Practical 7

Design Class & Object Diagram for the Software domain problem.

Class Diagram

A class diagram is a structural diagram in the Unified Modeling Language (UML) that visually represents the static structure of a system by defining its **classes, attributes, methods, and relationships** between different components. It is a key component in object-oriented design, helping to model real-world entities and their interactions within a system.

- Importance of Class Diagrams

- **Defining System Structure**: Provides a blueprint of system architecture by depicting various classes and their associations.
- **Encapsulating Data and Behavior**: Clearly distinguishes attributes (data) and methods (functions) for each class.
- **Facilitating Code Implementation**: Acts as a foundation for developers by outlining how different objects interact.
- **Enhancing Communication**: Serves as a shared reference for stakeholders, developers, and designers to understand system behavior.
- **Supporting Maintainability**: Helps in debugging, extending, or modifying the system by offering a clear representation of dependencies.

- Components of a Class Diagram

- **Classes**: Represent real-world entities with attributes and operations.
- **Attributes**: Define the characteristics or properties of a class.
- **Methods (Operations)**: Represent the behaviors or functionalities of the class.
- **Relationships**: Show how different classes interact (Association, Aggregation, Composition, Inheritance).
- **Visibility Indicators**: Denote access levels (public +, private -, protected #).

Class Diagram for Movie Booking Management System

This class diagram provides an **object-oriented structure** for a **Movie Booking Management System**, representing key entities, their attributes, behaviors, and relationships. Below is a detailed breakdown of the primary classes.

1. User

A **User** represents a person who interacts with the system, either to book movie tickets, leave reviews, or manage their profile.

- **Attributes:**
  - username: Stores the unique identifier for a user.
  - password: Used for authentication and security.
- **Relationships:**
  - A User is authenticated through the Auth class, ensuring secure login and registration.
  - A User can have multiple Bookings, allowing them to reserve multiple tickets.
  - A User can write multiple Reviews, contributing feedback for different movies.

2. Auth

The **Auth** class is responsible for user authentication and account creation.

- **Attributes:**
  - username: Identifies the user attempting to log in.
  - password: Ensures secure access to the account.
- **Methods:**
  - login(): Verifies the user's credentials and grants access.
  - register(): Creates a new user account in the system.
  - generateUserID(): Generates a unique identifier for each user.
- **Relationships:**
  - Directly linked to User, as authentication is mandatory for any user interaction.

3. Profile

The **Profile** class stores personal information about the user, allowing them to manage their details.

- **Attributes:**
  - name: The user's full name.
  - email: Contact email for communication.
  - contact_no: A phone number for notifications and support.
  - userID: A unique identifier linking the profile to a user account.
  - profile_pic: An image associated with the user's account.

- **Methods:**
  - viewProfile(): Displays the user's profile details.
  - deleteProfile(): Allows a user to remove their profile.
  - editProfile(): Enables users to update their information.
  - viewTickets(): Retrieves a list of past and upcoming bookings.
- **Relationships:**
  - Connected to User, as each user has a single profile.

4. Movie

Represents a movie available for booking within the system.

- **Attributes:**
  - name: The title of the movie.
  - cast: Stores images of the main cast.
  - description: A summary of the movie's plot.
  - ratings: The average rating based on user reviews.
  - director: Name of the director.
  - movieID: A unique identifier for each movie.
- **Methods:**
  - showRatings(): Returns the movie's overall rating.
  - showDetails(): Displays a complete description of the movie.
- **Relationships:**
  - A Movie has multiple Showtimes, representing different screening times.
  - A Movie is linked to multiple Reviews, allowing users to provide feedback.
  - A Movie is available in one or more Theatres, showing where it can be watched.
  - The Admin manages movie records, ensuring accurate listings and updates.

5. Booking

Handles the reservation process for movie tickets.

- **Attributes:**
  - ticketID: A unique identifier for each booking.
  - movie: The name of the selected movie.
  - screen: The screen number in the theatre.
  - seat: The allocated seat number.
  - theater: The name of the theatre where the booking is made.
  - date: The date of the show.
- **Methods:**
  - bookTicket(): Initiates the booking process.
  - generateTicket(): Creates a digital or printable ticket.

  - cancelTicket(): Allows the user to cancel their reservation.
- **Relationships:**
  - A User can have multiple Bookings.
  - A Booking is associated with a specific Showtime.
  - A Booking requires a Payment transaction to be completed.

6. Payment

Manages the financial transactions for ticket purchases.

- **Attributes:**
  - transactionID: A unique number for tracking payments.
  - userID: Links the transaction to a specific user.
  - ticketID: Associates the payment with a booked ticket.
- **Methods:**
  - makePayment(): Processes the payment.
  - generateTransactionID(): Creates a unique ID for each transaction.
  - cancelPayment(): Allows for refunds or payment reversals.
- **Relationships:**
  - A Booking requires a Payment to be confirmed.

7. Showtime

Represents the scheduled screening times for a movie.

- **Attributes:**
  - movieID: Identifies the movie being shown.
  - theatreID: Identifies the theatre hosting the show.
  - showTime: A list of available screening times.
- **Methods:**
  - showShowTimes(): Retrieves the available showtimes for a movie.
- **Relationships:**
  - A Movie can have multiple Showtimes.
  - A Showtime is associated with a Theatre where it is played.
  - A Showtime is linked to multiple Bookings.

8. Theatre

Represents a physical theatre where movies are screened.

- **Attributes:**
  - name: The theatre’s name.
  - address: Location of the theatre.
  - movieList: A list of movies currently available.
- **Methods:**
  - moviesAvailable(): Checks whether a particular movie is currently playing.

- **Relationships:**
  - A Theatre offers multiple Movies.
  - A Theatre hosts multiple Showtimes.

9. Reviews

Stores user-generated feedback for movies.

- **Attributes:**
  - ratings: A numerical score given by the user.
  - description: The user's review text.
  - author: Identifies the reviewer.
  - type: The category of review (e.g., critic, user).
- **Methods:**
  - addReview(): Allows users to submit a review.
  - calculateReview(): Computes the overall movie rating.
- **Relationships:**
  - A User can write multiple Reviews.
  - A Movie can have multiple Reviews.

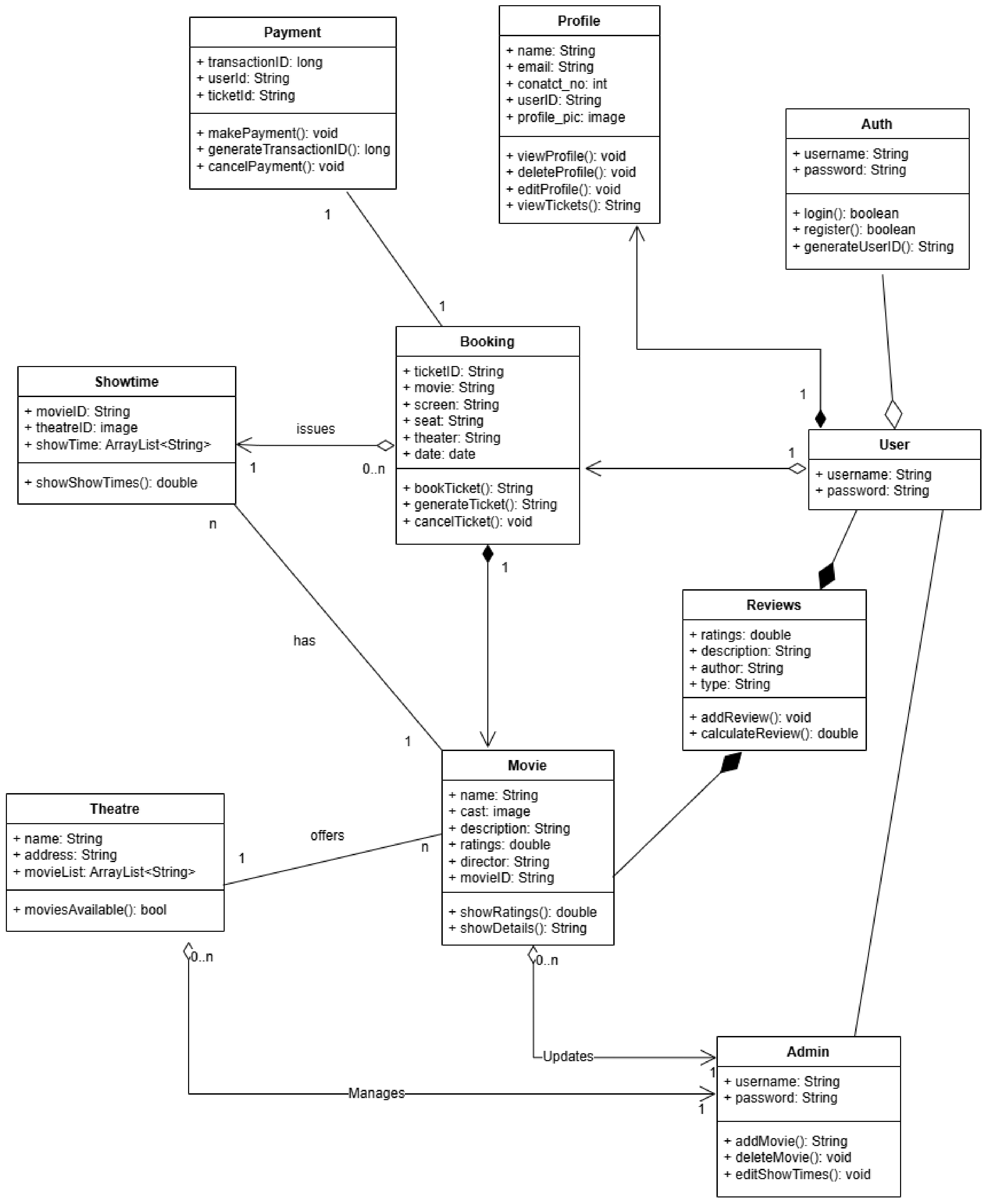
10. Admin

Responsible for managing movie listings, showtimes, and overall system maintenance.

- **Attributes:**
  - username: Admin login identifier.
  - password: Secure authentication for administrative access.
- **Methods:**
  - addMovie(): Adds new movies to the system.
  - deleteMovie(): Removes outdated or incorrect listings.
  - editShowTimes(): Modifies available screening times.
- **Relationships:**
  - The Admin directly updates the Movie and Showtime records.

System Flow Summary

1. A User logs in using the Auth system.
2. The Profile stores personal details, while the user browses Movies.
3. The user selects a Showtime and makes a Booking.
4. A Payment is processed for the ticket.
5. The user attends the screening at the Theatre.
6. After watching, they can leave a Review.
7. The Admin manages movie records and scheduling.



Class Diagram for Movie Booking Management System