MINI PROJECT REPORT

ON

ONLINE MOVIE TICKET BOOKING SYSTEM

Submitted by

VARSHA V S

NC.SC.U4CSE24049

For

23CSE111- Object Oriented Programming

II Semester

B.Tech. CSE

School of Computing

Amrita Vishwa Vidyapeetham, Nagercoil

S. No.	Section	Page No.
1.	Introduction	1
2.	Problem Statement	2
3.	Objectives30 of the Project	4
4.	System Requirements	5
4.1	Hardware Requirements	5
4.2	Software Requirements	5
5.	Modules of the Project	6
5.1	User Module	6
5.2	Movie Module	7
5.3	Show Module	8
5.4	Booking Module	9
5.5	Seat Module	10
6.	System Design	11
6.1	UML Diagrams	11
6.1.1	Use Case Diagram	11
6.1.2	Class Diagram	12
6.1.3	Object Diagram	13
6.1.4	Activity Diagram	14
7.	System Implementation (Java Code)	15
7.1	Code Overview	15
7.2	Functionalities	17

8.	Output Snapshots (Optional)	18
9.	Applications of the Project	19
10.	Limitations of the Project	21
11.	Conclusion	22
12.	Bibliography	23

1.INTRODUCTION

The Online Movie Ticket Booking System is a Java-based desktop application developed using Swing for the graphical user interface. It provides users with a seamless experience to register, log in, browse available movies, select showtimes, choose seats visually, and make payments—all within an attractive and interactive GUI environment. Online Movie Ticket Booking System is a user-friendly Java system that makes movie ticket booking possible for users, saving them time and effort in the process. Rather than approaching the theater itself, users may utilize this system to browse existing movies, pick showtimes, pick seats, and book tickets remotely. This project is developed with the use of core Java and is connected to a database in order to store movie information, user information, and booking details. The system aims to improve the speed of booking tickets, decrease manual labor, and give better convenience to the people watching movies and theater workers. This is a basic booking system that provides some common features like user registration, movie listing, selection of seats, and confirmation of tickets. This is a perfect project to showcase knowledge of Java programming, JDBC connectivity to databases, and fundamental UI development with Java Swing or console-based operations. Developed with Java for the inner application logic and combined with a database system such as MySQL, the project maintains data consistency, security, and real-time updating. The system not only benefits users but also cinema administrators through automation of processes and minimizing manual effort.

2.Problem Statement:

The traditional method of booking movie tickets often involves significant inefficiencies and drawbacks that hinder the overall customer experience. Moviegoers are required to either physically visit the theater or make a phone call to reserve their tickets, both of which can be time-consuming and inconvenient. The manual process not only adds a layer of complexity but also introduces the possibility of human errors, such as double bookings, incorrect reservations, or delays in processing. Moreover, customers may have to endure long waiting times, particularly during peak hours or blockbuster releases, which can cause frustration and dissatisfaction.

Furthermore, movie theaters frequently face challenges in managing seat availability effectively. Without a centralized and automated system, seat allocation becomes prone to

mismanagement, often resulting in double bookings or customers being assigned seats that are already taken. This lack of real-time availability tracking not only impacts the efficiency of the system but also creates confusion among both customers and theater staff. Additionally, theater-goers have no easy way of checking or modifying their bookings, which can lead to a lack of transparency and trust in the booking process. The absence of a simple and reliable way to view previous or upcoming bookings makes it difficult for users to keep track of their movie-going plans.

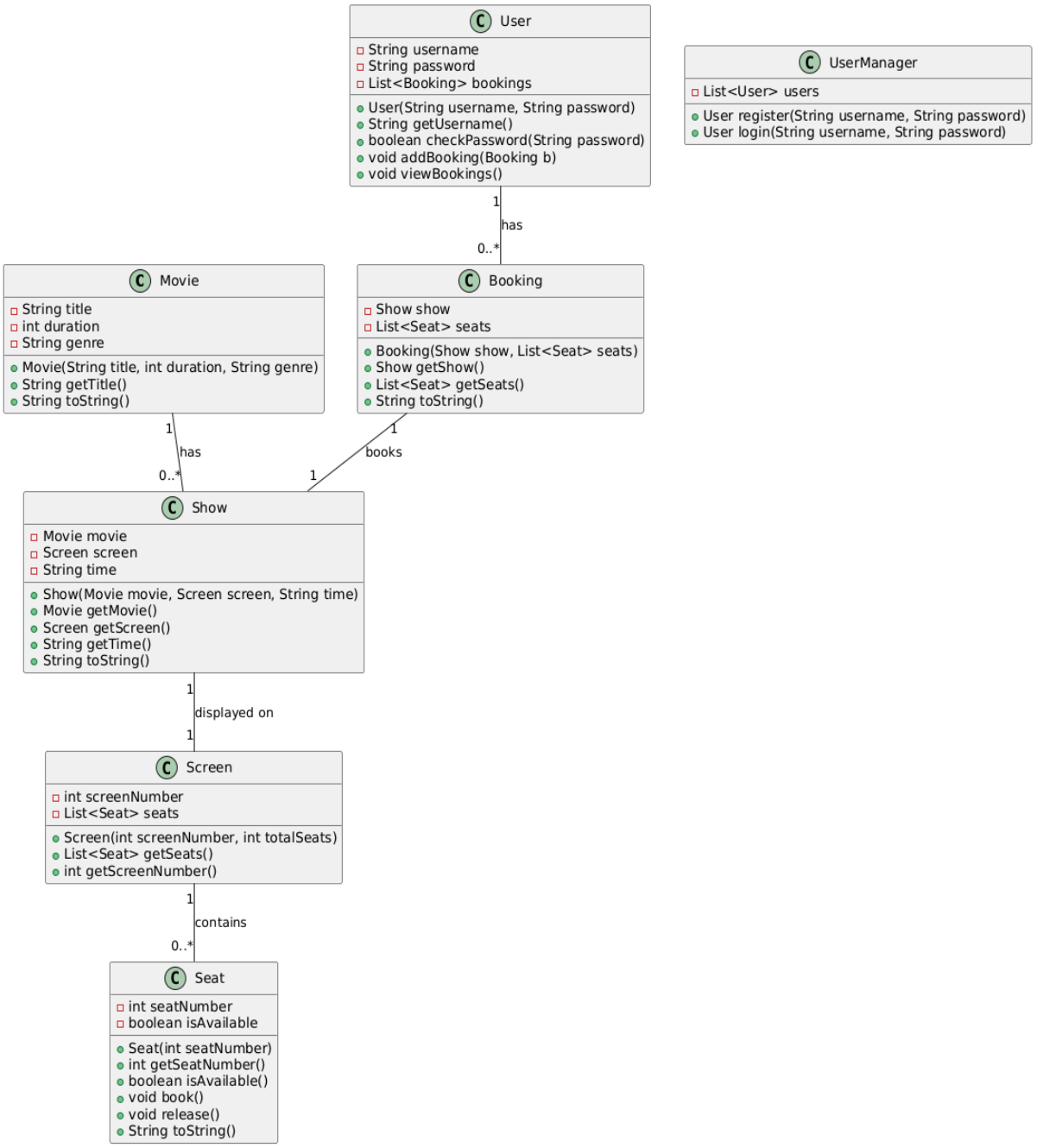
From a theater management perspective, the manual process is often prone to inefficiencies, especially when dealing with large volumes of ticket sales. On busy days or during the release of highly anticipated films, the system can easily become overwhelmed, leading to operational bottlenecks, which further frustrate both customers and staff. Furthermore, theaters that operate using traditional methods often miss out on opportunities to integrate additional features like online promotions, loyalty programs, or special offers, which can enhance customer satisfaction and drive sales. Without an efficient system in place to manage these aspects, theaters can lose out on potential revenue and repeat customers.

The Movie Ticket Booking System seeks to address these issues by providing a fully automated, digital platform for movie bookings. By moving away from manual processes and embracing automation, the system offers a fast, convenient, and error-free way for customers to browse available movies, check showtimes, select seats, and confirm their bookings online. Real-time seat availability checks ensure that users can select available seats without the fear of double bookings, while automated confirmation ensures that all bookings are processed promptly and accurately. The system also provides an easy-to-use interface, allowing users to easily navigate through available shows and book tickets without needing to visit the theater in person.

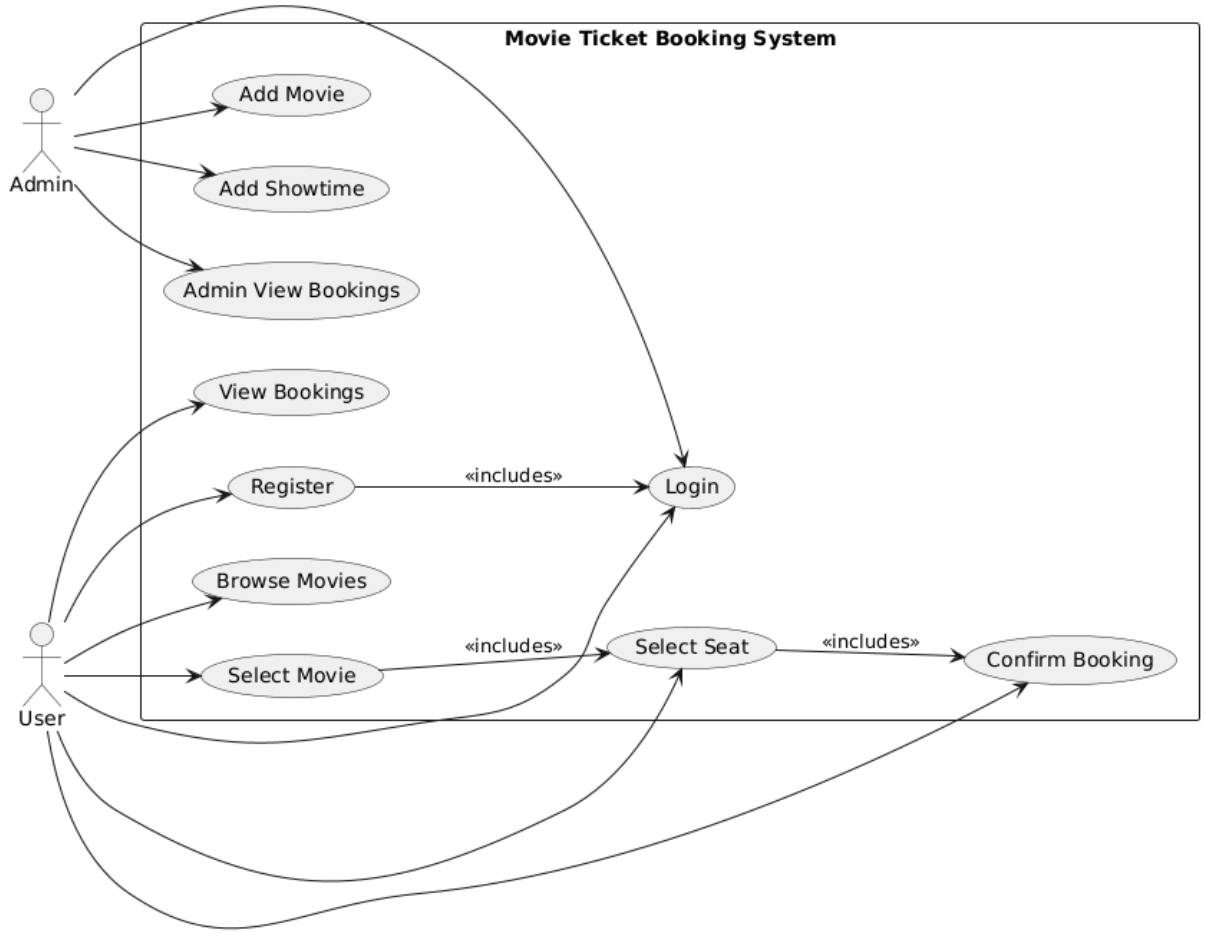
In addition to improving the booking process, the system offers features like the ability to view past bookings, track ticket history, and manage reservations, which enhances transparency and convenience for users. The integration of these features makes it possible for users to have a more personalized movie-watching experience, giving them the flexibility to modify bookings and manage their tickets as needed. On the theater's side, the automated system ensures better seat management, fewer errors, and enhanced operational efficiency. The system’s ability to handle peak booking times and its flexibility in dealing with a large volume of transactions make it a scalable and robust solution for theaters of all sizes.

UML Diagram:

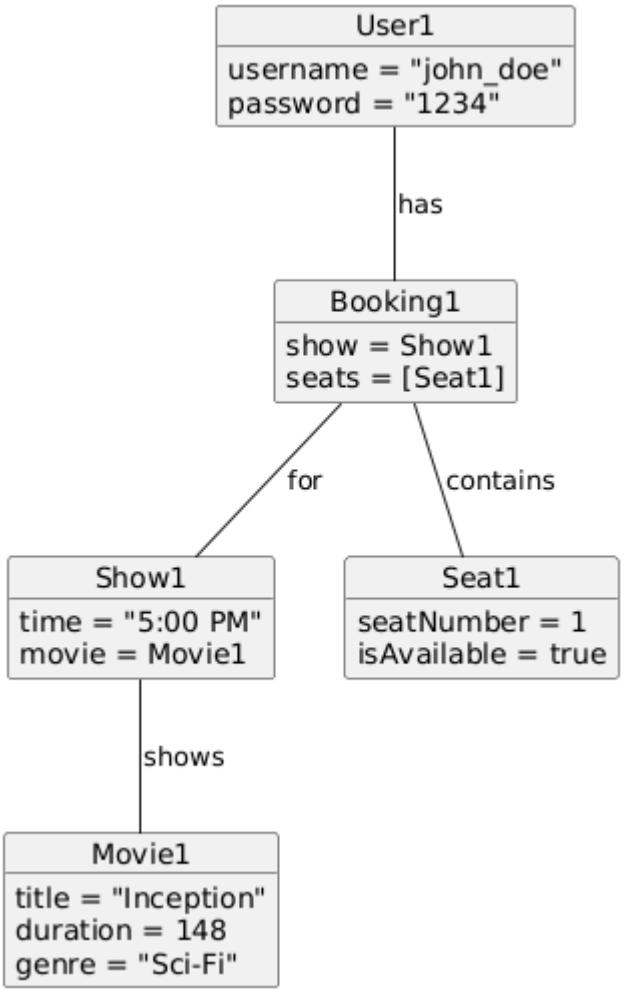
Class Diagram:



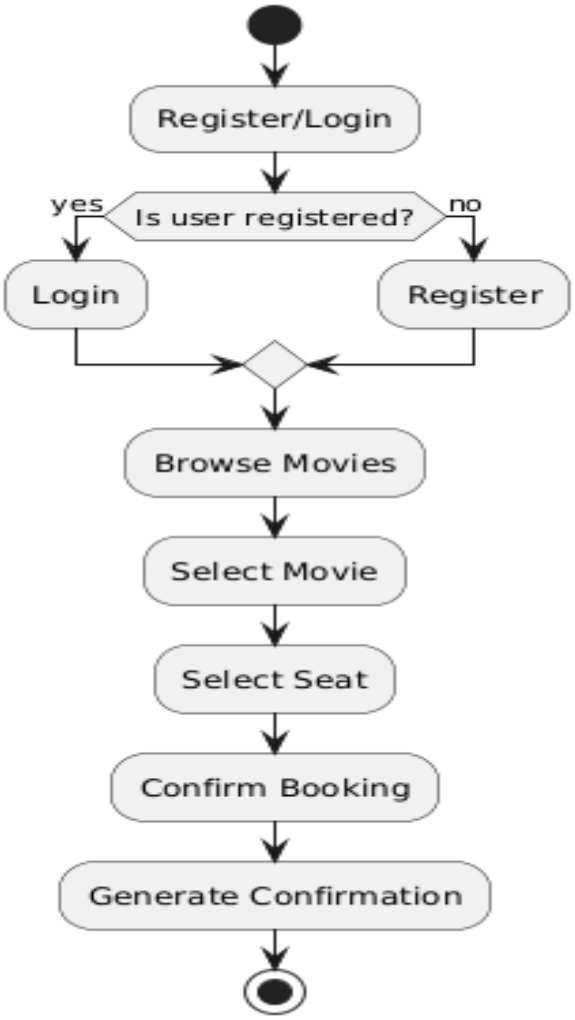
Use Case Diagram



Object Diagram



Activity Diagram



Modules of the Project

The Movie Ticket Booking System is divided into several functional modules to ensure modularity, maintainability, and ease of understanding. Each module is responsible for specific tasks and contributes to the overall functionality of the system.

1. User Module

This module manages everything related to users, such as registration, authentication, and their interaction with the system.

Features:

- User Registration:
    - Allows new users to create an account by providing a username and password.
    - Validates input and ensures usernames are unique.
- User Login:
    - Authenticates users using their registered credentials.
    - Grants access to booking functionalities upon successful login.
- User Profile Management:
    - Maintains user-specific data, including booking history.
    - Can be extended to allow password changes and profile updates.
- View Bookings:

    - Displays all bookings made by the logged-in user.
    - Provides details like movie name, show time, screen, and seat numbers.

Responsibilities:

- Authentication and authorization
- Booking history tracking
- Secure user session management

---

2. Movie Module

This module is responsible for handling all operations related to movie data.

Features:

- Movie Creation:
    - Enables admin users to add new movies to the system with attributes like title, genre, and duration.
- Movie Management:
    - Allows viewing, editing, or removing movie records (admin-side).
    - Prevents deletion of movies that are linked to scheduled shows.
- View Movie Details:

    - Users can view a list of available movies along with relevant information.
    - Acts as a browsing interface to help users decide on bookings.

Responsibilities:

- Maintain up-to-date movie database
- Provide accessible movie information for users and shows

---

3. Show Module

This module deals with scheduling and displaying movie shows.

Features:

- Show Creation:
    - Admins can create a show by associating a movie with a screen and assigning a show time.
    - Validates that the screen is available at the given time slot.
- View Available Shows:
    - Lists all upcoming shows with time, screen number, and associated movie.
    - Provides an entry point for users to initiate a booking.

Responsibilities:

- Scheduling movie shows
- Ensuring conflict-free screen assignments
- Show listing for end users

4. Booking Module

Handles the core functionality of the system—ticket booking.

Features:

- Seat Selection:
    - Displays the seating arrangement of the selected screen.
    - Allows users to choose one or more available seats for a show.
- Booking Confirmation:
    - Finalizes the booking by marking selected seats as booked.
    - Generates a confirmation with show and seat details.
- Booking History:
    - Stores and retrieves all bookings made by a user.
    - Displays comprehensive booking details for reference or cancellation.

Responsibilities:

- Ensuring atomic booking process to avoid double bookings
- Seat reservation and real-time availability update
- Handling booking logic and confirmations

5. Seat Module

Manages seats in each screen and their booking status.

Features:

- Seat Availability Management:
    - Maintains status of every seat as either Available or Booked.
    - Ensures real-time seat status is accurately displayed to users.
- Seat Reservation:
    - Marks seats as booked upon booking confirmation.
    - Releases seats if a booking is canceled or expired (optional future feature).

Responsibilities:

- Managing seat status for all shows
- Integrating with booking module to reflect live availability
- Preventing double booking of seats

CODE:

```
import java.util.*;

class Movie {
    private String title;
    private int duration;
    private String genre;

    public Movie(String title, int duration, String genre) {
        this.title = title;
        this.duration = duration;
        this.genre = genre;
    }

    public String getTitle() {
        return title;
    }

    public String toString() {
        return title + " (" + genre + ", " + duration + " mins)";
    }
}

class Seat {
    private int seatNumber;
    private boolean isAvailable;
```

```
    public Seat(int seatNumber) {
        this.seatNumber = seatNumber;
        this.isAvailable = true;
    }

    public int getSeatNumber() {
        return seatNumber;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void book() {
        isAvailable = false;
    }

    public void release() {
        isAvailable = true;
    }

    public String toString() {
            return "Seat " + seatNumber + " - " + (isAvailable ? "Available" :
"Booked");
    }
}

class Screen {
```

```
    private int screenNumber;
    private List<Seat> seats;

    public Screen(int screenNumber, int totalSeats) {
        this.screenNumber = screenNumber;
        this.seats = new ArrayList<>();
        for (int i = 1; i <= totalSeats; i++) {
            seats.add(new Seat(i));
        }
    }

    public List<Seat> getSeats() {
        return seats;
    }

    public int getScreenNumber() {
        return screenNumber;
    }
}

class Show {
    private Movie movie;
    private Screen screen;
    private String time;

    public Show(Movie movie, Screen screen, String time) {
        this.movie = movie;
        this.screen = screen;
        this.time = time;
```

```
    }

    public Movie getMovie() {
        return movie;
    }

    public Screen getScreen() {
        return screen;
    }

    public String getTime() {
        return time;
    }

    public String toString() {
                return movie.getTitle() + " at " + time + " on Screen " +
screen.getScreenNumber();
    }
}

class Booking {
    private Show show;
    private List<Seat> seats;

    public Booking(Show show, List<Seat> seats) {
        this.show = show;
        this.seats = seats;
    }
```

```
    public Show getShow() {
        return show;
    }

    public List<Seat> getSeats() {
        return seats;
    }

    public String toString() {
          StringBuilder sb = new StringBuilder("Booking for " + show + "\nSeats:
");
        for (Seat s : seats) {
            sb.append(s.getSeatNumber()).append(" ");
        }
        return sb.toString();
    }
}

class User {
    private String username;
    private String password;
    private List<Booking> bookings;

    public User(String username, String password) {
        this.username = username;
        this.password = password;
        this.bookings = new ArrayList<>();
    }
```

```
    public String getUsername() {
        return username;
    }

    public boolean checkPassword(String password) {
        return this.password.equals(password);
    }

    public void addBooking(Booking b) {
        bookings.add(b);
    }

    public void viewBookings() {
        if (bookings.isEmpty()) {
            System.out.println("No bookings yet.");
        } else {
            for (Booking b : bookings) {
                System.out.println(b);
            }
        }
    }
}

class UserManager {
    private List<User> users = new ArrayList<>();

    public User register(String username, String password) {
        for (User u : users) {
            if (u.getUsername().equals(username)) {
```

```
            System.out.println("Username already exists.");
            return null;
        }
    }
    User newUser = new User(username, password);
    users.add(newUser);
    System.out.println("User registered successfully.");
    return newUser;
  }

  public User login(String username, String password) {
    for (User u : users) {
        if (u.getUsername().equals(username) && u.checkPassword(password))
{
            System.out.println("Login successful!");
            return u;
        }
    }
    System.out.println("Invalid credentials.");
    return null;
  }
}

public class Main {
  private static List<Movie> movies = new ArrayList<>();
  private static List<Show> shows = new ArrayList<>();
  private static Scanner scanner = new Scanner(System.in);
  private static UserManager userManager = new UserManager();
```

```
    public static void main(String[] args) {
        setupData();

        User currentUser = null;
        while (currentUser == null) {
            System.out.println("\n1. Register\n2. Login\nChoose: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // consume newline
            System.out.print("Username: ");
            String username = scanner.nextLine();
            System.out.print("Password: ");
            String password = scanner.nextLine();

            if (choice == 1) {
                currentUser = userManager.register(username, password);
            } else if (choice == 2) {
                currentUser = userManager.login(username, password);
            }
        }

        while (true) {
                System.out.println("\n1. View Shows\n2. Book Ticket\n3. View My
Bookings\n4. Exit");
            int choice = scanner.nextInt();
            scanner.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    listShows();
```

```
                break;
            case 2:
                bookTicket(currentUser);
                break;
            case 3:
                currentUser.viewBookings();
                break;
            case 4:
                System.out.println("Goodbye!");
                return;
            default:
                System.out.println("Invalid choice.");
                break;
        }
    }
}

private static void listShows() {
    System.out.println("\nAvailable Shows:");
    for (int i = 0; i < shows.size(); i++) {
        System.out.println((i + 1) + ". " + shows.get(i));
    }
}

private static void bookTicket(User user) {
    listShows();
    System.out.print("Choose a show: ");
    int showIndex = scanner.nextInt() - 1;
    if (showIndex < 0 || showIndex >= shows.size()) {
```

```
    System.out.println("Invalid choice.");
    return;
}

Show selectedShow = shows.get(showIndex);
List<Seat> seats = selectedShow.getScreen().getSeats();

System.out.println("\nAvailable Seats:");
for (Seat seat : seats) {
    System.out.println(seat);
}

System.out.print("Enter seat numbers (comma separated): ");
scanner.nextLine();
String[] input = scanner.nextLine().split(",");

List<Seat> selectedSeats = new ArrayList<>();
for (String s : input) {
    int seatNum = Integer.parseInt(s.trim());
    if (seatNum < 1 || seatNum > seats.size()) {
        System.out.println("Invalid seat: " + seatNum);
        continue;
    }

    Seat seat = seats.get(seatNum - 1);
    if (seat.isAvailable()) {
        seat.book();
        selectedSeats.add(seat);
    } else {
```

```
                System.out.println("Seat " + seatNum + " already booked.");
            }
        }

        if (!selectedSeats.isEmpty()) {
            Booking booking = new Booking(selectedShow, selectedSeats);
            user.addBooking(booking);
            System.out.println("Booking Confirmed:\n" + booking);
        } else {
            System.out.println("No seats booked.");
        }
    }

    private static void setupData() {
        Movie m1 = new Movie("Inception", 148, "Sci-Fi");
        Movie m2 = new Movie("Titanic", 195, "Romance");
        movies.add(m1);
        movies.add(m2);

        Screen s1 = new Screen(1, 10);
        Screen s2 = new Screen(2, 10);

        shows.add(new Show(m1, s1, "5:00 PM"));
        shows.add(new Show(m2, s2, "8:00 PM"));
    }
}
```

OUTPUT:

```
1. Register
2. Login
Choose:
1
Username: varsha
Password: varshh
User registered successfully.

1. View Shows
2. Book Ticket
3. View My Bookings
4. Exit
1
```

```
Available Shows:
1. Inception at 5:00 PM on Screen 1
2. Titanic at 8:00 PM on Screen 2

1. View Shows
2. Book Ticket
3. View My Bookings
4. Exit
```

30

Applications of the Project

The Movie Ticket Booking System is a versatile application that can be used in a variety of real-world scenarios, not limited to cinemas alone. Its modular design and functionality make it adaptable to a wide range of domains where reservation and seat management are key components.

1. Movie Theaters and Multiplexes

- The primary use of this system is in **movie theaters**, where it digitizes and automates the ticket booking process.
- It allows theater owners to **manage show timings, assign movies to screens**, and track bookings efficiently.
- Users benefit from **real-time seat selection**, instant booking confirmation, and the ability to manage their bookings online.
- This eliminates the need for physical counters and reduces the workload on staff, improving customer service.

2. Event Management Systems

- The system architecture can be **easily adapted for other events** such as concerts, plays, stand-up comedy shows, or sports matches.
- Just like movie shows, these events also require **time-based scheduling, venue management, and seat booking**, making this system highly reusable.

- Organizers can use the platform to **list events, sell tickets, manage seat plans**, and monitor sales in real time.

3. Transportation Services

- With minimal modifications, this system can be used for **bus, train, or flight ticket booking**.

- Seat selection, booking history, and user authentication modules can be retained as-is, with only the data model and interface adjusted for travel routes and timings.

- This makes it useful for **intercity bus operators or small travel agencies** looking for an affordable booking solution.

4. Educational Institution Event Booking

- Colleges and universities often host events (seminars, workshops, festivals) that require **controlled entry with seat limitations**.

- The system can serve as an **internal platform** for students and staff to **register, reserve seats, and get confirmation**.

- It brings professionalism and organization to campus-level event planning.

5. Corporate Conference and Meeting Room Booking

- Within companies or co-working spaces, this system can be used to **book meeting rooms or auditoriums** for events or internal presentations.

- Admins can schedule slots, and employees can **view available rooms and time slots** before confirming a booking.
  This brings order to shared resource usage and avoids conflicts.

6. Mobile Ticket Booking Applications

- This system provides a **backend foundation** for developing a mobile app for ticket booking.

- APIs can be developed on top of this system to allow seamless integration with **Android/iOS applications**.

- Features like push notifications, QR-code-based ticket validation, and digital wallets can be added.

7. E-commerce Integration for Movie Merchandise

- The system can be expanded into an **e-commerce platform** by integrating merchandise sales such as movie posters, T-shirts, or collectibles.

- Users booking a movie can be shown related products during checkout, encouraging **upselling and marketing**.

Limitations of the Project:

1. **Limited User Interactions**: The current system lacks some advanced features like seat selection maps, user reviews, and ratings.
2. **Scalability**: The system is not designed to handle very large datasets or a large number of concurrent users.
3. **Single Screen Support**: The system only supports simple showings, without support for multi-screen booking or complex show managemen
4. **Payment Integration**: The system does not include payment gateway integration for processing transactions.

Bibliography:

1. “Design Patterns: Elements of Reusable Object-Oriented Software” by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
2. “UML Distilled: A Brief Guide to the Standard Object Modeling Language” by Martin Fowler
3. “Head First Design Patterns” by Eric Freeman, Elisabeth Robso
4. “Effective Java” by Joshua Bloch
5. Java API Documentation (https://docs.oracle.com/en/java/javase/)

Github Link :