Cinema Ticket Booking System

System Design & Functional Rules Documentation

---

1. System Overview

The **Cinema Ticket Booking System** is a comprehensive ticketing application designed to ease movie booking experience.

Core Concepts Applied:

- Encapsulation, Inheritance, Abstraction, and Polymorphism
- Use of LinkedList and TreeMap
- CRUD operations with JSON-based data storage
- Computational logic (pricing, sorting, and seat management)
- User interactivity and error handling

---

2. Core Entities and Relationships

Entity	Description	Key Attributes	Relationships
Movie	Represents a film currently showing.	movieId, title, description, genre, duration, rating, showtimes	1 → many → Showtimes
ShowType	Represents a screening room (2D, IMAX, Director's Club).	showTypeId, seatLayout, totalSeats, price	1 → many → Seats
ShowTime	Represent a screening time for the movie.	showTimeId, movieId, showTypeId, date, time	belongs to → Movie

Seat	Represents an individual seat with an availability status.	seatId, isAvailable	belongs to → CinemaHall
User	Represents a customer.	userID, firstName, lastName, email, userType(user/admin), bookingHistory	1 → many → Bookings
Booking	Represents a single transaction.	bookingID, userID, showtimeId, selectedSeats, totalAmount, paymentStatus	many → 1 → User
Payment	Handles transaction processing.	paymentID, amount, paymentMethod, paymentStatus	1 → 1 with Booking
Ticket	Represents the final QR ticket.	ticketID, bookingID, qrCode, isValid	1 → 1 with Booking

---

3. System Logic and Workflow

User Management

- A user must register before booking. Account info is saved in users.json.
- Registration requires: Full name, Email, Password, and Contact number.
- Users can log in with their registered credentials.
- The system validates credentials during login (via simple matching).
- Passwords are not encrypted (for simplicity, in this version).
- Logged-in users can view booking history stored in bookings.json.

Movie & Showtime Management

- Movies are loaded from a JSON file.
- Each movie can have multiple showtimes.
- Each showtime belongs to a specific cinema type: 2D, Director’s Club, or IMAX.

Show Type & Seating

- Each show type (2D, IMAX, Director's Club) has its own seat layout and pricing.
- Seats are stored in a **LinkedList** to dynamically handle seat availability.
- When booked, seats are marked unavailable and saved to the JSON file.
- If a user tries to select an already booked seat, an error message appears and they must reselect.

Cinema Type	Base Ticket Price	Seat Rows	Seats per Row
2D	₱300	5	10
Director's Club	₱600	4	8
IMAX	₱800	6	12

Booking Process

1. User logs in or registers.
2. The user selects a movie, showtype, and showtime.
3. The system displays a real-time seat map.
4. The user selects seats and confirms booking.
5. The system calculates total price (including discounts and fees).
6. User proceeds to payment.
7. After booking, confirmation details are shown and stored in bookings.json.

8. The system generates a digital ticket with a unique QR code.

---

4. Payment and Ticketing Rules

- Supported payment methods: Credit/Debit Card, GCash, PayMaya, or Pay at Counter.
- **Successful payment** → Ticket generated and booking saved.
- **Failed payment** → Transaction canceled.
- **Non-refundable** unless the cinema cancels the show.

---

5. Object-Oriented Design Principles

OOP Concept	Application
Encapsulation	Private attributes with public getters and setters in all model classes.
Inheritance	`ShowType` superclass extended by `StandardCinema`, `IMAXCinema`, and `DirectorsClubCinema`.
Abstraction	Abstract class `CinemaExperience` defines shared methods (`calculatePrice()`, `displayInfo()`).
Polymorphism	Overridden methods like `calculatePrice()` for each show type.
Overloading	`bookTicket()` overloaded for single or multiple seat selections.
Overriding	Subclasses override methods like `displayDetails()` for specific behavior.

---

6. Relationships (UML Basis)

- **1:1** → Booking ↔ Payment, Booking ↔ Ticket

- **1:M** → User → Bookings, Movie → Showtimes, ShowType→ Seats
- **M:M** → Movie ↔ ShowType(via Showtimes)

---

7. Data Structures & CRUD Operations

Data Structure	Purpose
LinkedList	Manages seat availability dynamically.
TreeMap	Organizes showtimes automatically in ascending order.
JSON Files	Acts as a local database for CRUD operations.

CRUD Operations:

- **Create** → Add new booking
- **Read** → View movies, showtimes, seats
- **Update** → Modify seat availability or user info
- **Delete** → Cancel booking

---

8. Computational Logic

Function	Logic
Total Price	`(Price + Booking Fee) − Discounts`
PWD Discount	VAT exempt and 20% off
Senior Discount	VAT exempt and 20% off
Booking Fee	₱30.00 booking fee

Seat Availability Check	Seat cannot be double-booked

---

9. Error Handling Rules

- Validate all user input (seat number, payment type, etc.).
- Prevent duplicate bookings.

  Handle missing or corrupted JSON files.
- Catch invalid input formats (e.g., strings instead of numbers).
- Provide clear, friendly error messages and re-entry prompts.

---

10. User Interaction Flow

1. **Welcome Screen** → Login or Register
2. **Movie List Display** → View unsorted and sorted list
3. **Select Movie & Showtime**
4. **Select Show Type (2D, IMAX, Director's Club)**
5. **Seat Selection** → Real-time availability
6. **Booking Confirmation** → View price, discounts, total
7. **Choose Payment Method**
8. **Ticket Generation** → QR code issued
9. **Summary Screen** → Booking complete

---

11. Folder Structure

```
CinemaBookingSystem/
│
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── cinema/
│   │   │   │   ├── model/        # Entities (Movie, Seat, User, etc.)
│   │   │   │   ├── service/      # Logic (BookingService,
PaymentService)
│   │   │   │   ├── util/         # Helpers (JSONHandler, QRGenerator)
│   │   │   │   └── main/         # Entry point (Main.java)
│   │   └── resources/
│   │       └── data/             # JSON files (movies.json,
users.json)
│   └── test/                     # Unit tests
│
├── docs/
│   ├── System_Design_Guidelines.docx
│   ├── ClassDiagram.uml
│   └── Flowchart.uml
│
├── .gitignore
├── README.md
└── pom.xml / build.gradle
```

---

12. Implementation Notes

- JSON files serve as **local storage** instead of a database.
- All modules must follow **OOP best practices**.
- Include **DSA documentation** explaining the use of LinkedList and TreeMap.
- Computational logic (pricing, discounts, sorting) must be documented clearly.
- The system should be **interactive through the console or GUI (optional)**.

Summary

This Cinema Ticket Booking System demonstrates how OOP and DSA principles can be applied to a real-world problem.It simulates the full user journey — from browsing movies to ticket generation — with well-structured code, reusable classes, and clean logic.