Library Management System Software Updated Development Plan

GitHub Link:

https://github.com/JoseValenciaCodes/CEN-3024C-2420C-LMS

Jose Brache Garcia

Valencia College

CEN 3024C-24204, Software Development I

Prof. Evans

January 24, 2025

Introduction

The aim of the present software development plan is to provide a reliable manual for the complete and satisfactory construction of a command-line based application that will serve to manage the list of patrons of a local library. Therefore, the present plan contains the following sections:

- **Requirements Definition:** A clear definition of the requirements the Library Management System must fulfill along with any additional constraints if there are any.
- **Requirements Gathering:** A conclusive exploration of the expectations and needs of the librarians who will use the application
- **Implementation Plan:** A step-by-step recipe required to develop the system applying Object-Oriented principles clearly displayed through the use of a UML diagram displaying all the classes and methods involved and how they interact with each other
- **Testing Plan:** A thorough testing approach to ensure the system will behave as the client needs and expects through the use of unit testing to ensure methods return the expected output and user testing to guarantee customer satisfaction.

Requirements Definition

The library requires the application to perform CRUD operations (Create, Read, Update, Delete) on patrons' data through a menu-driven interface. The specific definitions of the requirements is listed below:

- **Options Menu:** It lets the user choose an option from a list, enter the required input to perform it, and after seeing the output, the options menu will be displayed. It will also have an exit option.
- **Adding Patrons in Bulk from a Text File:** The librarian will be able to import the data of existing patrons where each line of the text represents a patron into the application. Where each line will be parsed and verified to add a new patron to the application.
- **Adding a Single Patron Manually:** The librarian will be able to add one single patron directly into the application through entering the patron's pertinent details in their respective input fields.
- **Read the details of a patron by ID:** The librarian will be able to enter the ID of a given patron to view all the details that belong to that respective patron
- **Read the details of all patrons:** The librarian will be able to see a list of all the registered patrons along with the details of each of them.

- **Update the details of a patron by ID:** The librarian will be able to enter the ID of a given patron and choose the fields to modify in order to change the data of that same patron
- **Delete a patron by ID:** The librarian will be able to enter the ID of a given patron to completely delete the details of that patron from the system.
- **Export current list of patrons to a text file:** The librarian will be able to export the list of librarians stored in the memory of the application as a text file where each line represents a patron and the details are separated by the “-” character.

Requirements Gathering

The library management system requires to be designed in such a way it is easily navigated by the user and meets user’s expectations. Therefore, the requirements previously listed need to be described by the following features to enhance customer experience:

- **UI/UX Features:** Since the application will not contain common GUI widgets like buttons, dropdowns, tabs, and search input fields. Hence, there will need to be a variation of spacing and font colors to aid the user use the application:
    - Title Spacing:
        - Centering the title of the application and space 2 lines above and 2 lines below to introduce the user to the application

- Default Color: Default font color of the console. It will be used in
    - The presentation of the application to the user
    - The input the user types
- Blue Color: It will mean getting information. Hence, it will be used in:
    - The option to get the list of all patrons
    - The option to read a patron’s detail by ID
    - The option to export patron’s data
    - The output of reading operations
- Green Color: It will mean success and/or what is new. Hence, it will be used in:
    - The option to add users in bulk from a text file
    - The option to add a user directly
    - The output of adding, updating, and deleting users
- Red Color: It will mean danger, error, and abrupt and irreversible change. Hence, it will be used in:
    - The option to exit the application
    - The option to delete a patron
    - Any error output displayed in the application

- **Data Persistence Features:** Since the application will not be connected to a database that will persist the data librarians manage as they use the application. The librarian will conveniently be able to import an old list of patrons, use the application to perform changes on that list, and then be able to export the list that includes all the new changes to the list of patrons. Hence, the name of the exported file will include the data and time when the file was exported to aid librarians track down the time patron's data was handled by the application.

Implementation Plan

This is a step-by-step guideline that specifies the technical details to build the Library Management System applying the Object-Oriented Programming paradigm through the Java programming language.

Firstly it is required to describe the classes that will be used, what their purpose will be, along with the attributes and methods they will implement with their respective inputs and outputs.

- **Main Class:** The programs starts out from this class
    - Static Methods
        - main: Java method required to start the whole program

- **AnsiColorCodes Class:** Data class meant to store strings that represent colors in the command line.
    - Attributes
        - DEFAULT_COLOR: The command's line default color
        - RED_COLOR: Red color for the command line
        - GREEN_COLOR: Green color for the command line
        - BLUE_COLOR: Blue color for the command line
- **Patron Class:** It will be used to represent a patron in the system.
    - Attributes
        - id: 7-digit string representing the ID of the patron
        - name: a string of no more than 15 characters that is the patron's name
        - address: a string used to store the patron's address
        - overdueFine: A double no longer than 250 that represents the amount of the overdue fine
    - Methods
        - Empty constructor
        - Constructor that includes all attributes
        - Constructor that includes all attributes but the id
        - Getters and setters for each attribute

        - toString in order to print a patron to the console

- **PatronDataValidator Class**: It will be used to validate the data the user inputs about patrons. It will use static methods since an instance of this class won't be required.
    - Static methods
        - isValidId: It takes a string as a parameter and returns true if it is formed only by digits and its length is 7. Otherwise, it returns false
        - isValidName: It takes a string as a parameter and returns true if it has between 1 and 15 characters inclusive. Otherwise, it returns false
        - isValidAddress: It takes a string as a parameter and returns true if the string is not empty. Otherwise, it returns false.
        - isValidOverdueFine: It takes a double as a parameter and returns true if it is not greater than 250. Otherwise, it returns false
        - validatePatron: Throws validation errors in case the given patron object contains invalid data
        - validateUpdatePatron: Throws validation errors in case changes in the data of a patron were invalid

- **OptionsMenu Class:** It will be used to display the menu-interface and let the user choose the desired option
    - Attributes
        - scanner to read user input
        - userChoice to store the menu option the user chose
        - lms to have a reference of the Library Management System
    - Methods
        - Constructor with no arguments that initialized all attributes
        - introduceLMS: Prints out text that introduces the program to the user
        - introduceMenuOptions: Prints out text that introduces the menu options to the user
        - getRegisteredPatronsDetailsOptions: Code that handles the menu option to get data of all registered patrons in the LMS
        - getPatronDetailsByIdOption: Code that handles the the menu option to get data of a specific patron by its id
        - importPatronFromExternalTextFileOptions: Code that handles the menu option to import a file with patrons data to add them to the system

- exportPatronToExternalFileOptions: Code that handles the menu option to export the data of all registered patrons in the system to an external text file
- addPatronManuallyToTheSystemOption: Code that handles the menu option to add a patron directly into the system by reading user input
- updatePatronByIdOption: Code that handles the menu option to update a patron's data by finding it first through its id
- deletePatronByIdOptions: Code that handles the menu option to delete a patron from the system by finding it first through its id
- askForPatronDetails: Code that eases the operation of presenting a user a question to then return the value of the user's response
- displayMenu: Prints the options the user could take
- getUserChoice: Will retrieve the option takes

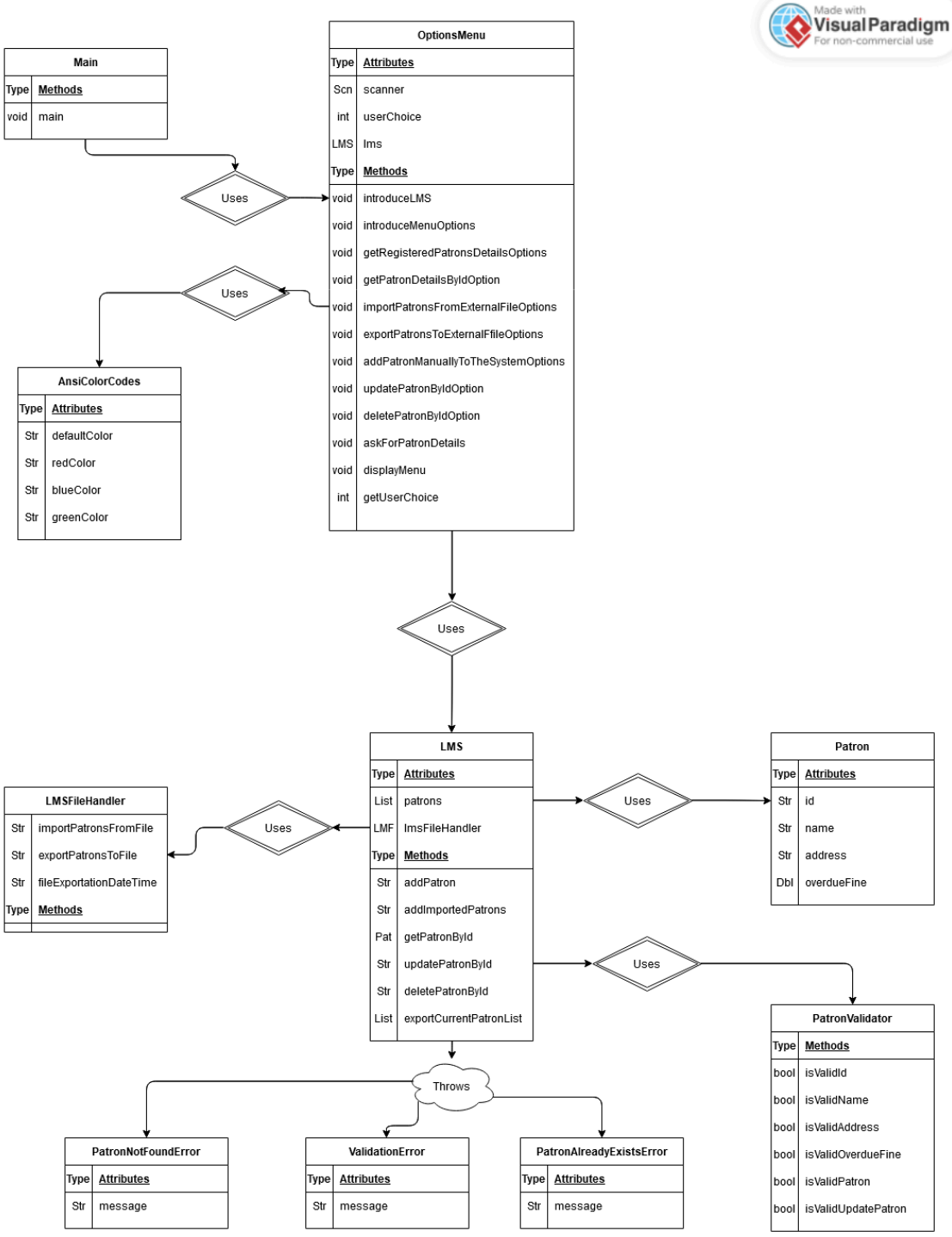
- **LMS Class:** It will be used to manage the list of patrons.
    - Attributes
        - patrons: List of patrons managed by the application
        - lmsFileHandler: Reference to the File Handler of the LMS
    - Methods:
        - Getter and setter for patrons List
        - patronExists: Gets an id and returns true if there is a patron in the list of registered patrons that has that same id.
        - addPatron: Takes a patron object and adds it to the patrons list
        - addImportedPatrons: takes a list of patrons and extends them to the existing list of patrons of the LMS class
        - getAllRegisteredPatrons: Prints the details of all patrons
        - getPatronById: Takes a string as a parameter and returns a Patron if found
        - updatePatronById: Takes a string as a parameter for the patron id along with a new Patron instance to update an existing patron if found.
        - deletePatronById: Takes a string as a parameter for the patron id and prints a success message if the patron was successfully deleted

        - exportCurrentPatronsList: It takes a string as a parameter for the file name of the file that will export the data of patrons in the system

- **LMSFileHandler Class:** It will be used to deal with I/O File Based operations:
    - Methods
        - importPatronsFromFile: It will take a string representing the name of the file that contains the list of patrons in order to process them into an array to return it
        - exportPatronsToFile: It will take a string representing the name the user wants for it and it will also take the list of current patrons in the system to create a new text file and write the data of the patrons in the system.
        - fileExportationDateTime: Return the date-time in the month-day-year-hour-minute format to then be appended to the name of the exported file

- **ValidationError Class:** Inherits from RuntimeException class, it is an error class meant to be thrown when the user entered invalid data for a patron
- **PatronNotFoundError Class:** Inherits from RuntimeException class, it is an error meant to be thrown when the user requests a patron that is not in the system
- **PatronAlreadyExistsError Class:** Inherits from RuntimeException class, it is an error meant to be thrown when the user tries to add a new user to the system that has the same ID of a user that already exists in the LMS.

UML Diagram


Made with VisualParadigm For non-commercial use
Main
Type Methods
void main
Uses
OptionsMenu
Type Attributes
Scn scanner
int userChoice
LMS lms
Type Methods
void introduceLMS
void introduceMenuOptions
void getRegisteredPatronsDetailsOptions
void getPatronDetailsByIdOption
void importPatronsFromExternalFileOptions
void exportPatronsToExternalFfileOptions
void addPatronManuallyToTheSystemOptions
void updatePatronByIdOption
void deletePatronByIdOption
void askForPatronDetails
void displayMenu
int getUserChoice
Uses
AnsiColorCodes
Type Attributes
Str defaultColor
Str redColor
Str blueColor
Str greenColor
Uses
LMS
Type Attributes
List patrons
LMF lmsFileHandler
Type Methods
Str addPatron
Str addImportedPatrons
Pat getPatronById
Str updatePatronById
Str deletePatronById
List exportCurrentPatronList
Uses
Patron
Type Attributes
Str id
Str name
Str address
Dbl overdueFine
Uses
LMSFileHandler
Str importPatronsFromFile
Str exportPatronsToFile
Str fileExportationDateTime
Type Methods
Uses
PatronValidator
Type Methods
bool isValidId
bool isValidName
bool isValidAddress
bool isValidOverdueFine
bool isValidPatron
bool isValidUpdatePatron
Throws
PatronNotFoundError
Type Attributes
Str message
ValidationError
Type Attributes
Str message
PatronAlreadyExistsError
Type Attributes
Str message

Testing Plan

To ensure the application works as expected two testing approaches will be used. For that, it will be required for key methods that deal with internal operations in the application and not with providing output to the user to return a value. This action has hereby been corrected.

- **Unit Testing:** Relevant validation methods will be individually tested to make sure they currently handle the right input and return the right output.
    - A new **UnitTests** class will be implemented in order to ensure the appropriate output is delivered
    - **PatronDataValidator Class Unit Tests**

Test Case	Input	Expected Output
Recognize valid ID	“1234567”	true
Invalid ID no digits	“nodigitId”	false
Validate Name	“Jose”	true
Empty Address Valid	“”	false
Valid Fine	145.87	true
Invalid Fine	560.34	false

- **LMSFileHandler Class Unit Tests**

Test Case	Input	Expected Output
Successful file import	fileName	Success feedback str
Erroneous file import	invalidFileName	Error feedback str
Successful file import	fileName	Success feedback str

- **LMS Class Unit Tests**

Test Case	Input	Expected Output
Check patron exists	id	true
Check for no patron	id	false
addPatron success	Valid patron data	Success feedback str
addPatron error	Invalid patron data	Error feedback str
importPatrons success	Valid file name	Success feedback str
importPatrons error	Invalid file name	Error feedback str
getPatrons success	nothing	Patrons List
getPatron success	Valid patron id	Patron data
getPatron error	Invalid Patron Id	Error feedback str
updatePatron success	Valid patron data	Success feedback str
updatePatron error	Invalid patron data	Error feedback str
deletePatron success	Valid Patron Id	Success feedback str
deletePatron error	Invalid Patron Id	Error feedback str
exportPatrons success	Valid file name	Success feedback str
exportPatrons error	Invalid file name	Error feedback str

Deployment Plan

The LMS will be deployed into one executable file that will be run using the following command:

java -jar LMS.jar

Where the command is run in the same directory where the ***LMS.jar*** executable file is present.

The ***LMS.jar*** file is a **JAR** file. In other words, a special Java file that packages all the code required to run an application into itself so that it could be easily executed by any device that has installed the Java Virtual Machine without having access to the source code.

The Library Management System will be deployed once all tests run as expected and bugs are debugged.