|  |
| --- |
| Software Development Plan: Library Management System |
| **Software Development I** |
| **CEN-3024C** |

|  |
| --- |
| Micah Phillip  9-1-2025 |

# Introduction

The purpose of this software development plan is to outline the design, structure, and development strategy for a console-based Library Management System (LMS). This system will be developed to meet the needs of the client, a local librarian, seeking an efficient and straightforward solution for managing patron information. The plan serves to guide development, ensure alignment with user needs, and provide clarity on the required functionality, implementation steps, and testing strategy to complete the Library Management System.

# System Requirements

The library management system must fulfill the following functional requirements:

* 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 dollars to 250 dollars.
  + Each line of input text file represents a patron with the individual’s information like name and address being separated by a dash between them.
  + The fine do not have a dollar sign in front of them
* Have a menu-based user interface on the home page of the console.
  + Interface must have at least three interactions. Add Patron, Remove Patron, and Display all Patrons.
* Be able to add patrons. This will be achieved by using a text file or by inputting the information manually.
  + This system should allow the user to enter patrons for as long as there is space to store data.
* Be able to remove patrons from the system.
  + This will be done by entering the unique 7-digit ID number into the system to remove the entry.
* Be able to display all patrons currently stored in the system.
* Has Internal memory storage for the data

## Constraints

* Must be implemented as a command-line interface (CLI) application.
* Patron IDs must be 7-digit unique integers.
* Overdue fines must be a decimal number between $0.00 and $250.00.
* The application must parse input from a text file where patron data is dash-separated.

# 2. User Needs and Interaction

## Intended User

* Librarian using a terminal to manage the patron records.

## Use Cases

* UC1: Add Patrons from File
  + User selects option to import patrons.
  + System reads file line-by-line, parses data, and adds patrons to internal list.
* UC2: Manually Add Patron
  + User selects option to add manually.
  + System prompts for ID, name, address, and fine.
  + Validates input and adds to list.
* UC3: Remove Patron
  + User selects option to remove.
  + System prompts for patron ID.
  + System deletes record if it exists.
* UC4: View Patron List
  + User selects option to view.
  + System displays all patrons in a readable format.
* UC5: Exit
  + User selects option to quit the program.

## User expectations for the library management system

* A simple, intuitive command-line menu that can be navigated with simple keystrokes.
* Accurate validation (e.g., rejecting duplicate IDs or invalid fines).
* Quick loading and display of data in a clear easy to read manner.
* Confirmation messages for actions like add/remove.

The Library Management System must be able to perform the following tasks:

* Maintain a patron list efficiently and display it in a clear readable structure.
* Avoid manual record-keeping or database use.
* Quickly load bulk patron data via a file and be able to send an error message in case of incomplete files being loaded.
* Add or remove individual patrons quickly and easily.
* View the full list of patrons for account lookup or reporting.

# Implementation Plan/Development Order

Coding Language: Java since it is the easiest to build this system on.

Data Storage: The data (patrons) will be stored in a list format for easy readability and display to the user (the librarian).

Main Components: Patron Class, Library Management System Class, and Main Class.

# 1. Patron Class

This class represents a single library patron.

## Attributes:

* int id – Unique 7-digit patron ID
* String name – Full name of the patron
* String address – Patron's address
* double fineAmount – Overdue fine (from $0.00 to $250.00)

Methods:

* Constructor: Initializes all fields.
* toString() – Returns a formatted string representing the patron.

+-------------------------------+

| Patron |

+-------------------------------+

| - id: int |

| - name: String |

| - address: String |

| - fineAmount: double |

+-------------------------------+

| +Patron(id: int, |

| name: String, |

| address: String, |

| fineAmount: double) |

| +getId(): int |

| +getName(): String |

| +getAddress(): String |

| +getFineAmount(): double |

| +toString(): String |

+-------------------------------+

# 2. LibraryManagementSystem Class

This class will manage the list of patrons and handle user interaction via the console.

## Attributes:

* ArrayList<Patron> patrons – A list to store Patron objects.

## Methods:

* addPatron(Patron patron) – Adds a single patron manually.
* addPatronsFromFile(String filePath) – Reads patrons from a text file and adds them to the list.
* removePatronById(int id) – Removes a patron based on the provided ID.
* displayPatrons() – Displays all current patrons in the system.
* runMenu() – Controls the menu loop for CLI interactions.

+--------------------------------------------+

| LibraryManagementSystem |

+--------------------------------------------+

| - patrons: List<Patron> |

| - in: Scanner |

+--------------------------------------------+

| +runMenu(): void |

| -addPatronsFromFile(): void |

| -addPatronManually(): void |

| -removePatronById(): void |

| -displayPatrons(): void |

| -parseAndAdd(line: String): boolean |

| -addValidated(p: Patron): boolean |

| -printMenu(): void |

+--------------------------------------------+

# 3. Main Class

The entry point of the program.

## Method:

* main(String[] args) – Creates a LibraryManagementSystem instance and calls runMenu().

+--------------------+

| Main |

+--------------------+

| +main(args: String[]): void |

+--------------------+

# Testing Strategy

## Unit Testing

* Test each method of the Patron and LMS classes.
  + Example: Ensure add\_patron() rejects duplicate IDs.

## Input Validation

* Test with valid and invalid data: short/long IDs, missing fields, invalid fines.

## File Parsing

* Test file input with edge cases (extra dashes, blank lines, malformed records).

## CLI Menu Navigation

* Test for invalid menu options.
* Confirm loop exits only when requested.

# Deployment

*/\*\*  
 \* Name: Micah Phillip  
 \* Course: CEN 3024 - Software Development 1  
 \* Date: September 06, 2025,  
 \* Class: Main.java  
 \* This is the entry point of the program. It initializes the LibraryManagementSystem  
 \* and starts the command-line interface. The overall goal of the program is to  
 \* manage library patrons by allowing users to import, add, remove, and display patron data.  
 \*/*public class Main {  
 public static void main(String[] args) {  
 new LibraryManagementSystem().runMenu();  
 }  
}

*/\*\*  
 \* Name: Micah Phillip  
 \* Course: CEN 3024 - Software Development 1  
 \* Date: September 06, 2025,  
 \* Class: Patron.java  
 \* This class defines a single library patron with ID, name, address, and fine amount.  
 \* It is an immutable value object used by the LibraryManagementSystem class to track users.  
 \*/*import java.text.DecimalFormat;  
  
public class Patron {  
 private final int id;  
 private final String name;  
 private final String address;  
 private final double fineAmount;  
  
 private static final DecimalFormat *MONEY* = new DecimalFormat("0.00");  
  
 // Constructor  
 public Patron(int id, String name, String address, double fineAmount) {  
 this.id = id;  
 this.name = name;  
 this.address = address;  
 this.fineAmount = fineAmount;  
 }  
  
 // Getters  
 public int getId() { return id; }  
 public String getName() { return name; }  
 public String getAddress() { return address; }  
 public double getFineAmount() { return fineAmount; }  
  
 */\*\*  
 \* Method: toString  
 \* Purpose: Returns a formatted string representation of the patron  
 \* Arguments: None  
 \* Returns: A string showing ID, name, address, and fine  
 \*/* @Override  
 public String toString() {  
 return String.*format*("%07d | %-20s | %-30s | $%s",  
 id, name, address, *MONEY*.format(fineAmount));  
 }  
}

*/\*\*  
 \* Name: Micah Phillip  
 \* Course: CEN 3024 - Software Development 11  
 \* Date: September 06, 2025,  
 \* Class: LibraryManagementSystem.java  
 \* This class provides the interface and functionality for managing patrons.  
 \* It allows the user to add patrons (manually or from file), remove patrons,  
 \* and display all current patrons. It handles validation and data integrity.  
 \*/*import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.\*;  
  
public class LibraryManagementSystem {  
  
 private final List<Patron> patrons = new ArrayList<>();  
 private final Scanner in = new Scanner(System.*in*);  
  
 */\*\*  
 \* Method: runMenu  
 \* Purpose: Starts the command-line interface for the system  
 \* Arguments: None  
 \* Returns: void  
 \*/* public void runMenu() {  
 String choice;  
 do {  
 printMenu();  
 choice = in.nextLine().trim();  
 switch (choice) {  
 case "1" -> addPatronsFromFile();  
 case "2" -> addPatronManually();  
 case "3" -> removePatronById();  
 case "4" -> displayPatrons();  
 case "5" -> System.*out*.println("Good‑bye!");  
 default -> System.*out*.println("❌ Invalid option. Try again.");  
 }  
 } while (!"5".equals(choice));  
 }  
  
 */\*\*  
 \* Method: addPatronsFromFile  
 \* Purpose: Imports patrons from a file provided by the user  
 \* Arguments: None  
 \* Returns: void  
 \*/* private void addPatronsFromFile() {  
 System.*out*.print("Enter path to patron file: ");  
 String path = in.nextLine().trim();  
 int imported = 0, skipped = 0;  
  
 try (BufferedReader br = new BufferedReader(new FileReader(path))) {  
 String line;  
 while ((line = br.readLine()) != null) {  
 if (parseAndAdd(line)) imported++; else skipped++;  
 }  
 System.*out*.printf("✔ Import finished (%d added, %d skipped).%n", imported, skipped);  
 } catch (IOException e) {  
 System.*out*.println("❌ Could not read file: " + e.getMessage());  
 }  
 }  
  
 */\*\*  
 \* Method: addPatronManually  
 \* Purpose: Adds a single patron from user input  
 \* Arguments: None  
 \* Returns: void  
 \*/* private void addPatronManually() {  
 try {  
 System.*out*.print("7‑digit ID: ");  
 int id = Integer.*parseInt*(in.nextLine().trim());  
 System.*out*.print("Name: ");  
 String name = in.nextLine().trim();  
 System.*out*.print("Address: ");  
 String address = in.nextLine().trim();  
 System.*out*.print("Fine (0‑250): ");  
 double fine = Double.*parseDouble*(in.nextLine().trim());  
  
 if (addValidated(new Patron(id, name, address, fine))) {  
 System.*out*.println("✔ Patron added.");  
 }  
 } catch (NumberFormatException ex) {  
 System.*out*.println("❌ Invalid numeric entry.");  
 }  
 }  
  
 */\*\*  
 \* Method: removePatronById  
 \* Purpose: Removes a patron using their 7-digit ID  
 \* Arguments: None  
 \* Returns: void  
 \*/* private void removePatronById() {  
 System.*out*.print("Enter 7‑digit ID to remove: ");  
 try {  
 int id = Integer.*parseInt*(in.nextLine().trim());  
 if (patrons.removeIf(p -> p.getId() == id)) {  
 System.*out*.println("✔ Patron removed.");  
 } else {  
 System.*out*.println("⚠ ID not found.");  
 }  
 } catch (NumberFormatException ex) {  
 System.*out*.println("❌ Not a valid ID.");  
 }  
 }  
  
 */\*\*  
 \* Method: displayPatrons  
 \* Purpose: Displays all patrons currently stored  
 \* Arguments: None  
 \* Returns: void  
 \*/* private void displayPatrons() {  
 if (patrons.isEmpty()) {  
 System.*out*.println("(No patrons in system)");  
 } else {  
 System.*out*.println("ID | Name | Address | Fine");  
 System.*out*.println("---------+----------------------+--------------------------------+-------");  
 patrons.forEach(System.*out*::println);  
 System.*out*.printf("Total patrons: %d%n", patrons.size());  
 }  
 }  
  
 */\*\*  
 \* Method: parseAndAdd  
 \* Purpose: Parses a single line from a file and attempts to add a Patron  
 \* Arguments: String line - the raw input line from the file  
 \* Returns: boolean - true if added, false if invalid  
 \*/* private boolean parseAndAdd(String line) {  
 String[] parts = line.split("\\s\*-\\s\*");  
 if (parts.length != 4) {  
 System.*out*.println("Skipping malformed line: " + line);  
 return false;  
 }  
 try {  
 int id = Integer.*parseInt*(parts[0]);  
 String name = parts[1];  
 String addr = parts[2];  
 double fine = Double.*parseDouble*(parts[3]);  
  
 return addValidated(new Patron(id, name, addr, fine));  
 } catch (NumberFormatException ex) {  
 System.*out*.println("Skipping malformed numeric data: " + line);  
 return false;  
 }  
 }  
  
 */\*\*  
 \* Method: addValidated  
 \* Purpose: Validates and adds a Patron to the list  
 \* Arguments: Patron p - the patron to add  
 \* Returns: boolean - true if valid and added, false otherwise  
 \*/* private boolean addValidated(Patron p) {  
 if (String.*valueOf*(p.getId()).length() != 7) {  
 System.*out*.println("❌ ID must be exactly 7 digits.");  
 return false;  
 }  
 if (patrons.stream().anyMatch(existing -> existing.getId() == p.getId())) {  
 System.*out*.println("❌ Duplicate ID.");  
 return false;  
 }  
 if (p.getFineAmount() < 0 || p.getFineAmount() > 250) {  
 System.*out*.println("❌ Fine must be between 0 and 250.");  
 return false;  
 }  
 patrons.add(p);  
 return true;  
 }  
  
 */\*\*  
 \* Method: printMenu  
 \* Purpose: Displays the main menu  
 \* Arguments: None  
 \* Returns: void  
 \*/* private void printMenu() {  
 System.*out*.println("""  
 \n===== Library Management System =====  
 1) Import patrons from file  
 2) Add patron manually  
 3) Remove patron  
 4) Display all patrons  
 5) Exit  
 Select an option:""");  
 }  
}