission results for any errors or failed tests!

5.1 Compilation

As you have learned in class, compilation is the process of transforming high level program code (in our case, Java code) into lower-level instructions (in our case, Java bytecode), in order to be able to run it later. To compile a Java file (e.g the Book.java file), issue the command:

javac Book.java

If the code is syntactically correct, and the compiler is able to find all the classes the code refers to, this will produce the file Book.class, which contains the compiled bytecode.

If there are errors in the code, the compilation errors are presented to the user, and no class file is produced. A short description and location information is printed for each error, which you can use to find and fix it.

5.2 Execution

Once you have a .class file (e.g a file called Test.class), you can execute it with the command:

java Test *(and not the command* java Test.class*)*

There are several conditions required for the program to run:

- The Java file you want to run (in this case Test.java) must contain a *main* method, which has the signature:

  public static void main(String[] args)

  (More about the different keywords in this signature in future lessons).

So, for example, in order to create an array of Book objects, then print their literary value one by one, we could write the following class, which would contain the main method:

```
public class Tester {

  public static void main(String[ ] args) {

        Book[] books = new Book[10];        // create an array of 10 books.

        for(int i = 0; i < books.length; i++) {

              books[i] = new Book(/* here you insert the constructor parameters*/);

        }

        for(int i = 0; i < books.length; i++) {

              System.out.println(books[i].getLiteraryValue());

        }

  }

}
```

- If the program uses other user defined objects (e.g. Book), then the corresponding class files (in this case, the Book.class file) also have to be in the same folder (we will see a more general criteria later in the course).

When one of the above conditions is not met, running the program generates a *runtime error*, which cannot be discovered during compilation. We will learn more about those errors later in the course.

6 Simplifying assumptions

- Generally, you can assume the constructors of the three different classes will receive only valid input; books will receive valid strings as titles and authors, and positive integers as year of publication and the different literary values; the same goes for the parameters used to initialize Patron and Library objects.
- You can assume you will always get valid objects as arguments, so you **don't** have to check for null Books in getBookScore(), willEnjoyBook() or addBookToLibrary().
- The same goes for Patron objects in registerPatronToLibrary() and getPatronId().
- You **cannot** assume the Book object you receive in addBookToLibrary() is not already in the library.
- You **cannot** assume the Book object you receive in getBookId() is present in the library.

- You can't assume id values you get in methods such as isBookAvailable(), borrowBook() and returnBook() are valid, meaning that they were ever assigned to some Book object by this library.
- The same goes for Patron ids in borrowBook() and suggestBookToPatron().

7 Further guidelines

- Write clear, readable code, that is as self-explanatory as possible. This includes choosing informative names for methods and variables, and the way you organize your code.
- In places where your code isn't self-explanatory, clarify it, to others and yourself, using documentation.
- Test each class separately before testing them all together. Create a simple program that creates a Book object, and test each of its methods to make sure that they function correctly. Do the same with the Patron class. Finally, test to see that Library objects interact correctly with Patron and Book objects.
- While running your code, you need to compile after every few new lines you add, and run it after every new method or two you add, to make sure you are always working with functional code.
- Since the program you are required to write as part of this exercise has no main method, you should write a separate test class, which will contain a main method; in this method you will be able to use elements of your program (in this case, Books, Patrons and Libraries) and check for their correct behavior. You can then run your test class as detailed above (that is if your test class with the main method is named Test, have your Test.java file in the same folder as your other classes, execute javac for each java file and then execute java Test).

  If you do write such additional classes, **do not submit their source files (**.java**) or class files (**.class**)**!
- Your code should be written according to the course coding style guidelines (see here).
- You are encouraged (and should) ask questions about the exercise in its designated forum.

8 Grading and Testing

The grade for this exercise is based on both automatic testing (50%) and code review (50%). You will be able to see the result of some of the automatic tests as soon as you submit your solution. You can correct your errors and resubmit as many times as you want before the deadline.
The rest of the tests are hidden and will tackle more subtle aspects of your code. Thus, you are encouraged to test your code on more elaborate scenarios.

- If you submit during the late submission period, your grade will be penalized, even if the new submission is identical to a previous one.
- Any submissions after the late submission period will not be accepted, unless authorized by the course staff, via the personal requests forum.
- **DO NOT** send messages to our private email addresses.
- A submitted solution that contains code which does not compile will get an automatic zero.
- See also the the Course Requirements, Emails and Forums document.
- And the the Exercise Preparation and Submission Regulations document.

9 Submission

- The exercise is due on **Wednesday, March 28th at 23:55**.
- You should submit **only** the following files:
  1. Book.java
  2. Patron.java
  3. Library.java
  4. README *(as explained here)*
- Make sure you do not submit any .class files by mistake.
- Create a JAR file named ex0.jar containing only these files by invoking the shell command jar cvf ex0.jar Book.java Patron.java Library.java README with the shell's current working directory set to the folder containing the above files. Your JAR file should not contain any other files.
- Make sure that the JAR file you submit passes the presubmit script by **carefully** reading the response file generated by your submission. **Exercises failing this presubmit script will get an automatic 0!**

Good Luck !