UCS2313 : OBJECT ORIENTED PROGRAMMING LANGUAGE
MOVIE TICKET BOOKING SYSTEM

PRITHIVIRAJAN D (3122225001099)

RISHAB S (3122225001104)

RITHEKHA K (3122225001106)

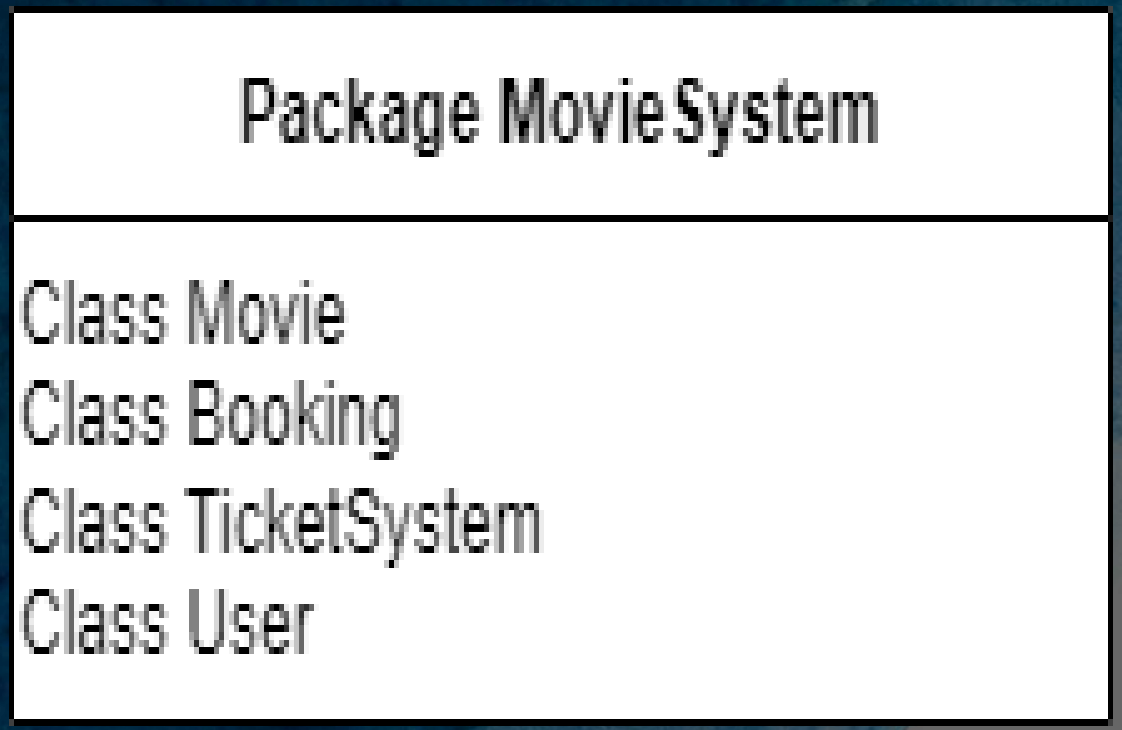
PROBLEM STATEMENT:

To design a Movie Ticket Booking System that facilitates seamless and efficient ticket reservations for customers. The system should include functionalities such as displaying a list of available movies, showing their respective schedules and seat availability, allowing users to select seats, and completing the booking process. Additionally, the system should incorporate features for user authentication, ensuring secure transactions, and managing reservations. The objective is to create a user-friendly interface that enhances the overall movie-going experience, streamlining the ticket booking process and providing a reliable platform for both customers and administrators.

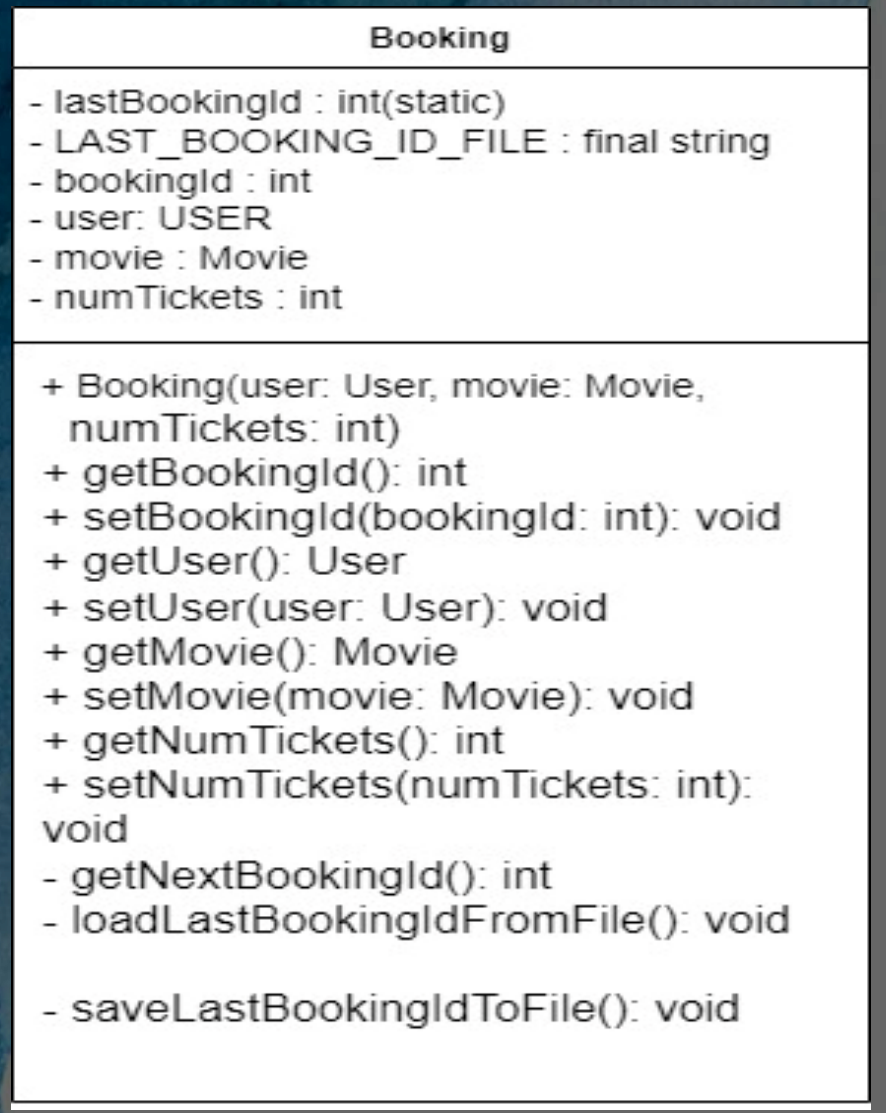
OBJECTIVE :

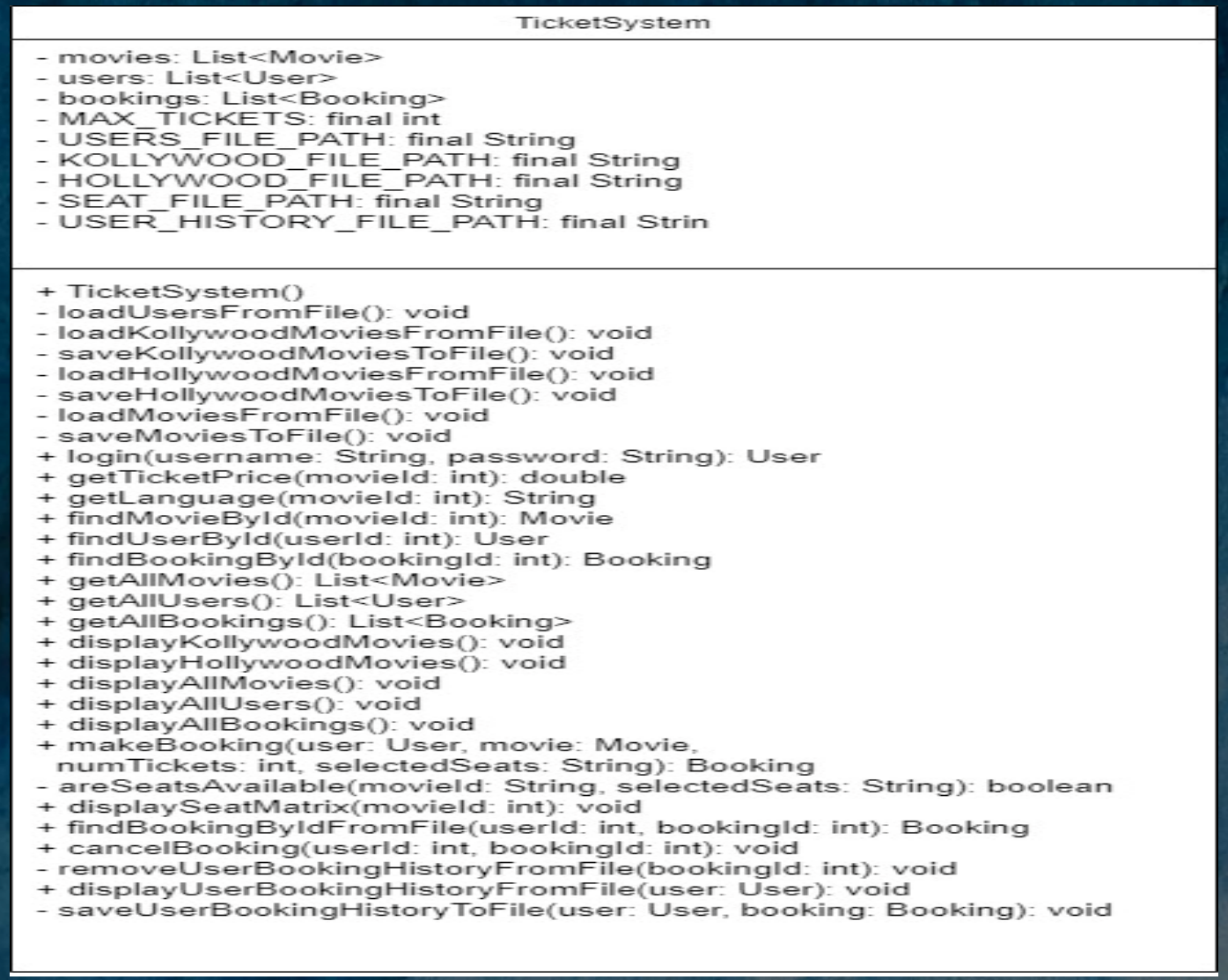
The objective behind developing a movie ticket booking system stems from the need to modernize and simplify the process of purchasing movie tickets. Traditional ticketing methods can be time-consuming and inconvenient, often leading to long queues and potential customer dissatisfaction. By implementing a solution leveraging technology, the aim is to provide a more user-friendly and efficient platform for movie enthusiasts. This system enhances the customer experience by offering a streamlined ticket reservation process and contributes to the overall advancement of the entertainment industry by embracing digital solutions. Additionally, it allows for better management of cinema resources, ensuring optimized seat occupancy and improved operational efficiency for the users.

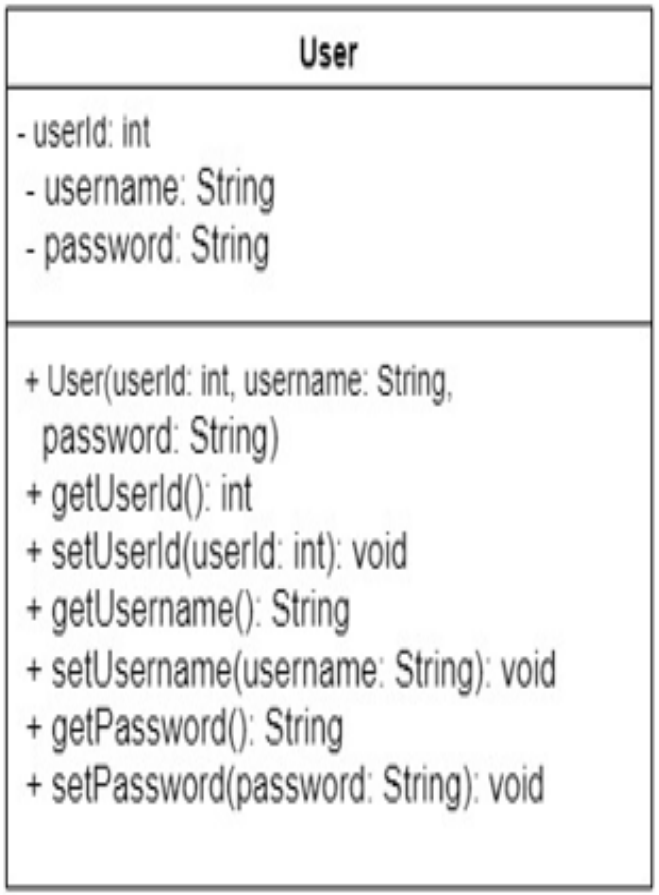
CLASS DIAGRAM :




Movie
- id: int
- title: String
- genre: String
- duration: int
- availableSeats: int
- ticketPrice: double
- language: String
- ticketSystem: TicketSystem
+ Movie(ticketSystem: TicketSystem, id: int, title: String, genre: String, duration: int, availableSeats: int, ticketPrice: double, language: String)
+ getId(): int
+ setId(id: int): void
+ getTitle(): String
+ setTitle(title: String): void
+ getGenre(): String
+ setGenre(genre: String): void
+ getDuration(): int
+ setDuration(duration: int): void
+ getAvailableSeats(): int
+ setAvailableSeats(availableSeats: int): void
+ updateAvailableSeats(numTickets: int): void
+ getTicketPrice(): double
+ getLanguage(): String
+ saveToFile(): void
+ saveToFile_h(): void


Booking
- lastBookingId : int(static)
- LAST_BOOKING_ID_FILE : final string
- bookingId : int
- user: USER
- movie : Movie
- numTickets : int
+ Booking(user: User, movie: Movie, numTickets: int)
+ getBookingId(): int
+ setBookingId(bookingId: int): void
+ getUser(): User
+ setUser(user: User): void
+ getMovie(): Movie
+ setMovie(movie: Movie): void
+ getNumTickets(): int
+ setNumTickets(numTickets: int): void
- getNextBookingId(): int
- loadLastBookingIdFromFile(): void
- saveLastBookingIdToFile(): void


TicketSystem
- movies: List<Movie>
- users: List<User>
- bookings: List<Booking>
- MAX_TICKETS: final int
- USERS_FILE_PATH: final String
- KOLLYWOOD_FILE_PATH: final String
- HOLLYWOOD_FILE_PATH: final String
- SEAT_FILE_PATH: final String
- USER_HISTORY_FILE_PATH: final Strin
+ TicketSystem()
- loadUsersFromFile(): void
- loadKollywoodMoviesFromFile(): void
- saveKollywoodMoviesToFile(): void
- loadHollywoodMoviesFromFile(): void
- saveHollywoodMoviesToFile(): void
- loadMoviesFromFile(): void
- saveMoviesToFile(): void
+ login(username: String, password: String): User
+ getTicketPrice(movieId: int): double
+ getLanguage(movieId: int): String
+ findMovieById(movieId: int): Movie
+ findUserById(userId: int): User
+ findBookingById(bookingId: int): Booking
+ getAllMovies(): List<Movie>
+ getAllUsers(): List<User>
+ getAllBookings(): List<Booking>
+ displayKollywoodMovies(): void
+ displayHollywoodMovies(): void
+ displayAllMovies(): void
+ displayAllUsers(): void
+ displayAllBookings(): void
+ makeBooking(user: User, movie: Movie,
numTickets: int, selectedSeats: String): Booking
- areSeatsAvailable(movieId: String, selectedSeats: String): boolean
+ displaySeatMatrix(movieId: int): void
+ findBookingByIdFromFile(userId: int, bookingId: int): Booking
+ cancelBooking(userId: int, bookingId: int): void
- removeUserBookingHistoryFromFile(bookingId: int): void
+ displayUserBookingHistoryFromFile(user: User): void
- saveUserBookingHistoryToFile(user: User, booking: Booking): void


User
- userId: int
- username: String
- password: String
+ User(userId: int, username: String,
password: String)
+ getUserId(): int
+ setUserId(userId: int): void
+ getUsername(): String
+ setUsername(username: String): void
+ getPassword(): String
+ setPassword(password: String): void

MODULES SPLIT – UP:

Files Outside the Package:

1. users: Text file containing user information.
2. hollywood: Text file associated with Hollywood movies.
3. kollywood: Text file related to Kollywood movies.
4. Miniproject: Java file which contains the main .
5. seat_matrix: Text file storing seat availability and layout information.
6. userhistory: Text file to store the details of the bookings of users.
7. last_booking_id: Text file to store the previous generate BookingId(to have unique BookingIds)

Package: movie system

-Java Files:

1. Booking.java: Implementation of the booking functionality.
2. Movie.java: Handling movie-related operations.
3. TicketSystem.java: Core file managing the ticket system logic.
4. User.java: Managing user-related functionalities.

This modular breakdown separates the text files, each serving a specific purpose, from the Java files within the "movie system" package. The Java files inside the package collectively contribute to the movie ticket booking system, covering booking, movie management, ticket system logic, and user-related functionalities.

OOPS CONCEPTS USED :

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism
- Packages
- Exception Handling

OBJECT ORIENTED FEATURES USED:

Encapsulation:

Encapsulation is evident in the MovieSystem through the use of private fields, getter and setter methods. For instance, the `Movie` class encapsulates movie details like `title`, `genre`, and `ticketPrice`, allowing controlled access. Similarly, in the `Booking` class, private attributes such as `bookingId` and `selectedSeats` are encapsulated, promoting data integrity and restricting direct access. This encapsulation shields the internal implementation and enforces a clear interface for interacting with objects, enhancing code maintainability and security.

Abstraction:

Abstraction is demonstrated through the MovieSystem classes, where essential details and functionalities are encapsulated. For instance, the `Movie` class abstracts movie details, providing a high-level interface to interact with movie data. Similarly, the `TicketSystem` class encapsulates movie-related operations, promoting a simplified interaction for the user without exposing underlying complexities. Additionally, the `Booking` class abstracts the concept of a booking, hiding implementation details while providing essential booking information. This approach enhances code modularity and comprehensibility.

Inheritance:

The `TicketSystem` class uses composition to manage a list of `Movie` objects. While it doesn't directly inherit from the `Movie` class, it contains and manipulates instances of `Movie`. In methods like `loadKollywoodMoviesFromFile` and `loadHollywoodMoviesFromFile`, it creates `Movie` objects based on file data. This design represents a has-a relationship, where `TicketSystem` "has a" list of `Movie` objects for handling movie-related functionalities

Polymorphism:

Polymorphism is evident in the code through method overloading. Methods like `makeBooking` and `areSeatsAvailable` accept different parameter types, enabling flexibility in handling various data scenarios. This allows the same method name to be reused with different parameter signatures, showcasing polymorphic behavior and enhancing code readability.

Packages:

The code is organized into a package named “MovieSystem”, enhancing code organization and modular development. The use of packages helps prevent naming conflicts, improves code readability, and provides a more structured project layout.

Exception Handling:
In the given code, exception handling is demonstrated primarily through the use of `try`, `catch`, and `finally` blocks. The methods that involve file I/O or data parsing are wrapped in try-catch blocks to handle potential exceptions. For instance, methods like `loadUsersFromFile`, `loadKollywoodMoviesFromFile`, and others use try-catch to manage exceptions such as `FileNotFoundException` or `IOException`. Additionally, there are specific catch blocks for handling `NumberFormatException` during data parsing. The `finally` block is used in some cases to ensure that resources are closed. Overall, these mechanisms enhance the robustness of the program by gracefully handling potential errors during file operations and data processing.

These features contribute to the overall design and organization of the movie ticket booking system in Java.

THANKYOU !