Software Development Plan for Simple Library Management System

Marla Peraza Ravelo

CEN-3024C-24204 Software Development I

Professor Ashley Evans

January 19, 2025

**Introduction**

The objective of this project is to apply the knowledge acquired in class about the Software Development Life Cycle (SDLC) by creating a simple program that fulfills the needs of a fictional client. The client in this case is a local library that is in need of a new software system to help the librarians add, remove, and query all the patrons easily. This document will outline the process of defining and gathering the requirements from the client, the most important parts about the implementation of the project, and the testing methodology that will be used once the application is created to ensure that the client is satisfied with the final product.

**Requirements Gathering**

As mentioned above, the client is a local library who needs a new software system. According to them, they need a console-based application that will manage the library’s list of current patrons. A librarian must be able to input new patrons either from information inside a file or manually. Each patron in the system has a unique 7-digit ID number. Each patron also has a name, address, and overdue fine amount, ranging anywhere from $0-$250. The librarian must also be able to see the current patrons and remove them by their ID number. The customer asked for a menu-like screen that shows the different options to interact with the patrons. No database is needed to store information about the patrons.

The user's input text file needs to be formatted to a certain standard. Each line in the file is going to represent a patron. Each element (id, name, address, and current overdue fine amount) is separated by a dash. There will be no dollar sign shown with the overdue amount.

The examples below illustrate how the patrons need to be formatted inside an input text file for the application to be able to read each one.   
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

**Requirements Definition**

After gathering information from the customer, a list of deliverables has been created:

1. An interactive console-based application to manage the adding and removing of patrons.
2. A method within the application to add patrons by reading the formatted contents of a file.
3. A method within the application to add patrons manually by entering their details (id, name, address, and overdue fine).
4. A method within the application to remove patrons from the Library Management System using their ID number.
5. A method within the application to display all patrons currently in the Library Management System.
6. An interactive menu on-screen to show the users the different options to manipulate patrons.

This application will not include connection to a database, and all the information about the patrons will be stored inside a list of objects.

**Implementation Plan**

A screenshot of a computer

Description automatically generated

The image above represents the two classes that are going to be created in order to implement the program. The first class is Patron, which represents the patrons that are going to be added and removed, and has four private attributes: id (String), name(String), address(String), and overdueFine(double). This class also has three methods: the constructor method Patrons(), which takes in the different parameters to build a Patron object, the getter getId(), which returns the id of the Patron object, and the overridden method toString(), which is going to print all the contents of each Patron object when it is called.

The next class is LibraryManagementSystem, which is going to contain all the necessary methods to add, remove and display the patrons (i.e. perform different actions with the list of Patron objects). It contains one private attribute, patrons, which is a list of objects of type Patron.

There are two methods inside this class that allow the user to enter the details for a new Patron. The first is the method newPatronFromFile(), which allows the user to enter the details from an already existing file. It is going to take a String filepath representing the path of the desired file, and it is going to read that file and add the new Patron object to the list.

The newPatronManually() method enables the user to enter the Patron’s details manually, by prompting him or her to enter each field value at a time. It taks a Scanner object, which is going to be the Scanner used to get user input.

The removePatronById () method takes in a string id, and then it removes the Patron that contains that id from the list.

The method displayPatrons() prints all the Patrons inside the list, showcasing all their details.

The method displayMenu() is going to print a menu interface for the user, which is going to contain the different actions the user can perform, as well as the title of the program. Each action is going to be enumerated, and the user is going to be prompted to enter a number to choose an action. A tentative view of the menu follows:

Library Management System

1. Add patrons from a file

2. Add a patron manually

3. Remove a patron using Id number

4. Display all Patrons

5. Exit

The implementation of the main method is going to be as follows:

The title of the program (Library Management System), along with the current patrons will be displayed by the use of the displayPatrons() function. Since there are no current patrons at the start of the program, a message informing the user that the list is empty will be displayed instead. This message will be updated once the user enters new patrons in the system.

Below it, the LMS menu containing all the options will be displayed for the user to make a choice.

The preceding and following code will be surrounded by a do-while loop that will continue to run as the user for an input until the user chooses number 5 (Exit), in which case the program will stop.

Inside the do-while loop, a switch statement is going to run which will evaluate the input given by the user. If the user inputs a 1, he or she is going to be prompted to enter the path of the file that contains the patron’s details. The file path will be passed to the newPatronFromFile() method, which will create a new Patron object and add it to the patrons list.

If the user inputs 2, the newPatronManually() function will be called and the user will be prompted to enter the id, name, address, and overdue fine in that order. This will create a new entry for the list of patrons.

If the user inputs 3, he or she will be prompted to enter the id of the patron that he or she wishes to eliminate. That value will be passed to the removePatronById() function, which will remove the Patron object from the list.

If the user inputs 4, the program will print the patrons using the displayPatrons() method.

If the user inputs 5, the program will stop with a break statement. Any other choice will display a message informing the user that his or her decision is not valid and will prompt him or her to try again.

**Testing Plan**

The testing of the application will occur as follows.

First, the case for the first choice of the switch statement (1) will be tested. If it works correctly, the program should take the path input from the user and read the contents of the file correctly, then it should take the contents and create a new Patron object with them to be added to the list. However, it might be the case that the user does not input the path correctly, which would cause an IO exception. To eliminate this, the code inside the newPatronFromFile() method must be surrounded by a try-catch block that will protect against IO exceptions, displaying a message when the file is not found. Different inputs from the user (like random letters or numbers, or a wrong path) will be employed to make sure that this method works. Since the success of the function at retrieving the information from the file depends on the format of the data (as specified in the requirements) it is important that the data is formatted the right way from the start. Cases for when the data is not properly formatted or is missing should also be tested, and the application should still be able to run.

Second, the case for the second choice of the switch statement (2) should be tested. According to the specifications, the id number has to be 7 digits long, so either exception handling or an if statement should be implemented to limit the input to no more and no less than 7 digits. Cases for when the user inputs letters or other symbols should also be included, which could be handled by a try-catch block that catches InputMismatch exceptions. According to user specifications, the Overdue Fine should be a number between $0 and $250. Another exception or if statement could be used to eliminate this problem and limit the user input to that range.

Next, the case for the third choice of the switch statement (3) should be tested. Since the program asks for an Id input, for the program to work the user needs to input a valid id. Cases for different numbers, letters, inputs containing both numbers and letters, and blank spaces should be tested. If one of these is used, the program should output a message saying that there is no match.

For the fourth and fifth choices of the switch statement (4 and 5) I do not have any conceivable test case other than an input from the user that is not included in the switch statement, which will be covered by a default statement. In this case the program will output a message letting the user know that his or her choice is invalid, and to try again.

**Deployment**

/\*  
 Marla Peraza Ravelo  
 CEN 3024C - Software Development 1  
 January 24, 2025  
 Main.java  
 This class will create the Library Management System, display the menu for the user to choose the actions, and call the methods to manipulate the patrons list  
 \*/  
import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
  
 //creating the objects and variables  
 LibraryManagementSystem lms = new LibraryManagementSystem();  
 Scanner scanner = new Scanner(System.*in*);  
 int userInput;  
  
 //do-while loop that contains the switch statement  
 do {  
 //Displaying title, patrons and menu  
 System.*out*.println("\nLibrary Management System");  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 lms.displayPatrons();  
 lms.displayMenu();  
 userInput = scanner.nextInt();  
 // Consume newline character  
 scanner.nextLine();  
  
 //beginning of the switch statement  
 switch (userInput) {  
 //Add patrons from file  
 case 1:  
 System.*out*.print("Enter file path: ");  
 String filePath = scanner.nextLine();  
 lms.newPatronFromFile(filePath);  
 break;  
 //Add a patron manually  
 case 2:  
 lms.newPatronManually(scanner);  
 break;  
 //Remove a patron by ID  
 case 3:  
 System.*out*.print("Enter ID to remove: ");  
 String id = scanner.nextLine();  
 lms.removePatronById(id);  
 break;  
 //Display all patrons  
 case 4:  
 lms.displayPatrons();  
 break;  
 //xit  
 case 5:  
 System.*out*.println("Exiting the system");  
 break;  
 //Invalid choice  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 }  
 } while (userInput != 5);  
  
 //closing the scanner  
 scanner.close();  
 }  
}

/\*  
 Marla Peraza Ravelo  
 CEN 3024C - Software Development 1  
 January 24, 2025  
 Patron.java  
 This class creates a Patron object and will print the id, name, address and overdue fine of each Patron  
 \*/  
class Patron {  
 //Attributes  
 private String id;  
 private String name;  
 private String address;  
 private double overdueFine;  
  
 /\*  
 method: Patron  
 parameters: id (String), name (String), address(String), overdueFine(double)  
 return: -  
 purpose: Constructor method, creates a Patron object  
 \*/  
 public Patron(String id, String name, String address, double overdueFine) {  
 this.id = id;  
 this.name = name;  
 this.address = address;  
 this.overdueFine = overdueFine;  
 }  
 /\*  
 method: getId  
 parameters: None  
 return: String  
 purpose: retrieves the id from the Patron object  
 \*/  
 public String getId() {  
 return id;  
 }  
  
 /\*  
 method: toString  
 parameters: None  
 return: String  
 purpose: prints the details of the Patron object  
 \*/  
 @Override  
 public String toString(){  
 return "{Id = " + id +"; Name = " + name + "; address = " + address + "; Overdue Fine = $" + overdueFine +"}";  
 }  
}

/\*  
 Marla Peraza Ravelo  
 CEN 3024C - Software Development 1  
 January 24, 2025  
 LibraryManagementSystem.java  
 This class contains all the methods necessary to manipulate a Patron object,  
 including adding patrons from a file or manually, removing patrons, displaying the current patrons in the  
 system, and displaying the LMS menu.  
 \*/  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.Scanner;  
import java.util.ArrayList;  
import java.util.List;  
  
public class LibraryManagementSystem {  
 //Attribute  
 private List<Patron> patrons;  
  
 /\*  
 method: LibraryManagementSystem  
 parameters: None  
 return: -  
 purpose: Constructor method, creates a LibraryManagementSystem object  
 \*/  
 public LibraryManagementSystem() {  
 patrons = new ArrayList<>();  
 }  
  
 /\*  
 method: newPatronFromFile  
 parameters: filePath(string)  
 return: void  
 purpose: reads the contents of a file using the file path, breaks them into id, name,  
 address, and overdue fine, and creates a new Patron object to be added to the patrons list  
 \*/  
 public void newPatronFromFile(String path) {  
 try (BufferedReader r = new BufferedReader(new FileReader(path))) {  
 String ln; //line  
 while ((ln = r.readLine()) != null) {  
 String[] parts = ln.split("-");  
 if (parts.length == 4) {  
 String id = parts[0];  
 String name = parts[1];  
 String address = parts[2];  
 double overdueFine = Double.*parseDouble*(parts[3]);  
 patrons.add(new Patron(id, name, address, overdueFine));  
 }  
 }  
 } catch (IOException e) {  
 System.*out*.println("Error reading the file: " + e.getMessage());  
 }  
 }  
  
 /\*  
 method: newPatronManually  
 parameters: scanner(Scanner)  
 return: void  
 purpose: prompts the user to input id, name, address and overdue fine, and creates a  
 new Patron object to be added to the patrons list  
 \*/  
 public void newPatronManually(Scanner scanner) {  
 try {  
 System.*out*.print("Enter a 7-digit ID: ");  
 String id = scanner.nextLine();  
 if (id.length() != 7) {  
 throw new InvalidIdException("The ID has to have exactly 7 digits");  
 }  
 System.*out*.print("Enter a name: ");  
 String name = scanner.nextLine();  
 System.*out*.print("Enter an address: ");  
 String address = scanner.nextLine();  
 System.*out*.print("Enter the overdue fine (from 0.00 to 250.00): ");  
 double overdueFine = scanner.nextDouble();  
 if (overdueFine < 0.00 || overdueFine > 250.00) {  
 throw new OverdueRangeException("Overdue Fine should be between $0 and $250.00");  
 }  
 // Get rid of the newline character  
 scanner.nextLine();  
  
 patrons.add(new Patron(id, name, address, overdueFine));  
 System.*out*.println("Patron has been added.");  
 } catch (InvalidIdException e) {  
 System.*out*.println(e.getMessage());  
 }  
 catch(OverdueRangeException e){  
 System.*out*.println(e.getMessage());  
 }  
 }  
 /\*  
 method: removePatronById  
 parameters: id (String)  
 return: void  
 purpose: Removes a patron from the patrons list given its id  
 \*/  
  
 public void removePatronById(String id) {  
 boolean isRemoved = patrons.removeIf(patron -> patron.getId().equals(id));  
 if (isRemoved) {  
 System.*out*.println("Patron has been removed");  
 } else  
 System.*out*.println("Id provided does not match any existing patrons");  
 }  
  
 /\*  
 method: displayPatrons  
 parameters: None  
 return: void  
 purpose: displays all the current patrons' information on the screen. If the  
 list is empty, returns a corresponding message  
 \*/  
 public void displayPatrons() {  
 if (patrons.isEmpty()) {  
 System.*out*.println("\nCurrently there are no patrons in the system");  
 } else {  
 System.*out*.println("Patrons in the system:");  
 for (Patron p : patrons) {  
 System.*out*.println(p);  
 }  
 }  
 }  
  
 /\*  
 method: displayMenu  
 parameters: None  
 return: void  
 purpose: displays the LMS menu  
 \*/  
 public void displayMenu() {  
 System.*out*.println("\n1. Add patrons from a file"+  
 "\n2. Add a patron manually" +  
 "\n3. Remove a patron using ID number"+  
 "\n4. Display all patrons"+  
 "\n5. Exit"+  
 "\nEnter a number to start: ");  
 }  
}

/\*  
 Marla Peraza Ravelo  
 CEN 3024C - Software Development 1  
 January 24, 2025  
 OverdueRangeException.java  
 This class creates an OverdueRangeException which displays a message when it is thrown. It is used to  
 make sure overdueFine stays between $0.00 and $250.00  
 \*/  
public class OverdueRangeException extends Exception{  
 /\*  
 method: OverdueRangeException  
 parameters: message(String)  
 return: -  
 purpose: Constructor method, creates an OverdueRangeException  
 \*/  
 public OverdueRangeException(String message){  
 super(message);  
 }  
  
}

/\*  
 Marla Peraza Ravelo  
 CEN 3024C - Software Development 1  
 January 24, 2025  
 InvalidIdException.java  
 This class creates an InvalidIdException which displays a message when it is thrown. It is used to  
 make sure the id entered has exactly 7 digits  
 \*/  
public class InvalidIdException extends Exception{  
 /\*  
 method: InvalidIdException  
 parameters: message(String)  
 return: -  
 purpose: Constructor method, creates an InvalidIdException  
 \*/  
public InvalidIdException(String message){  
 super(message);  
 }  
}