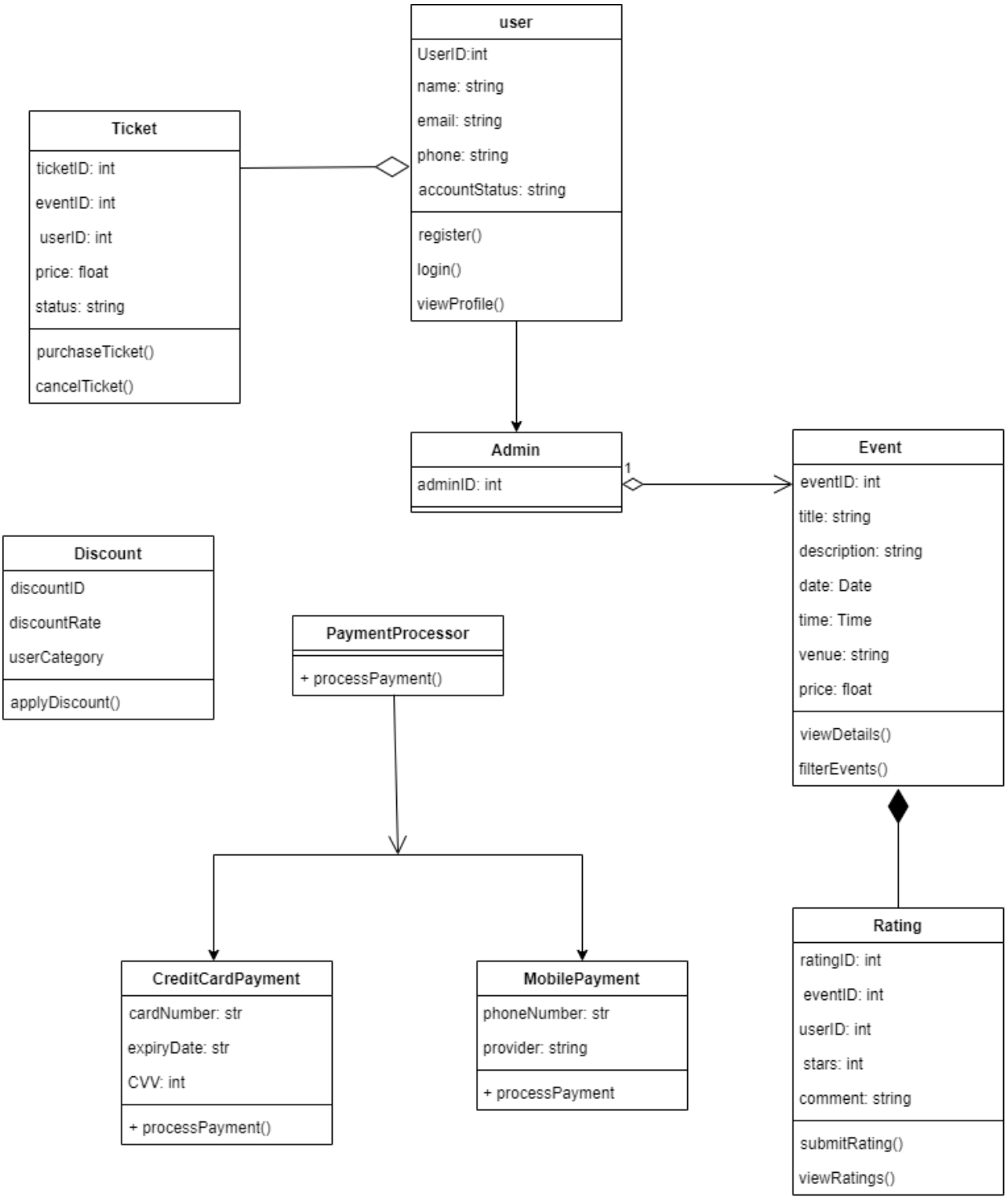
CS 438 - Assignment 2

System Design Specification for Booking Ticket Event System (BTES)

May Moktar AL-Gallai

1. UML Booking Ticket Event System (BTES):



2. Class Definitions (Clarification & Additional Details)

1. User Class:

- Methods: Already specified (register(), login(), viewProfile(), loadUsersFromCSV(), saveUserToCSV())
- Associations: User should have an association with Ticket since a user will purchase tickets. This will likely be a one-to-many association (one user can buy multiple tickets).
- Dependencies: The User class depends on the CSV file to load and save user data.

2. Admin Class:

- Methods: Already specified (addEvent(), editEvent(), viewSummary())
- Associations: Admin should have an association with the Event class (since admins manage events) and perhaps an association with Ticket for viewing ticket summaries.
- Dependencies: The Admin class depends on event and ticket data stored in CSV files.

3. Event Class:

- Methods: Already specified (viewEventDetails(), searchEvent(), filterEvents())
- Associations: An Event should be associated with Ticket because events are connected to ticket purchases. Also, Event may be associated with Rating if users can rate events.
- Dependencies: Depends on a CSV file for event storage.

4. Ticket Class:

- Methods: Already specified (purchaseTicket(), printTicket(), cancelTicket())
- Associations: Ticket should be associated with User and Event (each ticket belongs to a user and corresponds to an event).
- Dependencies: Ticket data depends on a CSV file.

5. Payment Class:

- Methods: Already specified (processPayment())
- Associations: Payment should be associated with Ticket, as each payment is tied to a ticket purchase.
- Dependencies: Payment information is stored in a CSV file.

6. Discount Class:

- Methods: Already specified (applyDiscount())
- Associations: Discount should be associated with User (since discounts apply to users) and potentially Event (if discounts are event-specific).
- Dependencies: Discount data is stored in a CSV file.

7. Rating Class:

- Methods: Already specified (submitRating(), viewRatings())
- Associations: Rating should be associated with both User (who submits the rating) and Event (that is being rated).
- Dependencies: Rating information is stored in a CSV file.

3. Glossary

Term	Description
User Class	This class manages user profiles, handles user registration and login, manages ticket purchases, and allows users to provide feedback on events. User data is read from and written to a CSV file.
Admin Class	This class is responsible for managing event data, including adding, editing, and removing events. It also generates summaries of event performance. All event data is stored and retrieved from a CSV file.
Ticket Class	This class manages ticket purchases, cancellations, and printing. It reads ticket data from and updates it to a CSV file, ensuring ticket availability and validation.
Event Class	This class stores information about events, including the event title, date, time, venue, and available tickets. It allows for event searches, filtering, and sorting, all based on CSV file data.
Payment Class	This class manages the payment process for purchasing tickets. Transaction details are recorded in a CSV file. Payment methods could include credit card or mobile payment.
Discount Class	This class applies discounts to ticket purchases based on eligibility criteria, such as user category (e.g., student, senior). Discount records are saved in the CSV file.
Rating Class	This class allows users to submit and view ratings and comments for events. Ratings and feedback are saved in a CSV file and can be viewed later by users or admins.
CSV File	A comma-separated values file used for storing structured data in a tabular format. It is used in this system as the primary data storage for users, events, tickets, payments, and ratings.
Data Import/Export	The process of reading data from and writing data to CSV files. This replaces traditional database queries, with operations performed through file handling.
File I/O Operations	Input/output operations involving reading from and writing to CSV files, used to manage system data such as users, events, and tickets.

4. Design Decision Rationale

Significant Design Decision:

A key design decision in the development of this system was leveraging inheritance to manage the relationship between the User and Admin classes. Additionally, composition was employed to organize relationships between the User, Ticket, and Event classes, allowing for better separation of concerns.

Rationale:

- The Admin class shares common features with the User class, such as user management and profile viewing, but has extended privileges like adding, editing, and deleting events. By extending the User class, we avoid code duplication, making the design both scalable and maintainable.
- Composition is used to model the relationships between classes such as User, Ticket, and Event. Each User can own multiple Tickets, and each Ticket is associated with an Event. This promotes clear and logical data organization.
- Using inheritance for Admin and composition for relationships follows the principle of code reuse and enhances the clarity of the overall design, ensuring future updates or new roles can be integrated with minimal effort.

Pros and Cons:

Pros:

- Encapsulation: Each class manages its own data and responsibilities, allowing for better modularity and adherence to object-oriented principles.
- Inheritance: The Admin class inherits the common functionality from User, which avoids code duplication and maintains consistency across user management tasks.
- Composition: The use of composition (for example, between User and Ticket) allows for a more flexible structure where relationships between classes can evolve without major restructuring.
- Polymorphism: With polymorphism, both Admin and User can interact with shared components (such as Ticket and Event), allowing for flexible and extendable system behavior based on user roles.

Cons:

- Inheritance: Although inheritance simplifies the relationship between User and Admin, adding more complex roles or more layers of hierarchy (e.g., different types of admins or users) in the future could introduce the need for refactoring, as the design might become rigid and harder to extend.
- Coupling: There is a potential for tight coupling between the User, Ticket, and Event classes, especially if many interactions are required between them. This could make future changes more complex, though encapsulation and clear interface definitions can mitigate this risk.

This design decision was made to ensure the system is modular, maintainable, and capable of handling future expansions with minimal disruption to the core architecture. By balancing inheritance for shared functionality and composition for relationships, we optimize for scalability while keeping the codebase clean and easy to understand.

5. Data Organization Description

Stored Data:

The system will store various types of data in structured CSV files, each corresponding to specific entities within the system. The following data will be stored:

1. **User Data**:
    - **Details**: This includes information about registered users, such as:
        - userID: A unique identifier for each user.
        - name: The name of the user.
        - email: The user's email address.
        - phone: The user's contact number.
        - accountStatus: The status of the account (e.g., active, suspended).
    - **Storage File**: users.csv
2. **Event Data**:
    - **Details**: Information about events, including:
        - eventID: A unique identifier for each event.
        - title: The name or title of the event.
        - description: A brief description of the event.
        - date: The date of the event.
        - venue: The location where the event will take place.
        - price: The price for attending the event.
    - **Storage File**: events.csv
3. **Ticket Data**:
    - **Details**: Data related to ticket purchases, including:
        - ticketID: A unique identifier for each ticket.
        - eventID: The identifier of the event associated with the ticket.
        - userID: The identifier of the user who purchased the ticket.
        - price: The price of the ticket.
    - **Storage File**: tickets.csv
4. **Payment Data**:
    - **Details**: Information on payments made for ticket purchases, including:
        - paymentID: A unique identifier for each payment transaction.
        - amount: The amount paid for the ticket(s).

        - method: The payment method used (e.g., credit card, mobile payment).
        - status: The status of the payment (e.g., successful, pending, failed).
    - **Storage File**: payments.csv
5. **Rating Data**:
    - **Details**: User-submitted ratings and comments for events, including:
        - ratingID: A unique identifier for each rating.
        - eventID: The identifier of the event being rated.
        - userID: The identifier of the user who submitted the rating.
        - ratingValue: The rating score (e.g., 1 to 5 stars).
        - comment: Any comments provided by the user regarding the event.
    - **Storage File**: ratings.csv

Data Storage:

The system will use **CSV files** to store data in a structured format. Each file will contain relevant data for its corresponding entity, as listed below:

- users.csv: Stores user data.
- events.csv: Stores event details.
- tickets.csv: Stores ticket information for purchased tickets.
- payments.csv: Stores payment transactions.
- ratings.csv: Stores user ratings and comments for events.

CSV Operations:

Each class in the system will have methods to **load**, **update**, and **save** data from/to these CSV files. These methods will ensure data persistence across the system.

- **User Class**:
    - loadUsersFromCSV(): Load user data from the users.csv file.
    - saveUserToCSV(): Save a new user's data to the users.csv file.
- **Event Class**:
    - loadEventsFromCSV(): Load event data from the events.csv file.
    - saveEventToCSV(): Save new or modified event data to the events.csv file.
- **Ticket Class**:
    - loadTicketsFromCSV(): Load ticket data from the tickets.csv file.
    - saveTicketToCSV(): Save purchased ticket information to the tickets.csv file.

- **Payment Class**:
    - loadPaymentsFromCSV(): Load payment data from the payments.csv file.
    - savePaymentToCSV(): Save payment transaction data to the payments.csv file.
- **Rating Class**:
    - loadRatingsFromCSV(): Load event ratings from the ratings.csv file.
    - saveRatingToCSV(): Save user-submitted ratings and comments to the ratings.csv file.

By organizing data in this way, the system ensures efficient data management and retrieval while maintaining clear separation between different types of data. This structure also simplifies updates, as data changes can be directly written to the corresponding CSV file.