

# **Project Based Evaluation**

## **Java Programming**

### **Project Id-(JPG09-15)**

#### **Project Report**

#### **Semester-IV (Batch-2023)**

### **Movie Ticket Booking System**



#### **Supervised By:**

Dr. Susama bagchi

#### **Submitted By:**

Karanbir , 2310990708

Kartik Jindal , 2310990709

Nahar , 2310990744

Pranav Goyal , 2310991426

**Department of Computer Science and Engineering  
Chitkara University Institute of Engineering &  
Technology , Chitkara University Punjab**

## **INDEX**

<b>Sr. No</b>	<b>Table of Content</b>	<b>Page</b>
1	Abstract	3-5
2	Table of content	6
3	Introduction	7-12
4	Problem Definition and Requirements	13-14
5	Proposed Design / Methodology	15
6	Results	16-18
7	References	19

## **Abstract :-**

This project presents a Movie Ticket Booking System developed in Java using a Command-Line Interface (CLI). The system is designed to streamline the booking and cancellation of movie tickets through a simple, text-based menu, providing an interactive and intuitive experience for users. Core functionalities include viewing available movies and showtimes, booking and cancelling seats, and viewing personal booking history. The system uses a 2D array to manage seat availability for each movie and a singly linked list to store and retrieve booking records. Data persistence is handled through basic file operations, allowing the system to retain booking data across sessions. Additionally, the application includes a virtual coin system, where users start with a set amount of coins and spend or earn coins based on bookings and cancellations. The project effectively demonstrates object-oriented programming principles, basic data structures, and file handling in Java without relying on external libraries or graphical interfaces.

## **Key Features :-**

### **Movie Management**

- Each movie entry includes a title and showtime.
- A 2D boolean array is used to manage seat availability (5 rows  $\times$  10 columns).
- Users can view the list of available movies and their showtimes.

### **Ticket Booking and Cancellation**

- Users can book or cancel seats by specifying row and column numbers.
- A virtual coin system deducts 100 coins per seat booked and refunds 100 coins for cancellations.
- Seat availability is updated in real-time during booking and cancellation.

### **Data Handling Using Core Structures**

- A 2D array tracks seat availability for each movie show.

- A singly linked list manages user booking history with dynamic addition and removal of tickets.
- The linked list structure supports operations like adding, removing, and retrieving booked ticket records.

### **File-Based Persistence**

- Bookings are saved to and loaded from a plain text file (bookings.txt).
- Ensures that user booking data is retained between sessions without using a database.

### **Command-Line Interface (CLI)**

- Users interact with the system through a simple, text-based menu.
- Options include viewing movies, booking seats, cancelling seats, and checking booking history.
- Designed for clarity and ease of use in educational or terminal-based environments.

## 1. Introduction :-

The system is designed to streamline the booking and cancellation of movie tickets through a simple, text-based menu, providing an interactive and intuitive experience for users. Core functionalities include viewing available movies and showtimes, booking and cancelling seats, and viewing personal booking history. The system uses a 2D array to manage seat availability for each movie and a singly linked list to store and retrieve booking records. Data persistence is handled through basic file operations, allowing the system to retain booking data across sessions. Additionally, the application includes a virtual coin system, where users start with a set amount of coins and spend or earn coins based on bookings and cancellations.

### Objectives :

In today's digital world, the manual handling of movie ticket reservations through traditional methods—such as counters or paper records—is often time-consuming, error-prone, and inefficient. To address these challenges, this project introduces a Movie Ticket Booking System developed in Java using a Command-Line Interface (CLI).

The application is designed to help users book and cancel movie tickets through a simple, menu-driven interface in the terminal. It provides features such as viewing available movies and showtimes, managing seat bookings, tracking booking history, and simulating basic payment functionality through a virtual coin system.

The system is built with an educational focus on core Java programming concepts, including object-oriented design, 2D arrays, custom linked lists, and file handling. It does not rely on external databases or GUIs, making it lightweight, easy to understand, and ideal for beginners or academic learning.

### Efficient Movie Booking System

Simplify the booking and cancellation of movie tickets through a command-line interface.

### Interactive User Interface (CLI)

Present users with a menu to interact with the system using text-based input/output, eliminating the need for graphical components.

### **Core Java and DSA Implementation**

Utilize Java's object-oriented features, a 2D array for seat tracking, and a singly linked list to manage ticket records dynamically.

### **Seat Booking with Real-Time Feedback**

Allow users to choose and confirm seats while updating seat availability instantly.

### **Virtual Coin System**

Simulate a lightweight payment system by managing user coins for each booking and refund upon cancellation.

### **Data Persistence with File I/O**

Store booking records in a text file to ensure data is preserved across sessions without using a database.

## **1.2 Significance :**

The Movie Ticket Booking System holds value both as a practical utility and an educational tool

### **Educational Value in Java Learning**

By implementing real-world functionalities using basic Java constructs, the system helps learners understand classes, file operations, and linked list handling in a real scenario.

### **Automation of Manual Processes**

It reduces reliance on manual booking logs and introduces a more structured approach to seat and booking data management.

### **Improved Booking Accuracy**

Real-time seat status tracking ensures users cannot double-book or cancel seats that aren't reserved, improving accuracy and reliability.

### **Lightweight and Easily Extensible**

The absence of GUI or database makes it easy to deploy, modify, and scale for educational or small project use cases.

**Data Integrity via File Storage**

Bookings are consistently written to a file, ensuring data remains safe even after the program ends.

**Broader Use Potential**

The design and logic can be adapted to other CLI-based systems such as bus ticketing, classroom seat allocation, or event registrations.

## **2. Problem Definition and Requirements :-**

### **2.1 Problem Definition :**

In the context of small-scale cinemas or learning environments, traditional ticket booking methods—such as paper registers or spreadsheet logs—often result in errors, confusion, and inefficient operations. Managing seat availability manually can lead to double bookings, inaccurate records, and a poor customer experience.

Moreover, basic ticketing systems may lack persistence and structure, requiring users to re-enter data in each session. There is a need for a lightweight, offline-capable, and beginner-friendly system that automates the booking process and ensures reliable data handling.

This project addresses the problem by introducing a Java-based Command-Line Movie Ticket Booking System, which:

1. Allows users to book and cancel seats interactively
2. Tracks and updates seat availability in real-time using a 2D array.
3. Stores booking records using a custom linked list.
4. Maintains booking history through file I/O for persistent storage.
5. Simulates payment logic using a virtual coin system

This system is particularly designed for educational use, demonstrating practical applications of data structures and file operations without external dependencies like GUIs or databases.

### **2.2 Requirements :**

#### **Functional Requirements:**

##### **Movie List and Showtimes**

Users can view a predefined list of movies and their associated showtimes.

##### **Seat Booking**

Users can select a movie and book one or more seats by specifying row and column numbers.

Each seat booking deducts 100 virtual coins from the user.

##### **Seat Cancellation**

Booked seats can be cancelled by specifying the same row and column.

Each successful cancellation refunds 100 coins to the user.



## **Booking History**

Users can view their complete booking history during the session.

Bookings are stored and displayed with movie name, time, and seat number.

## **File Persistence**

Booking records are saved to a text file (bookings.txt) and reloaded at startup.

## **Non-Functional Requirements:**

### **Usability**

The text-based interface should be simple and easy to navigate using standard terminal inputs.

### **Performance**

Seat booking, cancellation, and lookup operations should complete quickly, even with multiple records.

### **Reliability**

The system should ensure data integrity when saving and loading booking history from the file.

### **Portability**

The application should run on any machine with Java installed, without requiring external libraries or GUIs.

### **Maintainability**

The code should be modular and easy to extend — such as adding more movies or adjusting seat layout in the future.

### **Simplicity Over Complexity**

The system avoids GUI, networking, or database features to keep it accessible for students learning core Java.

### **3. Methodology for the Smart Student Portal :-**

The development of the Movie Ticket Booking System was approached in a structured, modular manner to ensure clarity, maintainability, and proper demonstration of core programming principles in Java. The focus was on using fundamental data structures, object-oriented programming, and file I/O operations, rather than complex frameworks or graphical interfaces.

#### **1. Requirements Gathering and Analysis**

Objective Definition:

Identify the basic operations needed — booking seats, cancelling bookings, tracking user history, and ensuring data persistence through files.

Constraints Considered:

Ensure the system works through a CLI (Command-Line Interface) with no dependency on GUI components or external libraries.

#### **2. System Design:**

**Architecture Design:**

The system is divided into core classes:

Movie: Manages seat availability using a 2D array.

Ticket: Holds details of a single booking.

Ticket Node and Ticket LinkedList: Handle dynamic storage of bookings.

Movie Booking CLI: Controls user interaction through a text-based menu.

**Data Flow:**

Booking and cancellation operations modify the seat matrix and linked list.

Booking records are written to a file and loaded upon startup.

### **3. Implementation:**

Technology Stack:

Language: Java

Interface: CLI (Console-based interaction)

Persistence: Text file (bookings.txt)

Steps Followed:

Created data models for movies and tickets.

Implemented core logic for seat booking and cancellation.

Developed linked list for maintaining booking history.

Integrated file reading/writing for data persistence.

Implemented a coin system to simulate basic transactions.

### **4. Testing:**

#### **Unit Testing:**

Validate individual modules such as booking logic, seat availability updates, and payment processing.

#### **Integration Testing:**

Ensure smooth communication between modules (e.g., UI and database, payment and booking engine).

#### **User Acceptance Testing (UAT):**

Invite stakeholders to test the system and provide feedback on usability, navigation, and overall experience

### **5. Deployment:**

Environment:

The program runs in any Java-supported environment (command-line based) without special setup.

Execution:

The compiled .class file is executed via the terminal using the java command.

**User Instructions:**

Simple terminal prompts guide the user through all actions, requiring no prior training.

**6. Maintenance and Support:****Environment:**

The program runs in any Java-supported environment (command-line based) without special setup.

**Execution:**

The compiled .class file is executed via the terminal using the java command.

**User Instructions:**

Simple terminal prompts guide the user through all actions, requiring no prior training.

## 4. Results :-

```
PS C:\Users\ASUS\OneDrive\Documents\Java,DSAProject\JavaProject> javac MovieBookingCLI.java
PS C:\Users\ASUS\OneDrive\Documents\Java,DSAProject\JavaProject> java MovieBookingCLI.java

=== Movie Booking System ===
Coins: 1000
1. View Movies
2. Book Seats
3. Cancel Seats
4. My Bookings
5. Exit
Choose an option: 2
```

Figure 1

```
Select a movie number: 3
How many seats to book? 2
How many seats to book? 2
Enter Row (1-5) and Column (1-10) for seat 1:
1 4
Seat booked successfully.
Enter Row (1-5) and Column (1-10) for seat 2:
2 4
Seat booked successfully.

=== Movie Booking System ===
Coins: 800
How many seats to book? 2
Enter Row (1-5) and Column (1-10) for seat 1:
1 4
Seat booked successfully.
Enter Row (1-5) and Column (1-10) for seat 2:
2 4
How many seats to book? 2
Enter Row (1-5) and Column (1-10) for seat 1:
1 4
```

Figure 2

```
class TicketNode {
    Ticket ticket;
    TicketNode next;

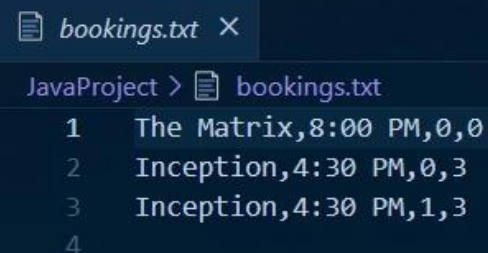
    TicketNode(Ticket ticket) {
        this.ticket = ticket;
    }
}

class TicketLinkedList {
    private TicketNode head;
    private final String filePath = "bookings.txt";

    public TicketLinkedList() {
        loadFromFile();
    }

    public void add(Ticket ticket) {
        TicketNode newNode = new TicketNode(ticket);
        if (head == null)
            head = newNode;
        else {
            TicketNode current = head;
            while (current.next != null)
                current = current.next;
            current.next = newNode;
        }
        saveToFile();
    }
}
```

Figure 3



The screenshot shows a text editor window titled "bookings.txt". The content of the file is as follows:

```
JavaProject > bookings.txt
1 The Matrix,8:00 PM,0,0
2 Inception,4:30 PM,0,3
3 Inception,4:30 PM,1,3
4
```

Figure 4

```

class TicketNode {
    Ticket ticket;
    TicketNode next;

    TicketNode(Ticket ticket) {
        this.ticket = ticket;
    }
}

class TicketLinkedList {
    private TicketNode head;
    private final String filePath = "bookings.txt";

    public TicketLinkedList() {
        loadFromFile();
    }

    public void add(Ticket ticket) {
        TicketNode newNode = new TicketNode(ticket);
        if (head == null)
            head = newNode;
        else {
            TicketNode current = head;
            while (current.next != null)
                current = current.next;
            current.next = newNode;
        }
        saveToFile();
    }
}

```

Figure 5

```

public class MovieBookingCLI {
    private static final Scanner scanner = new Scanner(System.in);
    private static final List<Movie> allMovies = new ArrayList<>();
    private static final TicketLinkedList bookedTickets = new TicketLinkedList();
    private static int userCoins = 1000;

    Run | Debug
    public static void main(String[] args) {
        seedMovies();
        while (true) {
            System.out.println(x:"\n=== Movie Booking System ===");
            System.out.println("Coins: " + userCoins);
            System.out.println(x:"1. View Movies");
            System.out.println(x:"2. Book Seats");
            System.out.println(x:"3. Cancel Seats");
            System.out.println(x:"4. My Bookings");
            System.out.println(x:"5. Exit");
            System.out.print(s:"Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1 -> listMovies();
                case 2 -> handleSeats(booking:true);
                case 3 -> handleSeats(booking:false);
                case 4 -> showBookingHistory();
                case 5 -> System.exit(status:0);
                default -> System.out.println(x:"Invalid option.");
            }
        }
    }
}

```

Figure 6

## **5. References :-**

### **5.1 Official Documentation**

Java Documentation: <https://docs.oracle.com/en/java/>

Swing Documentation: <https://docs.oracle.com/javase/tutorial/uiswing/>

### **5.2 Educational Platforms**

GeeksforGeeks: <https://www.geeksforgeeks.org/>

W3Schools: <https://www.w3schools.com/>

Coursera: <https://www.coursera.org/>

### **5.3 Programming Communities**

Stack Overflow: <https://stackoverflow.com/>

GitHub: <https://github.com/>

### **5. 4 Software Development Blogs**

Medium: <https://medium.com/>

Dev.to: <https://dev.to/>

### **5.5 Code Repositories and Examples**

GitHub Gists: <https://gist.github.com>