SOFTWARE DEVELOPEMTN PLAN

LMS Software Development Life Cycle Project Part 1

For Local Library

Date: 09/15/2025

Ver.2

Prepared For:

Professor Lisa Macon

Course Name & CRN: Software Development I - 14877

Prepared By:

Kenji Nakanishi

2729 Portchester Ct. Kissimmee, FL 34744

1. Introduction

The current Software Development Plan (SDP) encapsulates the need for a software system - Library Management System (LMS) - for our local community library librarian. This program will allow librarians to manage current patron's records efficiently. Each patron is composed of a unique 7-digit ID number and first name, last name, home address, and overdue fee amount.

This SDP will provide insight on the following in order. Requirement definition, Requirements gathering, Implementation plan, and Testing plan.

This SDP is developed under the supervision of the Valencia College Software Developer I course structure. As this is the first version of the SDP document for this project, update will follow under Valencia College Software Developer I course undermining future upgrades or modifications.

2. Requirements Definition

This software application, the Library Management System (LMS), will allow librarians to perform the following functionalities:

- Adding
    - Add patrons by manually enter details of a new patron from the console (CLI)
    - Add patrons to LMS from a text file
- Removing
    - Remove patrons from LMS by typing ID number
- Displaying
    - Display a list of patrons entered in the LMS system
    - Navigation on-screen menu for Librarian

Composition of each patron:
Unique 7-digit ID number–FirstName LastName-Address–Overdue fine amount

Example:
1245789-Sarah Jones-1136 Gorden Ave. Orlando, FL 32822-40.54
3256897-Mason Arby-6060 Saginaw St. Casselberry, FL 34852-0
4567891-Avery Jones-1919 Pine Lance Blvd. Oviedo, FL 32478-1.36

Important highlight:

- Unique ID number: 7-digits only number
- Name: Composed by first and last name, first letter capitalized on both names
- Address: Full Address including state and zip code, capital every word first letter
- Overdue fine amount: No dollar sign needed and 2 decimal places ranging $0.00 ~ $250.00. Double type
- Each item is followed by a dash – no space before and after.

The program will have the following constraints & limits as this version is developed:

- The program will be developed to be used on a console-based interface, meaning no Graphic User Interface (GUI) will be needed at this time.
- Data will be stored within the application in memory only location meaning there is no need for a database environment or relational database.
- The program will be developed using Java programming language and tested on Windows 11 Operative System.

3. Requirements Gathering

The story that the Lead Librarian shared is that the local library has grown from only 15 members to over 200 members. Writing down on paper all the members' data has led to missing or incomplete information, extra time spent on localizing or updating records, paper waste, and more importantly, tracking down fees and being aware of members who owe money back to the library system has become a larger task to organize within.

The Lead Librarian said that understanding active members and address is an essential need. By knowing those details, they will be able to send promotional details, events, and even mail reminders of overdue to avoid late fees.

The librarian needs a simple easy console-based LMS that can store members data (patrons) including patron data (ID, name, address, and overdue fee).

Users indicated the following are the key expectations that the program should have and be able to perform:

- An easy to navigate menu, choosing a number select actions to perform.
- Displaying a warning message when data entered is duplicated or invalid
- [1] Entering manually a new patron by input of the sequence of: ID, FirstName, LastName, Address, and Overdue fine amount. *
- [2] Adding patrons by loading a saved .txt file from the user computer. *
- [3] An Option to remove patrons by typing ID number. *
- [4] List of all the patrons stored. *

*Numbers are going to be the options users can choose to access those options.

3.1 Additional considerations:

For developing this program user also shared that the software is expected to be in English followed by the specific format indicted in the Requirements Definition section number 2.

It also needs to be reliable and be able to perform operation when needed without unexpected failures. Customer indicated that this LMS support can handle at least 10,000+ patrons storage capacity.

4. Implementation Plan

The coding language to be used will be Java JDK 17+ since it is supported by Oracle until 2029 ensuring long-term stable performance.

This program will be composed of three main classes:

Patron: Contains the formatting of the patrons
NavigationMenu: Prints the console-based interface and holds the methods to perform operations.
LibraryApp: Main App that launches the program.

4.1 Patron Class

4.1.1 Public Patron Class:

Patron will include private field String uniqueID (7-digits), private String firstName, private String lastName, private String address, and private field double type fee (range 0.00 - 250.00).

All private fields to prevent accidental overwriting from other classes rules.

4.1.2 Public Patron Constructor:

Includes uniqueID, firstName, lastName, address, and fee as its arguments. Since there is no need to modify an input patron, we will not use setters.

4.1.3 Patron - Getters:

To access the private fields, add getters for all the fields previously indicated. This will allow access to read.

4.1.4 Patron – ToString Method:

Since librarian needs a solution where one of the required features is "List of all the patrons stored," the Patron class must Include an @Override method called toString() to print Patron's values as per librarian requested.

Expected output format:

uniqueID-fistName lastName-address-fee

Follow format:

```
public String toString() {
return String.format("%s-%s %s-%s-%.2f",
uniqueID, firstName, lastName, address, fee);
 }
```

Important highlight:

- Please be advised that dash lines (-) separators will be used between each value.
- FirstName and lastName are separated by a single space.
- There are no extra spaces before or after dash lines.

Example output:
1245789-Sarah Jones-1136 Gorden Ave. Orlando, FL 32822-40.54
3256897-Mason Arby-6060 Saginaw St. Casselberry, FL 34852-0
4567891-Avery Jones-1919 Pine Lance Blvd. Oviedo, FL 32478-1.36

4.2 NavigationMenu Class

The customer desires to have easy to follow simple console-based menu. The librarian can select an option by typing the corresponding selection as following:

```
**************************************
Local Library Patrons Data
**************************************
[1] Add patron manually      [4] List all patrons
[2] Load patrons from file   [5] Exit program
[3] Remove patrons by ID

Enter your selection: _
```

Important highlight:

- Add Menu Title as Local Library Patrong Data with asterisk rows to separate from selection.
- Options have been split into two columns for readability.
- Prompted user to “Enter your selection” for guidance.
- Added a loop until user selects Exit program option.

4.2.1 Libraries

The Scanner utility class from java.util. is imported to read librarian input from the console. Java.io and java.nio.file are imported as well to support reading and writing patrons to text files.

4.2.2 Public Class NavigationMenu

In the NavigationMenu class will be reading user input and performing the selection entered by the librarian (add patron, load from file, remove by ID, List patrons, and exit).

Important highlight:

- Allow librarian input from the console by using Scanner in
- We can store patrons’ records in a text file and backup data

4.2.4 Public Class NavigationMenu Constructor

We will include the initialization of in-memory data store by adding a HashMap to hold patrons records. Also, we will prepare Scanner for librarian input.

4.2.5 Public Class NavigationMenu Methods

Since in this phase the project does not require a code yet, a basic overview of the methods is as follows:

Start(): Return Type: Void

Algorithm: This method will initialize the main loop that displays the NavigationMenu. Expect the librarian for a selection and call the needed methods for each case (AddingManually, RemovingByID, DisplayList), it will terminate the process by choosing exit.

DisplayNavigation(): Return Type: Void

Algorithm: Print the outline provided on 4.2 Navigation Menu section, and prompt librarian “Enter your selection:”

AddingManually(): Return Type: Void

Algorithm: Prompt librarian to enter data in sequential order: uniqueID, firstName, lastName, address, and fee.

Additionally, for uniqueID and fee, this method will be supported by ID_Fomat() and Fee_Format() which will verify the formatting is compliance with the customer needs declared on 4.1.1 Public Patron Class. If it is not valid format, it will display an error message.

AddingFromTxtFile(): Return Type: Void

Algorithm: Allow librarian to add multiple patron data by loading a text file. It will ask librarian for text file path and read each line of the file that will represent one patron separated by dash as per user request. It will use ID_Format() & Fee_Format() to verify formatting and duplicate IDs or invalid entry will be skipped.

RemovingByID(): Return Type: Void

Algorithm: Prompt librarians to input a patron ID. Search patron list and compare it to the input value. If patron ID exists in the list, found patron will be removed and confirmation screen will be displayed, if there was not a match a message displaying ID not found will be displayed.

DisplayList(): Return Type: Void

Algorithm: Displays all the stored patrons in ascending order by ID. It will be displayed following desired customer format defined in 4.1 Patron Class. If there is none patron stored, it will display no data entered.

LoadFromSavedFile(): Return Type: Void

Algorithm: When program starts up, it will verify if file LocalLibraryPatronsDataSave.txt. exists, if so, the method LoadFromSavedFile will load patrons previously saved to the text file LocalLibraryPatronsDataSave.txt. and will display a message “patrons were load from LocalLibraryPatronsDataSave.txt. file”.

Save(): Return Type: Void

Algorithm: Auto-save method that will ensure all data is recorded to LovalLibraryPatronDataSave.txt. whenever librarian press [5] Exit Program by ordering in ascending ID and confirm saved before exit.

4.2.6 Public Class NavigationMenu Verifying Correct input Methods

In addition to the methods that perform operations, we will check if the typed information from the librarian is correct on uniqueID & fee variables.

ID_Format(): Return Type: Boolean

Algorithm: Read user input String and verify that uniqueID is exactly as per user request seven numerical digits with non-symbols or alphabetic characters. If it is not correct, will ask user to input correct data.

Fee_Format(): Return Type: Double

Algorithm: Verify that the overdue fee is numeric value and within the acceptable user provided range 0.00 – 250.00. If correct store it and if not ask user to input correct data.

Highlights:

- This approach includes all the necessary methods in one class NavigationMenu, making it easier to organize and verify the code later.
- With the validation methods ID_format & Fee_Format, we make sure that the data format is correct and can be accepted.
- Although saving previously recorded patrons data was not explicitly required in the Requirements Gathering portion, this program includes this feature so that the librarian can load patrons that were already recorded in LocalLibraryPatronsDataSave.txt.

4.3 LibraryApp

4.3.1 Public Class LibraryApp Main Method

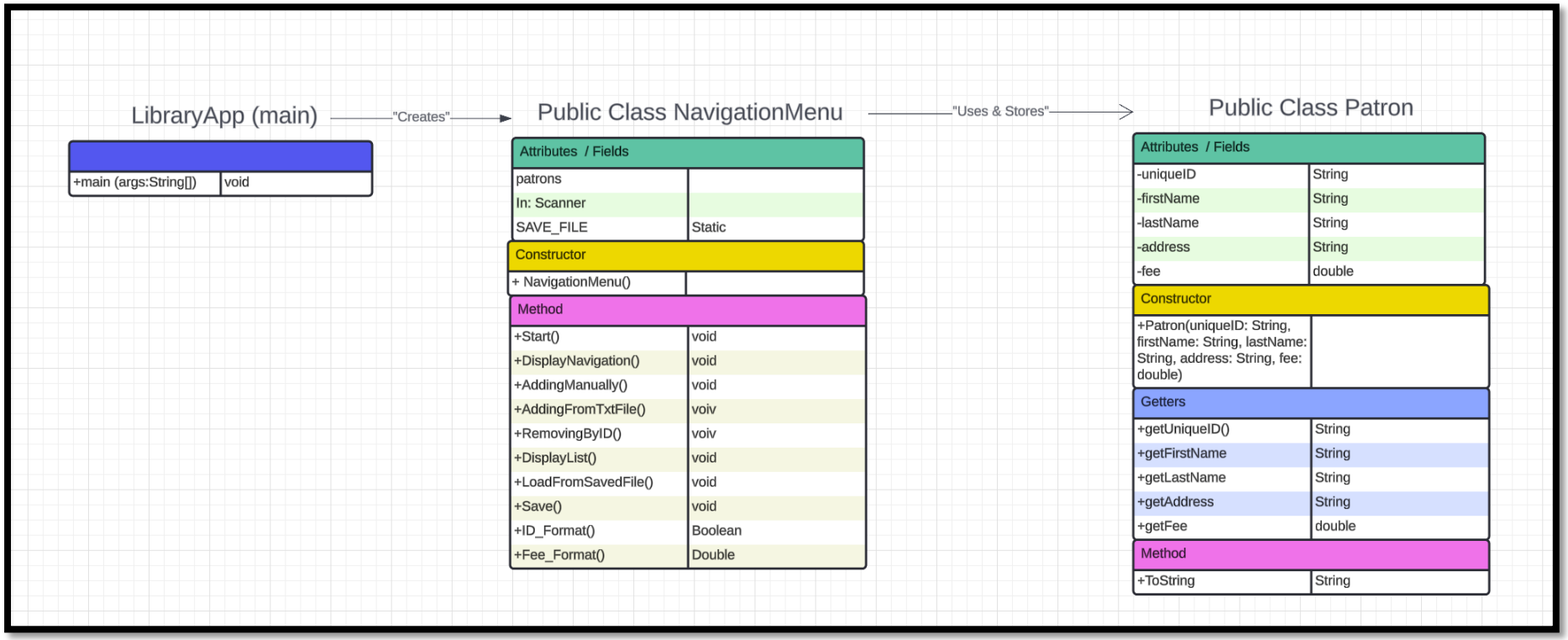
We must include the app that will initialize the console application. With this method main the NavigationMenu will be launched.

Highlight:

A simple structure for this part of the project since all operations are being performed within NavigationMenu Class methods.

4.4 UML Diagram:

Below is the Unified Modeling Language (UML) diagram including the general structure and relationships.



Please find the direct link of the UML diagram, this diagram has been created utilizing lucid app and it is currently public for general visualization:

https://lucid.app/lucidchart/9e0d92cf-8ce7-4487-a72c-19241a898a5b/edit?viewport_loc=2908%2C-1484%2C2319%2C2022%2C0_0&invitationId=inv_33992c71-3ffd-4e13-9c20-d2baf1dfb6a7

5. Testing Plan

To ensure our Local Library Management System (LMS) works as per customer needs, I created a testing plan including test case ID, name, steps, input, result, actual result, status, and comments. Specifically, I found the GeeksforGeeks article on software testing very detailed and helpful to shape the structure of the Local Library App.

Test Name		Local Library App Alpha Test
Testing Plan Description		Verify functionality of the Local Library Program
Prerequisites		Console environment
Tester's Name		Kenji Nakanishi
Enviromental Information		Operative System: Windows | System: Desktop Laptop
Test Scenario		Verify after input data the user can perform desired operation

Test Case ID	Test Name	Test Steps	Test Input	Expected Result	Actual Result	Status	Comments
For Method AddingManually()							
1	AddingManually() Valid Input	Select [1] Add patron Manually.	7654321-Kenji Nakanishi-48 Old Morton St. FL 32837-25.00	Patron added!			
2	AddingManually() Invalid Input	Select [1] Add patron Manually.	0000-Kenji Nakanishi-48 Old Morton St. FL-32837-25.00	Error: Verify digits.			
3	AddingManually() Invalid Fee	Select [1] Add patron Manually.	1111111-Kenji Nakanishi-48 Old Morton St. FL 32837-999.99	Error: Verify fee.			
For Method AddingFromTxtFile()							
4	AddingFromTxtFile() Valid File	Select [2] Load patrons from file	File with correct formatting patrons.	Patrons added!			
5	AddingFromTxtFile() Invalid File	Select [2] Load patrons from file	File with incorrect formatting patrons.	Error: File entry			
5	AddingFromTxtFile() Duplicate IDs	Select [2] Load patrons from file	File with duplicate IDs	Skipped Duplicate entries			
For Method RemovingByID()							
7	RemovingByID() Existing ID	Select [3] Remove patrons by ID	7654321	Removed!			
8	RemovingByID() No found ID	Select [3] Remove patrons by ID	2222222	Errir: ID not found			
For Method DisplayList()							
9	DisplayList() Contain Patrons	Select [4] List all patrons	Already +1 patrons stored	Ascending order IDs			
10	DisplayList() No Data	Select [4] List all patrons	No previous data stored	Empty!			
For Method Save()							
11	Save() Auto-Saving	Select [5] Exit program	Add Patrons previously, Exit the program and open it again and select [4] List all patrons	Patrons were load from LocalLibraryPatronsDataSave.txt. file			

Source:

"Software Testing Test Case." GeeksforGeeks, 10 Jan.2024, https://www.geeksforgeeks.org/software-testing/test-case/

6. Deployment

Including the source code of the project:

6.1 LibraryApp.java

```
/*
Project: LMS Software Development Life Cycle Project Part 2
Name: Kenji Nakanishi
Course and CRN: Software Development I & 14877
Professor: Lisa Macon

This is the Main App where the application is going to run the NavigatonMenu containing the
outline of the system CLI where the librarian can select what actions to perform.
*/
public class LibraryApp {
   public static void main(String[] args) {
      new NavigationMenu().start();
   }
}
```

6.2 Patron.java

```
/*Within Patron Class we will find variables that will compose each patron
its getters, and the ToString method where we will specify the format of the desired formatting" */
public class Patron{
   // Variable Declaration:
   private final String uniqueID;
   private final String firstName;
   private final String lastName;
   private final String address;
   private final double fee;
// Constructor
public Patron (String uniqueID, String firstName, String lastName, String address, double fee) {
  this.uniqueID = uniqueID;
  this.firstName = firstName;
  this.lastName = lastName;
  this.address = address;
  this.fee = fee;
}
// Getters
public String getUniqueID(){    return uniqueID;}
public String getFirstName(){    return firstName;}
public String getLastName(){    return lastName;}
public String getAddress(){    return address;}
public double getFee(){    return fee;}
   // Override method with the customer
   @Override
   public String toString(){
      return String.format("%s-%s %s-%s-%.2f",
      uniqueID,firstName,lastName,address,fee);
   }
   // toFileLine method  is the used format when Save() method is writing lines into the txt. file
   public String toFileLine(){
      return uniqueID + "-" + firstName + " " + lastName + "-" + address + "-" + String.format("%.2f", fee);
   }
}
```

6.3 NavigationMenu.java

```
// NavigationMenu will include all the needed Methods from path to file, display patron, add patron
// remove by ID of the member, List and display them, finally exit which will save the patron data on the
selected path.
// The program works on user input demand, meaning if the user select their own txt file, it will base on
their patron data.
// I also have included a resource file with LocalLibraryPatronDataSave.txt where I have previously
stored some patrons for user testing
// The program will save, update, and display the patron data whenever add, remove, and before exit
the program.

import java.nio.file.*; // For paths and file reading
import java.util.*; // Wildcard including Collections, List, Arrays,
import java.io.*; // Including BufferedReader, IOException

public class NavigationMenu {
   private Path ChooseTxtFile;
   private final HashMap<String, Patron> patrons = new HashMap<String,Patron>();
   private final Scanner in = new Scanner(System.in);

   public void start() {
      //It will print only 1 time to the user that a resource of sample patron data has been included and
the
      //librarian can just copy the path to access the resource or type their own absolute path.
      System.out.println("""
         ********************************************************************************
                        Local Library Patrons Data
         ********************************************************************************
         |You can also select [2] and copy path:                              |
         |                                                |
         |FROM JAR: LocalLibraryPatronsDataSave.txt                           |
         |FROM IntelliJ: out\\artifacts\\Patrons_jar\\LocalLibraryPatronsDataSave.txt    |
         |                                                |
         |where a sample data is stored!  Again, No needed if you have your own file! |""");


      // While loop with case accessing each method:
      // I have added some functionalities inside the methods to make the console app
      // A little more interactive.
      while (true) {
         DisplayNavigation();
         String selection = in.nextLine().trim();
```

```
        switch (selection) {
            case "1" -> AddingManually();
            case "2" -> AddingFromTxtFile();
            case "3" -> RemovingByID();
            case "4" -> DisplayList();
            case "5" -> {
                Save();
                System.out.println("Exiting Local Library LMS Patrons Data Program... Good Bye!!!");
                return;
            }
            default -> System.out.println("Error: Invalid input...");
        }
        System.out.println();
    }
  }
  //Navigation Menu is displayed using [#] to easily show user that the numbers are going to be used for
selection
  public void DisplayNavigation() {
    System.out.println("""

        ****************************************
            Local Library Patrons Data
        ****************************************

        [1] Add patron manually     [4] List all patrons
        [2] Load patrons from file  [5] Exit program
        [3] Remove patron by ID

        Selection:  """);
  }

  //Methods

  // AddingManually, it allows user to add the patron by typing directly to the console based app.
  // It also allows to type all the information without stopping every steps on meaning it will let you
input
  // all the data, and if error found, it will explain where was those error and will not record the patron
entered.
  public void AddingManually() {
    //  Print out a detailed instruction on how the system will prompt librarian's input
        System.out.println("\nINSTRUCTIONS: \n" +
        "You will be prompted to enter patron data in the following sequence:\n\n " +
        "\n1.ID " +
        "\n2.First Name" +
```

```
            "\n3.Last Name" +
            "\n4.Address" +
            "\n5.Overdue Fee\n");

        // Ask for UniqueID, indicating the acceptable format with numbers and nor characters nor
symbols
            System.out.print("Accepted format: Only 7 digit numbers from 0 ~ 9, not letters nor symbol." +
            "\nPlease, type 7-digit unique member ID: ");
            String seven_digits_ID = in.nextLine().trim();

        // Ask for First Name, highlight that is only first name
            System.out.print("Please, type only member's First Name: ");
            String first = in.nextLine().trim();
        // Ask for Last Name, highlight that is only last name
            System.out.print("Following, type member Last Name: ");
            String last = in.nextLine().trim();
        // Ask for Full Address
            System.out.print("Next, Enter Address. (Example: 1234 Squaretown St. Orlando, FL 32890) " +
                        "\nFull Home Address:");
            String address = in.nextLine().trim();
        // Ask for Fee amount within the accepted range which is printed to the user
            System.out.print("Enter overdue fee amount (0.00 ~ 250.00): ");
            String EnteredFee = in.nextLine().trim();

    // ErrorInEntry will verify either the input was correct or has some error
    // If Error found. it will print at the end once user entered all the fields.
    // Indicating in which specific items was the error found.
    boolean ErrorInEntry = false;

        // No detecting 7 digits will set ErrorInEntry to True | later it will be assigned under IDError
        if (!ID_Format(seven_digits_ID)) {
            System.out.println("\n*** ERROR: Verify digits. Unique ID must be exactly 7 digits. ***");
            ErrorInEntry = true;
            // Having a duplicate will set ErrorInEntry to True | later it will be assigned under IDError
        } else if (patrons.containsKey(seven_digits_ID)) {
            System.out.println("\n*** DUPLICATE: This patron ID already exist. Patron was not added to
the records. ***");
            ErrorInEntry = true;
        }
        // Having empty first, last or & address it will set ErrorInEntry to True | | later it will be assigned
under FirstNameError, LastNameError, or AddressError
        if (first.isEmpty() || last.isEmpty() || address.isEmpty() ) {
            System.out.println("\n*** ERROR: Fist Name, Last Name, and Address cannot be blank ***");
            ErrorInEntry = true;
```

```
        }
        // If there is not amount entered or not within range it will set ErrorInEntry to True | later it will
be assigned under FeeError
        Double fee = Fee_Format(EnteredFee);
        if (fee == null) {
            System.out.println("\n*** ERROR: Verify Fee amount. Fee must be a number in between [0.00
~ 250.00] ***");
            ErrorInEntry = true;
        }

    // Setting the conditions for each boolean
    boolean IDError = !ID_Format(seven_digits_ID) || patrons.containsKey(seven_digits_ID);
    boolean FirstNameError = first.isEmpty();
    boolean LastNameError = last.isEmpty();
    boolean AddressError = address.isEmpty();
    boolean FeeError = (fee == null);

    // It will print the librarian typed content and add ***** where error was found, if not ***** will
not be added.
    if (ErrorInEntry) {
      System.out.println("\n PATRON NOT ADDED - Input with error has been marked with *****");
      System.out.println("\nPatron ID: " + seven_digits_ID + (IDError ? " *****" : " "));
      System.out.println("First Name: " + first + (FirstNameError ? " *****" : " "));
      System.out.println("Last Name: " + last + (LastNameError ? " *****" : " "));
      System.out.println("Address: " + address + (AddressError ? " *****" : " "));
      System.out.println("Fee: " + EnteredFee + (FeeError ? " *****" : " "));
       return;
    }

    // HashMap put with Key seven_digits_ID to identify user ID and then value of all the entry data
    // to record into Patron HashMap if everything entered by the user was right!
        patrons.put(seven_digits_ID, new Patron(seven_digits_ID, first, last, address, fee));
        Save();

        // Print in the user screen the loading animation that the data is being recorded set
thread.sleep to 750 for realistic experience
         System.out.println("Adding new patron");

        for (int i = 0; i < 3; i++) {
          try {
            Thread.sleep(750); //three / quarter of a second
          } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
          }
```

```
            System.out.print("..");
        }
        // Inform librarian that the patron was added successfully and prints in the screen the last
entered patron.
        System.out.println("\nPatron has been added successfully!.");
        System.out.println("INFORMATION JUST ADDED: " +
        "\nMember ID: " + seven_digits_ID +
        "\nFirst Name: "  + first +
        "\nLast Name: " + last +
        "\nAddress: " + address +
        "\nOverdue Fee: " + String.format("%.2f", fee) + "\n\n");

        // Finally, also prints all the data stored so far, so librarian do not have to go [4] List all patrons
again!
          System.out.println("Patrons list in Ascending ID Order: \n");
          List<String> patrones1 = new ArrayList<>(patrons.keySet());
          Collections.sort(patrones1);
          for (String PatronMember : patrones1) {
          System.out.println(patrons.get(PatronMember)); // uses Patron.toString()
    }
  }

  // Validation methods: To make sure the format is correct and can be accepted ===
  // ID_Format Method: It will check that what the librarian has types is neither null and have 7 digits.
  public boolean ID_Format(String seven_digits_ID) {
    return seven_digits_ID != null && seven_digits_ID.matches("\\d{7}"); // exactly 7 digits number
String is required!
  }

  // Fee_Format Method: This method will verify the user input by checking the accepted range which is
0.00 < value < 250.00
    public Double Fee_Format(String EnteredFee) {
    try {
      // It converts in double format
      double value = Double.parseDouble(EnteredFee);
      if (value < 0.00 || value > 250.00) return null;
      return value;
      // It will also return null printing the Error message to the user
    } catch (Exception e) {
      return null;
    }
  }

  //AddingFromTxtFile Method: For this program, the user can choose either using my sample file
```

```
stored under
   // resources titled LocalLibraryPatronsDataSave.txt so they can just copy the path. Another option is
to
   // select the absolute path and the instruction portion will explain how to enter the txt file path.
    public void AddingFromTxtFile() {
      System.out.print("INSTRUCTIONS: You can type your own file path\n" +
                  "\nSample data path: " +
                  "\n|FROM JAR: LocalLibraryPatronsDataSave.txt                              |"+
                  "\n|FROM IntelliJ: out\\artifacts\\Patrons_jar\\LocalLibraryPatronsDataSave.txt    |"+
                 "\n|Enter file path: ");
      String TxtPath = in.nextLine().trim();
      // If librarian just copied the path including " or ' and pasted to the console.
      // This will remove those "" '' making them able to be read.
      if ( (TxtPath.startsWith("\"") && TxtPath.endsWith("\"")) ||(TxtPath.startsWith("'") &&
TxtPath.endsWith("'")) ) {
         TxtPath = TxtPath.substring(1, TxtPath.length()-1).trim();
       }

      // In case an exception has been caught it will print depending on the case the message to the user.
      Path path;
            try {
              path = Paths.get(TxtPath);
            } catch (Exception e) {
               System.out.println("*** ERROR: Invalid file path ***");
               return;
            }
            if(!Files.exists(path)) {
               System.out.println("*** ERROR: File doesn't exist. ***");
               return;
            }

     // For interface and interactivity:
     // Adding a counter indicator for line order, added values, duplicates, error and skipped values has
been introduced.
     int lineInTxt = 0, SuccessfullyAdded = 0, DuplicatesValue = 0, Errors = 0, Skipped = 0;
      try (BufferedReader reader = Files.newBufferedReader(Paths.get(TxtPath))) {
         String line;
         while ((line = reader.readLine()) != null) {
            lineInTxt++;
            line = line.trim();
            if (line.isEmpty()) continue;
            // Creating an array to store values separated by Unique ID, FullName, Address, and Fee.
            // User will type naturally in their Text file as following:
            // 1231231-Kenji Nakanishi-48 Old Morton St. Orlando FL. 32837-00.02
```

```
            // Notice how first and last name goes together without a dash line.
            // This part will separate whenever a dash is found.
            String[] info = line.split("-", 4);
            // Whenever finds a line that does not contain all 4 records will be skipped and count towards
the skip count.
            if (info.length != 4) {
               System.out.println("line number: " + lineInTxt + ": has been skipped.");
               Skipped++;
               continue;
            }
            //Important: Dividing in 4 parts:
            String Unique_ID       = info[0].trim(); // before 1st dash will hold Unique Patron ID
            String FullName        = info[1].trim(); // This part will hold first name and last name
            String Address         = info[2].trim(); // Address will be held in the third section
            String FeeInString     = info[3].trim(); // fee information in String format

            // I used the regular expression to find a white space character and when find it separate them,
            // Whenever found a whitespace, the program understand that the first part is the first name
            // the second part must be the last name
            String[] PatronsNameParts = FullName.split("\\s+");
            // if there is less than two names or more, != 2 will detect as error
            if (PatronsNameParts.length != 2) {
               System.out.println("line number: " + lineInTxt + " | ERROR: Naming format incorrect: " +
FullName);
               Errors++;
               continue;
            }
            String FirstName = PatronsNameParts[0]; // understanding the string before whitespace is
firstname
            String LastName  = PatronsNameParts[1]; // understanding the string after whitespace is
lastname
            // Whenever does not match with ID_Format that has a regex of 7 number format
            if (!ID_Format(Unique_ID)) {
               System.out.println("line number: " + lineInTxt + " | ERROR: Verify digits typed: " +
Unique_ID);
               Errors++;
               continue;
            }
            // Whenever the fee format is not within range as specified in Fee_Format
            Double fee = Fee_Format(FeeInString);
            if (fee == null) {
               System.out.println("line number: " + lineInTxt + " | ERROR:Verify fee typed: " + FeeInString);
               Errors++;
               continue;
```

```
            }
            // Whenever the Unique_ID has been previously entered
            if (patrons.containsKey(Unique_ID)) {
               System.out.println("line number: " + lineInTxt + " Skipped since DUPLICATE entries. " +
Unique_ID);
               DuplicatesValue++;
               continue;
            }
            // If everything goes well, those patrons passed the test will be added successfully!
            patrons.put(Unique_ID, new Patron(Unique_ID, FirstName, LastName, Address, fee));
            SuccessfullyAdded++;
         }
      } catch (IOException i) {
         System.out.println("I/O error: " + i.getMessage());
         return;
      }
      // This will storage the path so when hit exit it will be saved to the chosen path.
      this.ChooseTxtFile = path;
      // Print in the user screen the loading animation that the data is being loaded to add interactivity
same 750 milliseconds
       System.out.println("Loading patrons from Text file.");
       for (int i = 0; i < 3; i++) {
          try {
             Thread.sleep(750); //three / quarter of a second
          } catch (InterruptedException e) {
             Thread.currentThread().interrupt();
          }
          System.out.print("..");
       }
       // Print to the librarian what has been counted!
      System.out.printf("\nDone! \nPatrons that were added: %d, duplicates: %d, errors: %d,
Skipped: %d",
            SuccessfullyAdded, DuplicatesValue, Errors, Skipped);
   }

   // RemovingByID method: This method will  evaluate if the list is empty first.
   // If any ID found, it will print that any ID was not found.
   // If there are Patron IDs stored previously, the Librarian will be asked to check if the entered ID
information matches in name and addres
   //  User will be prompted to select by Y and N Y = to erase, N = to cancel action
   // If user enter another option out of y and n, the while loop will keep printing invalid entry
   public void RemovingByID() {
      if (patrons.isEmpty()) {
         System.out.println("*** ERROR: Patrons list is EMPTY!. ***");
```

```
            return;
        }
        // Prompt to enter the Patron ID to remove
        System.out.print("Enter 7-digit Patron ID to remove: ");
        String seven_digits_ID = in.nextLine().trim();
        // This will check if any patrons were previously entered
        Patron PID = patrons.get(seven_digits_ID);
        if (PID == null) {
            System.out.println("*** ERROR: Patron ID not found!. ***");
            return;
        }
        // This will show the patron information to ask the librarian kind of are you sure this is the data you
want to erase:
        System.out.println("It correspond to: \n" + PID);
        // Start a while loop to ask for user selection Y or N
        while (true) {
            System.out.println("Please confirm you want to remove this patron? (y/n); ");
            String confirmation = in.nextLine().trim();
            // for Yes = y and Y
            if (confirmation.equals("y") || confirmation.equals("Y")) {
                patrons.remove(seven_digits_ID);
                Save();
                System.out.println("Patrons have been removed successfully!\n\n");
                break;
                // for No = n and N
            } else if (confirmation.equals("n") || confirmation.equals("N")) {
                System.out.println("Cancelled");
                break;
                // in case other character was entered loop again.
            } else {
                System.out.println("Invalid, please enter Y for yes & N for no");
            }
        }
        // Print the patrons list after deleted so the librarian does not have to go [4] List all patron again.
        System.out.println("Patrons list in Ascending ID Order: \n");
        List<String> patrones2 = new ArrayList<>(patrons.keySet());
        Collections.sort(patrones2);
        for (String PatronMemberID : patrones2) {
            System.out.println(patrons.get(PatronMemberID)); // uses Patron.toString()
        }
    }

    // Method that will display the list of the patrons added so far
    // if the list is empty it will print to the user that there is none recorded.
```

```
    public void DisplayList() {
        if (patrons.isEmpty()) {
            System.out.println("No patrons stored to display.");
            return;
        }
        List<String> patrones = new ArrayList<>(patrons.keySet());
        // Print the list in Ascending order of ID:
        System.out.println("Patrons list in Ascending ID Order: \n");
        Collections.sort(patrones);
        for (String seven_digits_ID : patrones) {
            System.out.println(patrons.get(seven_digits_ID)); // uses Patron.toString()
        }
    }
    // Save method, it will inform the librarian first that the save path has not been selected yet.. and
encourage to specify one.
    // Basically, It does save after ManuallyAdding, RemoveByID, and when Exiting the program.
    public void Save() {
        if(ChooseTxtFile == null) {
            System.out.println("Patron DATA SAVED; However no file path has been specified to save it yet!!!
\n If you want to save this patron data please choose a file path using option [2] AddingFromTxtFile\n"+
                    "After adding the file path, this data will be stored as well!\n");
            System.out.println("Saving in temporary memory...");
            return;
        }
        List<String> ids = new ArrayList<>(patrons.keySet());
        Collections.sort(ids);
        // It uses ChooseTxtFile
        try (BufferedWriter bw = Files.newBufferedWriter(ChooseTxtFile)) {
            for (String id : ids) {
                bw.write(patrons.get(id).toFileLine());
                bw.newLine();
            }
        } catch (IOException ioe) {
            System.out.println("Error: Failed to save patrons: " + ioe.getMessage());
        }
    }
}
```