MOVIE TICKET BOOKING SYSTEM

Project submitted to the
SRM University– AP, Andhra Pradesh
for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology/Master of Technology
In
Computer Science and Engineering
School of Engineering and Sciences

Submitted By:

AP23110010540-Shriya Rao Balivada
AP23110010550-Bathula Venkata Asha
AP23110010567-Kalisetty Manjari
AP23110010506-Aravelli Jahnavi Jyotirmayi

Under the guidance of:

Dr. Kavitha Rani Karnena

SRM University–AP
Neerukonda, Mangalagiri, Guntur
Andhra Pradesh– 522 240
[November 2024]

Certificate

Date: 20-Nov-24

This is to certify that the work present in this Project entitled "MOVIE TICKET BOOKING SYSTEM" has been carried out by **[Shriya Rao Balivada, Jahnavi Jyotirmayi, Venkata Asha Bathula, Kalisetty Manjari]** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University– AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences.**

Supervisor

Dr. Kavitha Rani Karnena

(Signature)

Prof. / Dr. [Kavitha Rani Karnena]

Acknowledgements

We take immense pleasure in presenting our project, the Movie Ticket Booking System, which has been a culmination of teamwork, dedication, and guidance.

First and foremost, we extend our heartfelt gratitude to our esteemed project guide, **Prof. Kavitha Rani Karnena**, whose invaluable mentorship, technical insights, and constant encouragement steered us throughout the development process. Their constructive feedback and timely suggestions helped us navigate challenges and refine our work to achieve our desired objectives.

We are deeply grateful to our team members for their relentless efforts, unwavering commitment, and collaborative spirit. Each member brought unique skills, perspectives, and ideas that were instrumental in shaping this project. The mutual support, effective communication, and seamless coordination within the team played a pivotal role in overcoming obstacles and achieving success.

We would also like to acknowledge the numerous brainstorming sessions, late-night discussions, and meticulous testing efforts that went into making the Movie Ticket Booking System a functional, user-friendly, and efficient application. It is the collective dedication of the team that transformed this project from an idea into reality.

Finally, we appreciate the collective energy and passion that drove us to deliver this project successfully. The experience has not only enhanced our technical acumen but also strengthened our collaborative and problem-solving abilities, which we will carry forward in future endeavors.

Thank you once again to our guide and the team members who made this journey memorable and fulfilling.

Table of Contents

Abstract

The Movie Ticket Booking System is a software application designed to simplify and enhance the process of reserving movie tickets, providing a reliable and efficient solution to the limitations of traditional methods such as manual bookings and long waiting times. Developed entirely in C++, the system demonstrates the power and versatility of this language in creating robust and efficient applications.

The system enables users to explore movie listings, view schedules, select preferred show timings and seats, and confirm bookings through a streamlined process. Key features include real-time seat availability tracking, automated booking confirmations, and error handling mechanisms, all implemented using C++'s advanced capabilities such as file handling, data structures, and object-oriented programming.

The development process involved leveraging C++ to create a modular, scalable, and efficient solution. Data management was achieved using file handling techniques, ensuring secure and organized storage of user and booking information. The system's architecture is designed to be both user-friendly and performance-oriented, optimizing for speed and accuracy.

This report outlines the step-by-step design, implementation, and testing of the system. It evaluates the software's efficiency through practical scenarios, highlighting its ability to handle user inputs and transactions seamlessly.

In conclusion, the Movie Ticket Booking System showcases the application of C++ in solving real-world problems, providing a fast, reliable, and user-centric solution for the movie ticket booking process. This project stands as a testament to the language's capability to deliver high-performance solutions for complex tasks.

Introduction

The Movie Ticket Booking System is a command-line application designed to simplify the process of reserving seats for movies. Developed in C++, it provides an interactive menu-driven interface that guides users through selecting movies, theaters, show timings, seating classes, and specific seats. By leveraging Object-Oriented Programming (OOP) principles such as inheritance and polymorphism, the system ensures a modular and efficient design. It dynamically calculates ticket prices based on seating class (Platinum or Gold) and incorporates a robust mechanism to check and prevent seat booking conflicts. This application offers a realistic simulation of a movie ticket booking process, demonstrating core C++ concepts, including the use of templates, maps, and string manipulations.

The system is intended for showcasing the practical application of C++ programming in solving real-world problems. With a focus on simplicity and functionality, the system avoids complexities like database integration or graphical interfaces, relying solely on runtime logic and user-based inputs. It effectively handles edge cases such as invalid inputs, unavailable movies, and conflicting seat bookings, ensuring a smooth user experience.

Methodology

This project implements a movie ticket booking system using object-oriented programming in C++. The system provides an interactive console-based interface for users to book tickets for movies based on available options. The methodology followed includes:

1. **Requirement Analysis**
    - Identify the core features: movie availability check, theater selection, timings, seat booking, and class categorization (Platinum/Gold).
    - Address constraints such as seat availability, booking overlaps, and class-specific features.
2. **Design**
    - Use classes to encapsulate the system's core functionalities:
        - ➢ **Movie**: Represents a movie with a name and provides a static method to check availability.
        - ➢ **Ticket**: Abstract class to handle seat class types with derived classes for Platinum and Gold tickets.
3. **Implementation**
    - Built interactive menus to guide users through options and validate input to avoid errors.
    - Implemented functionality to:
        - ➢ Display available and upcoming movies.
        - ➢ Allow users to select theaters, timings, and seat classes.
        - ➢ Check for seat availability and prevent duplicate bookings.
    - Used templates and utility functions like displayOptions for reusable, modular code.
    - Incorporated cost calculation based on seat class and number of tickets.

4. **Testing and Validation**

   - Tested various scenarios including valid and invalid inputs, overlapping seat bookings, and edge cases for date and seat range.

   - Verified accurate cost calculation and adherence to user preferences for seating.

5. **User Experience**

   - Provided detailed prompts and summaries to enhance user understanding and interaction.

   - Added disclaimers for certain rows (e.g., rows near the screen) to simulate real-life seat constraints.

By following this methodology, we created a robust and user-friendly movie ticket booking system that addresses real-world challenges such as seat conflicts and cost calculation.

Code

```
#include <bits/stdc++.h>
using namespace std;

string toLowerCase(const string& str) {
    string lowerStr = str;
    transform(lowerStr.begin(), lowerStr.end(), lowerStr.begin(), ::tolower);
    return lowerStr;
}

class Movie {
public:
    string name;
    Movie(const string& name) : name(name) {}
    static bool isMovieAvailable(const vector<Movie>& movies, const string&
movieName) {
        string lowerMovieName = toLowerCase(movieName);
        for (const auto& movie : movies) {
            if (toLowerCase(movie.name) == lowerMovieName) {
                return true;
            }
        }
        return false;
    }
};

class Ticket {
protected:
    int ticketPrice;
    int maxSeats;
public:
    virtual int calculateCost(int numTickets) const = 0;
    int getMaxSeats() const { return maxSeats; }
    virtual ~Ticket() {}
};

class PlatinumTicket : public Ticket {
public:
    PlatinumTicket() { ticketPrice = 275; maxSeats = 15; }
    int calculateCost(int numTickets) const override { return ticketPrice * numTickets; }
};

class GoldTicket : public Ticket {
public:
    GoldTicket() { ticketPrice = 195; maxSeats = 16; }
    int calculateCost(int numTickets) const override { return ticketPrice * numTickets; }
};

template <typename T>
void displayOptions(const vector<T>& options) {
    for (size_t i = 0; i < options.size(); ++i) {
        cout << i + 1 << ". " << options[i] << endl;
    }
}

struct Booking {
    string date;
    string theater;
    string timing;
    string movie;
    set<string> bookedSeats;
};

vector<Booking> bookings;

Booking* findBooking(const string& date, const string& theater, const string& timing,
const string& movie) {
    for (auto& booking : bookings) {
        if (booking.date == date && booking.theater == theater && booking.timing ==
timing && booking.movie == movie) {
            return &booking;
        }
    }
    return nullptr;
}

int main() {
    vector<Movie> movies = {Movie("Lucky Baskar"), Movie("Amaran"), Movie("Ka")};
    vector<string> theaters = {"Mythri Cinemas", "Plateno Cinemas", "Cine Square"};
    vector<string> timings = {"10:00 AM", "2:00 PM", "6:00 PM", "9:00 PM"};

    while (true) {
        cout << "\n** Movie Tickets **" << endl;
        cout << "Press 1 to check available movies." << endl;
        cout << "Press 2 to check upcoming movies." << endl;
        cout << "Press 3 to exit." << endl;
```

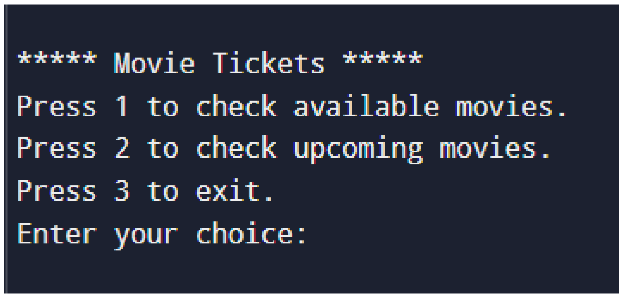
```
int choice;
cout << "Enter your choice: ";
cin >> choice;
cin.ignore();

if (choice == 1) {
    cout << "Available Movies:\n";
    for (const auto& movie : movies) cout << "- " << movie.name << endl;

    cout << "\nEnter the movie name you want to book: ";
    string movieName;
    getline(cin, movieName);

    if (Movie::isMovieAvailable(movies, movieName)) {
        cout << "Enter the date (18 to 24 November): ";
        int date;
        cin >> date;
        if (date < 18 || date > 24) {
            cout << "Invalid date!" << endl;
            continue;
        }
        string dateStr = to_string(date);

        cout << "\nSelect a theater:\n";
        displayOptions(theaters);
        int theaterChoice;
        cin >> theaterChoice;
        if (theaterChoice < 1 || theaterChoice > theaters.size()) {
            cout << "Invalid theater choice!" << endl;
            continue;
        }
        string selectedTheater = theaters[theaterChoice - 1];

        cout << "\nAvailable show timings:\n";
        displayOptions(timings);
        int timingChoice;
        cin >> timingChoice;
        if (timingChoice < 1 || timingChoice > timings.size()) {
            cout << "Invalid timing choice!" << endl;
            continue;
        }
        string selectedTiming = timings[timingChoice - 1];

        cout << "\nChoose class (Platinum/Gold): ";
        string seatClass;
        cin >> seatClass;

        Ticket* ticket;
        string selectedRow;
        if (toLowerCase(seatClass) == "platinum") {
            ticket = new PlatinumTicket();
            selectedRow = "A";
        } else if (toLowerCase(seatClass) == "gold") {
            ticket = new GoldTicket();
            vector<string> rows = {"B", "C", "D", "E", "F", "G", "H", "I", "J", "K"};
            while (true) {
                cout << "\nSelect a row (By entering the number):\n";
                displayOptions(rows);
```

```
                int rowChoice;
                cin >> rowChoice;
                if (rowChoice < 1 || rowChoice > rows.size()) {
                    cout << "Invalid row choice! Try again." << endl;
                    continue;
                }
                selectedRow = rows[rowChoice - 1];

                if (selectedRow == "I" || selectedRow == "J" || selectedRow == "K") {
                    cout << "Disclaimer: This row is very near to the screen. Proceed?
(yes/no): ";
                    string proceed;
                    cin >> proceed;
                    if (toLowerCase(proceed) != "yes") {
                        cout << "Please select a different row.\n";
                        continue;
                    }
                }
                break;
            }
        } else {
            cout << "Invalid class!" << endl;
            continue;
        }

        int startSeat, endSeat;
        while (true) {
            cout << "Enter the seat number or range (e.g., 4 for a single seat or 4 10 for a
range): ";
```

```
                cin >> startSeat;
                if (cin.peek() == '\n') {
                    endSeat = startSeat;
                } else {
                    cin >> endSeat;
                }

                if (startSeat < 1 || endSeat > ticket->getMaxSeats() || startSeat > endSeat) {
                    cout << "Invalid seat range! Please enter a valid range within 1 to " <<
ticket->getMaxSeats() << "." << endl;
                    continue;
                }

                Booking* existingBooking = findBooking(dateStr, selectedTheater,
selectedTiming, movieName);

                if (!existingBooking) {
                    bookings.push_back({dateStr, selectedTheater, selectedTiming,
movieName, {}});
                    existingBooking = &bookings.back();
                }

                bool seatsAvailable = true;
                for (int seat = startSeat; seat <= endSeat; ++seat) {
                    string seatId = selectedRow + to_string(seat);
                    if (existingBooking->bookedSeats.count(seatId)) {
                        seatsAvailable = false;
                        break;
                    }
                }

                if (!seatsAvailable) {
                    cout << "Some seats in the range are already booked for this date, theater,
and timing. Please try another range." << endl;
                    continue;
                }

                for (int seat = startSeat; seat <= endSeat; ++seat) {
                    existingBooking->bookedSeats.insert(selectedRow + to_string(seat));
                }
                break;
            }

            int numTickets = endSeat - startSeat + 1;
            int totalCost = ticket->calculateCost(numTickets);
            cout << "\nBooking Summary:\n";
            cout << "Movie: " << movieName << endl;
            cout << "Date: " << date << " November" << endl;
            cout << "Theater: " << selectedTheater << endl;
            cout << "Timing: " << selectedTiming << endl;
            cout << "Class: " << seatClass << endl;
            cout << "Row: " << selectedRow << endl;
            cout << "Seats: " << startSeat;
            if (startSeat != endSeat) cout << " to " << endSeat;
            cout << endl;
            cout << "Total Cost: " << totalCost << " rupees" << endl;

            delete ticket;
            } else {
                cout << "Movie not found!" << endl;
            }
        } else if (choice == 2) {
            cout << "Upcoming Movies:\n";
            cout << "1. Pushpa - 5, Dec 2024\n2. Mufasa - 20, Dec 2024\n3. Game Changer -
10, Jan 2025\n" << endl;
        } else if (choice == 3) {
            break;
        } else {
            cout << "Invalid choice!" << endl;
        }
    }
    return 0;
}
```

Outputs:

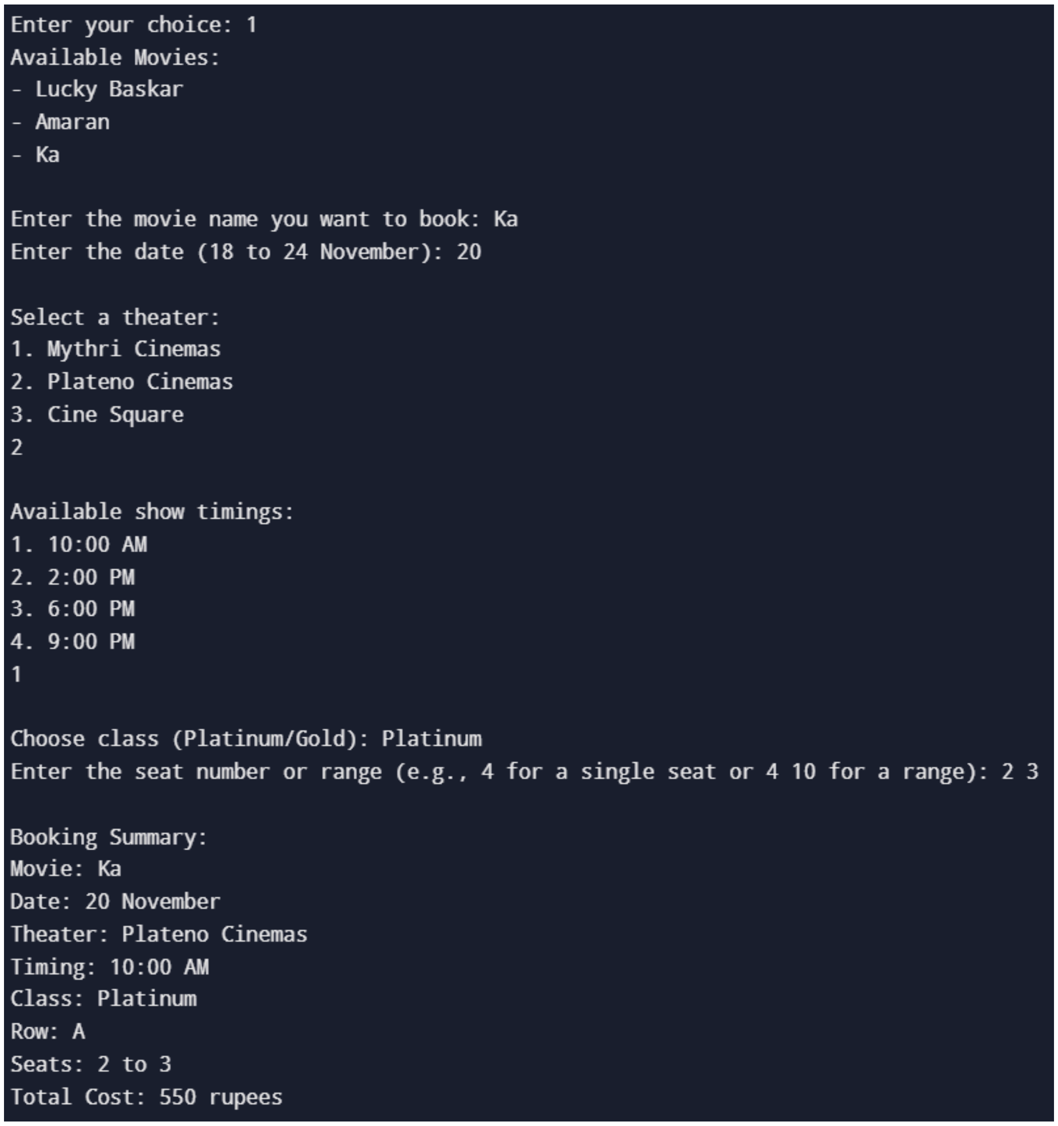
When the code runs, the following is shown:



```
***** Movie Tickets *****
Press 1 to check available movies.
Press 2 to check upcoming movies.
Press 3 to exit.
Enter your choice:
```

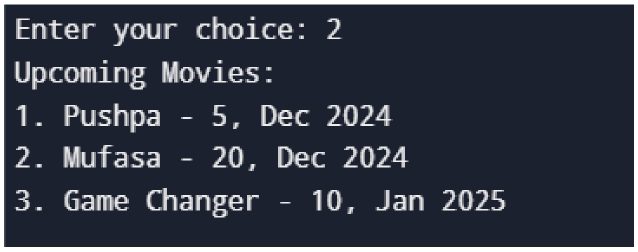
The following are the cases that the code displays:

Case 1: User chooses 1.



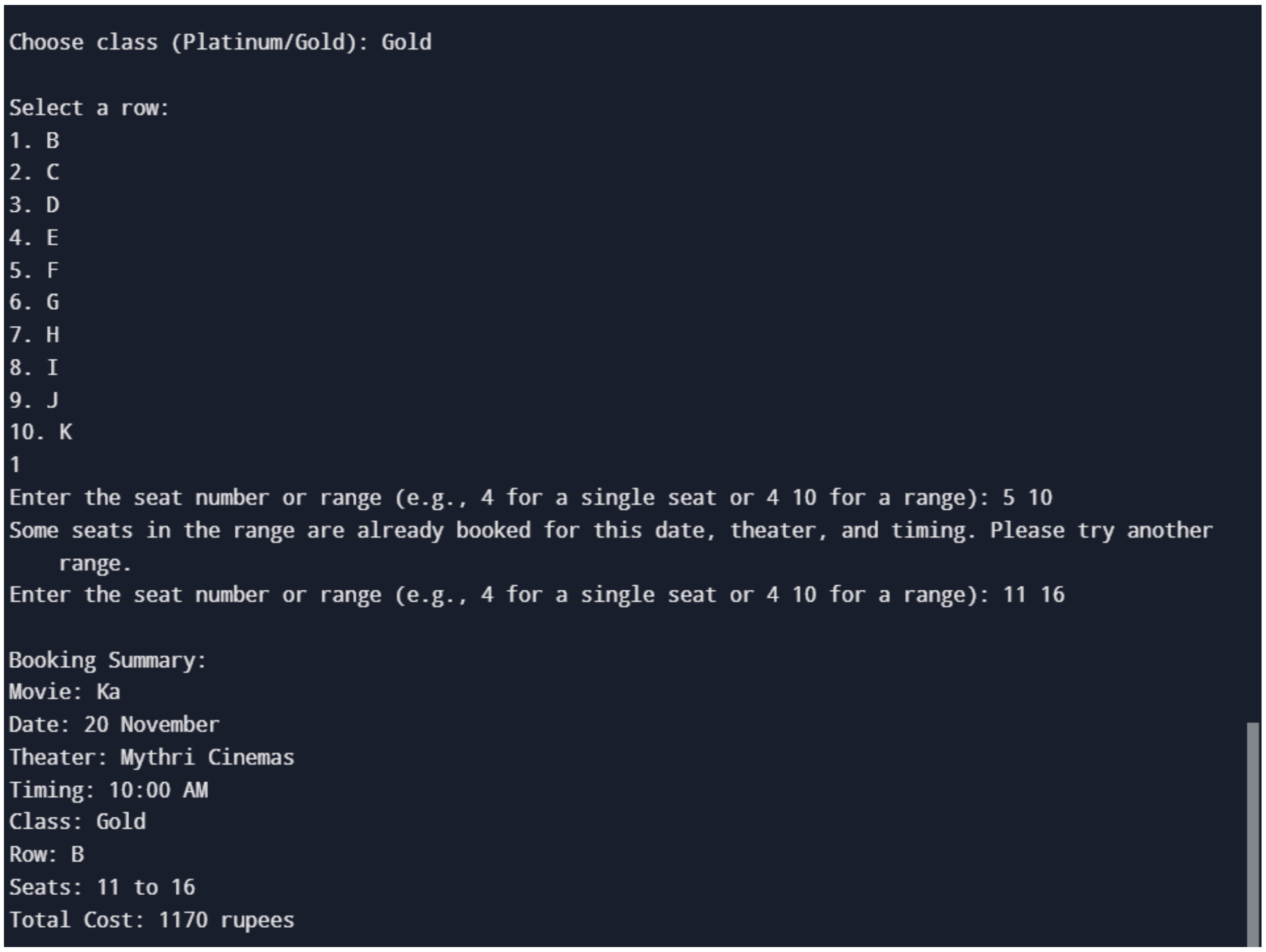
```
Enter your choice: 1
Available Movies:
- Lucky Baskar
- Amaran
- Ka

Enter the movie name you want to book: Ka
Enter the date (18 to 24 November): 20

Select a theater:
1. Mythri Cinemas
2. Plateno Cinemas
3. Cine Square
2

Available show timings:
1. 10:00 AM
2. 2:00 PM
3. 6:00 PM
4. 9:00 PM
1

Choose class (Platinum/Gold): Platinum
Enter the seat number or range (e.g., 4 for a single seat or 4 10 for a range): 2 3

Booking Summary:
Movie: Ka
Date: 20 November
Theater: Plateno Cinemas
Timing: 10:00 AM
Class: Platinum
Row: A
Seats: 2 to 3
Total Cost: 550 rupees
```

Case 2: If Uses presses 2.



```
Enter your choice: 2
Upcoming Movies:
1. Pushpa - 5, Dec 2024
2. Mufasa - 20, Dec 2024
3. Game Changer - 10, Jan 2025
```

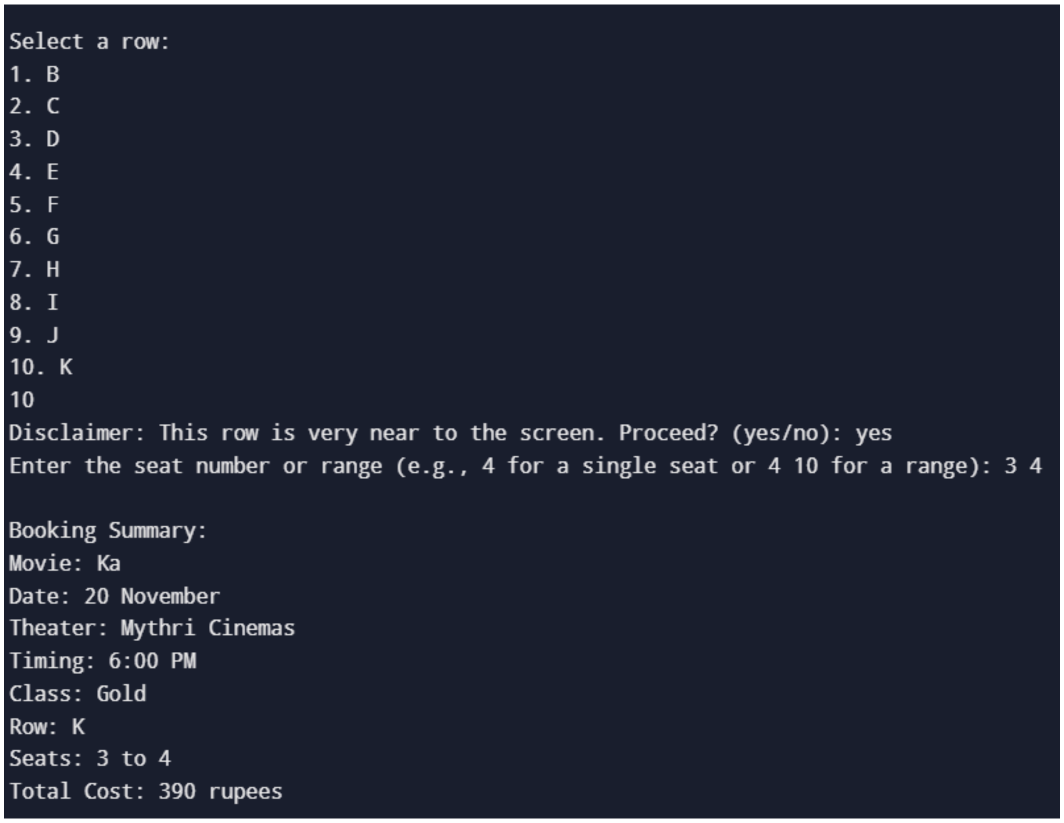
Case 3: If user books the same seats in the same theater, show timings and movie.



```
Choose class (Platinum/Gold): Gold

Select a row:
1. B
2. C
3. D
4. E
5. F
6. G
7. H
8. I
9. J
10. K
1
Enter the seat number or range (e.g., 4 for a single seat or 4 10 for a range): 5 10
Some seats in the range are already booked for this date, theater, and timing. Please try another
    range.
Enter the seat number or range (e.g., 4 for a single seat or 4 10 for a range): 11 16

Booking Summary:
Movie: Ka
Date: 20 November
Theater: Mythri Cinemas
Timing: 10:00 AM
Class: Gold
Row: B
Seats: 11 to 16
Total Cost: 1170 rupees
```

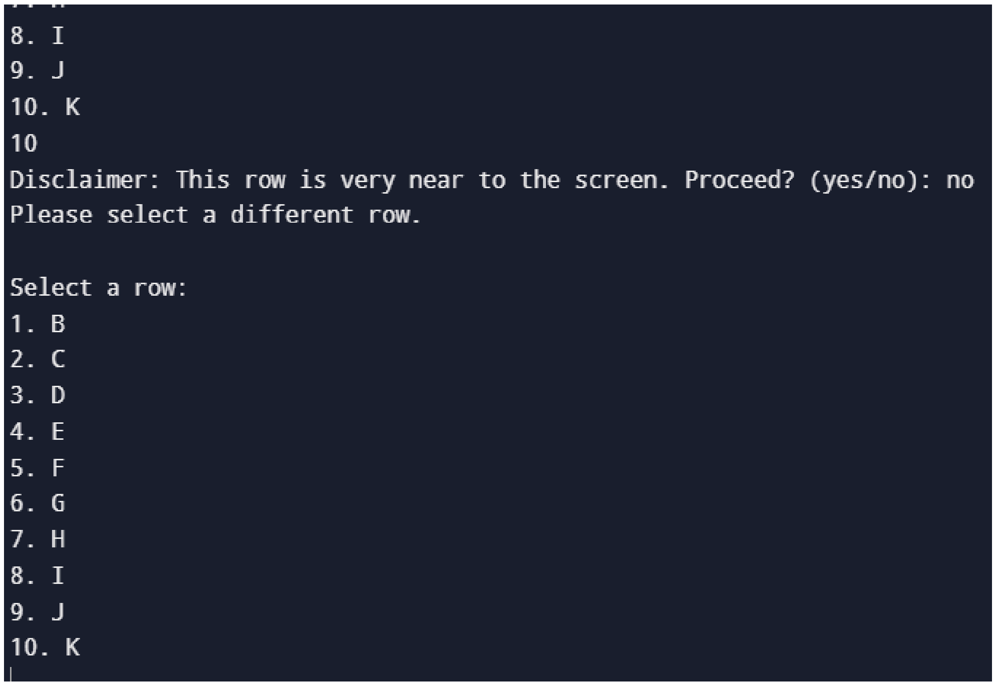
Case 4: If user selects a row closer to the screen, they have 2 options:

- Option 1: They choose Yes.



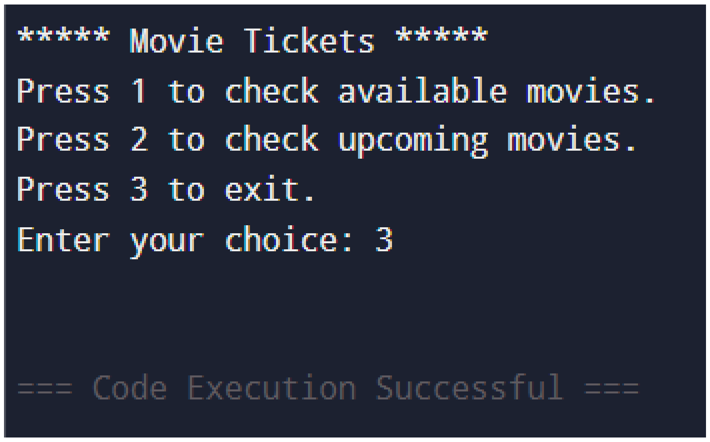
```
Select a row:
1. B
2. C
3. D
4. E
5. F
6. G
7. H
8. I
9. J
10. K
10
Disclaimer: This row is very near to the screen. Proceed? (yes/no): yes
Enter the seat number or range (e.g., 4 for a single seat or 4 10 for a range): 3 4

Booking Summary:
Movie: Ka
Date: 20 November
Theater: Mythri Cinemas
Timing: 6:00 PM
Class: Gold
Row: K
Seats: 3 to 4
Total Cost: 390 rupees
```

- Option 2: They choose No.



```
8. I
9. J
10. K
10
Disclaimer: This row is very near to the screen. Proceed? (yes/no): no
Please select a different row.

Select a row:
1. B
2. C
3. D
4. E
5. F
6. G
7. H
8. I
9. J
10. K
```

Case 5: User selects 3 to exit.



```
***** Movie Tickets *****
Press 1 to check available movies.
Press 2 to check upcoming movies.
Press 3 to exit.
Enter your choice: 3


=== Code Execution Successful ===
```

Conclusion

In conclusion, the movie ticket booking system effectively integrates key programming concepts such as object-oriented design, input validation, and cost calculation to provide a functional and user-friendly platform. By implementing classes like Movie, Ticket, and derived classes PlatinumTicket and GoldTicket, the system allows for seamless management of movie bookings, ensuring that users can easily check movie availability, select showtimes, and choose seat classes. The system's use of a map to track booked seats prevents double booking, enhancing reliability and improving the user experience.

While the system functions well within its defined scope, there is potential for further enhancements, such as refining the user interface, expanding the system to handle more complex booking scenarios, or introducing additional features like payment processing or movie ratings. Overall, this project showcases a solid application of programming fundamentals and provides a scalable foundation for developing more sophisticated booking systems in the future.