University “Politehnica” of Bucharest

Faculty of Electronics, Telecommunications, and Information Technology

**Library Management System**

**Object Oriented Programming**

Student: Neagu Alexandru

Group: 412G

2022

**Introduction**

The current application simulates a basic management system for a library. It is designed to store and create information regarding the librarian, the students who are lending the books. The program allows the user to create an entity, called student, which has the attributes: id and name. We can create also books, which have the following attributes: book number, name of the book and the author’s name. Also, if the student returns an issued book in more than fifteen days, the program will generate a message that will be the fine for that specific student. To interact with the user, it was built a command line interface structured as a numbered menu. The operator can select an option by entering its corresponding number and pressing enter. In the command line window, there will be shown the application data.

In order to build the program, I used 5 classes: book, book1, book2, book3 and student. The program was designed such that “book” and “student” are the main classes, “book1”,” book2”, “book3”, inheriting elements from the main class “book”. For all the classes there were created a set of attributes as well as methods, such as setters, getters, constructors and additional methods. To implement different categories of books, there were used different files, one for each category of the book, and one for the students’ records. The main function is based on a switch menu, that offers the user different options, such as: issue books, return books, update student records, update book records, and the opportunity to close the program. Each option ramifies into another switch – based menu, that gives the user furthermore possibilities.

When running the application, the first step, as a librarian is to create some books for each category, and some students, in order to lend books to them. After creating these instances, the user can perform actions like lending specific books to specific students, showing on the screen a specific student or book, with all their attributes, or the entire list as well. If the user wants, he can delete books and students, in order to replace them with another entities.

**Program Structure**

The build application is structured as follows:

1. A base class called book which contains generic information about a book.

(This is the class for best – selling book category)

1. Attributes:

- char book\_number []: the number of the book

- char book\_name []: the name of the book

- char author\_name []: contains the name of the author of the book

1. Methods:

* Setters to give values to the attributes
* Getters to return and display the values of the attributes
* void modify\_book (): a function that can be used by the client by pressing one of the options from the book menu in order to modify the entered attributes of one book
* implicit constructor

1. A class called **book1** which inherits elements from **book.** (This is the class for study book category)
2. Attributes:

* char book1\_number []: the number of the book
* char book1\_name []: the name of the book
* char author1\_name []: contains the name of the author of the book

1. Methods:

* Setters to give values to the attributes
* Getters to return and display the values of the attributes
* void modify\_book1(): a function that can be used by the client by pressing one of the options from the book menu in order to modify the entered attributes of one book
* implicit constructor

1. A class called **book2** which inherits elements from **book.** (This is the class for fiction book category)
2. Attributes:

* char book2\_number []: the number of the book
* char book2\_name []: the name of the book
* char author2\_name []: contains the name of the author of the book

1. Methods:

* Setters to give values to the attributes
* Getters to return and display the values of the attributes
* void modify\_book2(): a function that can be used by the client by pressing one of the options from the book menu in order to modify the entered attributes of one book
* implicit constructor

1. A class called **book3** which inherits elements from **book.** (This is the class for comedy book category)
2. Attributes:

* char book3\_number []: the number of the book
* char book3\_name []: the name of the book
* char author3\_name []: contains the name of the author of the book

1. Methods:

* Setters to give values to the attributes
* Getters to return and display the values of the attributes
* void modify\_book3(): a function that can be used by the client by pressing one of the options from the book menu in order to modify the entered attributes of one book
* implicit constructor

1. A base class called **student** which contains generic information about a student.
2. Attributes:

* char id\_number []: the id of the student
* char student\_name []: contains the name if the student
* char stbno []: contains the number of books that a student has lend
* int token: counts how many books a student has lend (a maximum of 6 books)

1. Methods:

* Setters to give values to the attributes
* Getters to return and display the values of the attributes
* void modify\_student(): a function that can be used by the client by pressing one of the options from the student menu in order to modify the entered attributes of one student
* implicit constructor

1. By using files, there were implemented the following methods:

* void write\_book(), write\_book1(), write\_book2(), write\_book3(): functions that use the methods create\_book() from each class in order to write a book with all its attributes in a file, which we can call inventory. We will have a separated inventory for each category of the books.
* void display\_a\_book() for each category: functions that use the method show\_book() from each class in order to display on the screen a specific book, identifying it by its number.
* void delete\_book() for each category: functions that allow the user to delete a specific book from any inventory he wants
* void display\_allb() for each category: functions that display on the screen, using the method report() from all the classes, all the books from one inventory
* void issue\_book() for each category: functions that use methods from all the book classes and the student class in order to issue a book from an inventory ( that the user is choosing by the switch – based menu ) to a student, chosen by the user by its id number
* void book\_return() for each category: functions that also use methods from all the classes implemented in this application in order to return to a specific inventory a book that was lend to a student. A fine will be applied if the student returns the book in more than fifteen days.
* void write\_student() : function that uses the method create\_student() to record in the file “ student.dat “ a student entity, with all its attributes
* void display\_a\_student() : function that uses the method show\_student() in order to display a specific student, identified by its id number
* void delete\_student() : function that allows the user to delete a specific student from the file where their data are stored
* void display\_all\_students() : function that displays on the screen, using the method report() from the student class, all the records of students found in the file

**Application Demonstration**

In the image below one can observe the construction of the objects of each class. As I said earlier, the menu will show up when we press run. So, the user is free to choose any option he wants. For example, let’s choose 3.

Text

Description automatically generated

When the user enters 3 from the keyboard, another switch – based menu will show up. This part of the program is built for creating student instances. If the user wants to create a student, pressing 1 is the solution. A screenshot of a computer

Description automatically generated

Here we can see how a student is created, entering from the keyboard its id number and name. Then we press 2 to go back to the main menu.

A screenshot of a computer

Description automatically generated

Now if the user is entering the value 2, we will be able to see the list of all the students existing in the specific file, in our case only “Mazilu”.

A screenshot of a computer

Description automatically generated

We can modify a student’s name by pressing 4. Now the student is called “Neagu”.

A screenshot of a computer

Description automatically generated

If we return to the main menu, we will next have to create some books to lend them to the student(s) created. So, we press 4 to update the book records.

A screenshot of a computer

Description automatically generated with medium confidence

The list of the book categories will show on the screen. Let’s say we create one book for each category. (In this image we created a book for the study book category)

A screenshot of a computer

Description automatically generated

Now we want to display on the screen the list of the books from one category (it is the same for all the categories). We can also show a specific book from one of the inventories (again, the same for each category).

A screenshot of a computer

Description automatically generated

The modifying and the deleting part of the books is the same as the one for the students.

For the lending part of the project, the user will have now to go back to the main menu and press 1. The same options for the categories will show up, the user being able to lend any kind of book to any of the students. Here we lend a book from the best – selling category to the student with the id number 45.

Text

Description automatically generated

For returning a book the user must press the option 2. The same menu with the categories will show up. In the image below we returned a book from the best – selling category in 12 days (entered from the keyboard).

A screenshot of a computer

Description automatically generated with medium confidence

Also, if the student returns the book in more than 15 days, the program will show a message that represents the fine that will be applied to that specific student. An example of this situation is given in the image below.

Text

Description automatically generated

**Conclusion**

The current application represents a basic command line library management system able to create, delete, modify student and book instances, as well as lend and return books. To construct the program there were used 5 different classes, 4 of them found in an inheritance relation.

While a library has more than 4 categories of books, I’ve chosen to build these many for the simplicity of the program. In order to simplify the main function, the menus for each action were created in different functions before. One of the most important aspects of this project is the use of files. Each category has its own file, which we can call inventory, in which we store all the books created from keyboard. In the case of the students, we have the same method.

The command interface was designed with switch instruction-based menus, which ease the use of the program in a significant way. All the actions in the main menu represent the methods or the use of methods in combination with other instructions.

**References**

1 - <http://www.cppforschool.com/project/library-management-system.html>

2-<https://learnprogramo.com/library-management-system-project-in-c-download-with-source-code/>

3 - <https://www.geeksforgeeks.org/file-handling-c-classes/>

4 - <https://www.edureka.co/blog/file-handling-in-cpp/>

5 - <https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm>

6 - <https://m.cplusplus.com/reference/iomanip/>

**Annex – Code Files**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated