Ticket Booking System

Task 1: Class & Object

Create a Following classes with the following attributes and methods:

1. **Event** Class**:**
   - **Attributes:**
     - event_name,
     - event_date DATE,
     - event_time TIME,
     - venue_name,
     - total_seats,
     - available_seats,
     - ticket_price DECIMAL,
     - event_type ENUM('Movie', 'Sports', 'Concert')
   - **Methods and Constuctors:**
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
     - **calculate_total_revenue()**: Calculate and return the total revenue based on the number of tickets sold.
     - **getBookedNoOfTickets()**: return the total booked tickets
     - **book_tickets(num_tickets)**: Book a specified number of tickets for an event. Initially available seats are equal to the total seats when tickets are booked available seats number should be reduced.
     - **cancel_booking(num_tickets)**: Cancel the booking and update the available seats.
     - **display_event_details():** Display event details, including event name, date time seat availability.
2. **Venue** Class
   - **Attributes**:
     - venue_name,
     - address
   - **Methods and Constuctors:**
     - **display_venue_details():** Display venue details.
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
3. **Customer Class**
   - **Attributes:**
     - customer_name,
     - email,
     - phone_number,
   - **Methods and Constuctors:**
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
     - **display_customer_details()**: Display customer details.

4. **Booking** Class to represent the Tiket booking system. Perform the following operation in main method. Note:- Use Event class object for the following operation.
    - **Methods and Constuctors:**
        - **calculate_booking_cost(num_tickets)**: Calculate and set the total cost of the booking.
        - **book_tickets(num_tickets)**: Book a specified number of tickets for an event.
        - **cancel_booking(num_tickets)**: Cancel the booking and update the available seats.
        - **getAvailableNoOfTickets()**: return the total available tickets
        - **getEventDetails()**: return event details from the event class

Task 2: Inheritance and polymorphism

1. **Inheritance**
    - Create a subclass **Movie** that inherits from **Event**. Add the following attributes and methods:
        - **Attributes:**
            1. genre: Genre of the movie (e.g., Action, Comedy, Horror).
            2. ActorName
            3. ActresName
        - **Methods:**
            1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
            2. **display_event_details():** Display movie details, including genre.
    - Create another subclass **Concert** that inherits from **Event**. Add the following attributes and methods:
        - **Attributes:**
            1. artist: Name of the performing artist or band.
            2. type: (Theatrical, Classical, Rock, Recital)
        - **Methods:**
            1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
            2. **display_concert_details():** Display concert details, including the artist.
    - Create another subclass **Sports** that inherits from **Event**. Add the following attributes and methods:
        - **Attributes:**
            1. sportName: Name of the game.
            2. teamsName: (India vs Pakistan)
        - **Methods:**
            1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
            2. **display_sport_details():** Display concert details, including the artist.
    - Create a class **TicketBookingSystem** with the following methods:
        - **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu_name:str):** Create a new event with the specified details and event type (movie, sport or concert) and return event object.
        - **display_event_details(event: Event)**: Accepts an event object and calls its **display_event_details()** method to display event details.

- **book_tickets(event: Event, num_tickets: int):**
    1. Accepts an event object and the number of tickets to be booked.
    2. Checks if there are enough available seats for the booking.
    3. If seats are available, updates the available seats and returns the total cost of the booking.
    4. If seats are not available, displays a message indicating that the event is sold out.
- **cancel_tickets(event: Event, num_tickets)**: cancel a specified number of tickets for an event.
- **main():** simulates the ticket booking system
    1. User can book tickets and view the event details as per their choice in menu (movies, sports, concerts).
    2. Display event details using the display_event_details() method without knowing the specific event type (demonstrate polymorphism).
    3. Make bookings using the book_tickets() and cancel tickets cancel_tickets() method.

Task 3: Abstraction

Requirements:

1. **Event Abstraction:**
    - Create an abstract class **Event** that represents a generic event. It should include the following attributes and methods as mentioned in *TASK 1*:
2. Concrete **Event** Classes**:**
    - Create three concrete classes that inherit from **Event** abstract class and override abstract methods in concrete class should declare the variables as mentioned in above *Task 2*:
        - Movie.
        - Concert.
        - Sport.
3. **BookingSystem** Abstraction:
    - Create an abstract class **BookingSystem** that represents the ticket booking system. It should include the methods of TASK 2 **TicketBookingSystem**:
4. Concrete **TicketBookingSystem** Class:

- Create a concrete class **TicketBookingSystem** that inherits from BookingSystem:
  - **TicketBookingSystem**: Implement the abstract methods to create events, book tickets, and retrieve available seats. Maintain an array of events in this class.
- Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats," and "exit."

Task 4: Has A Relation / Association

Create a Following classes with the following attributes and methods:

1. **Venue** Class
   - **Attributes**:
     - venue_name,
     - address
   - **Methods and Constuctors:**
     - **display_venue_details():** Display venue details.
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
2. **Event** Class**:**
   - **Attributes:**
     - event_name,
     - event_date DATE,
     - event_time TIME,
     - venue (reference of class **Venu**),
     - total_seats,
     - available_seats,
     - ticket_price DECIMAL,
     - event_type ENUM('Movie', 'Sports', 'Concert')
   - **Methods and Constuctors:**
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
     - **calculate_total_revenue()**: Calculate and return the total revenue based on the number of tickets sold.
     - **getBookedNoOfTickets()**: return the total booked tickets
     - **book_tickets(num_tickets)**: Book a specified number of tickets for an event. Initially available seats are equal to total seats when tickets are booked available seats number should be reduced.
     - **cancel_booking(num_tickets)**: Cancel the booking and update the available seats.
     - **display_event_details():** Display event details, including event name, date time seat availability.
3. **Event** sub classes**:**
   - Create three sub classes that inherit from **Event** abstract class and override abstract methods in concrete class should declare the variables as mentioned in above *Task 2*:
     - Movie.

    - Concert.
    - Sport.

4. **Customer** Class
    - **Attributes:**
        - customer_name,
        - email,
        - phone_number,
    - **Methods and Constuctors:**
        - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
        - **display_customer_details()**: Display customer details.
5. Create a class **Booking** with the following attributes:
    - bookingId (should be incremented for each booking)
    - array of customer (reference to the customer who made the booking)
    - event (reference to the event booked)
    - num_tickets(no of tickets and array of customer must equal)
    - total_cost
    - booking_date (timestamp of when the booking was made)
    - **Methods and Constuctors:**
        - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
        - **display_booking_details()**: Display customer details.
6. **BookingSystem** Class to represent the Ticket booking system. Perform the following operation in main method. Note: - Use Event class object for the following operation.
    - **Attributes**
        - array of events
    - **Methods and Constuctors:**
        - **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu:Venu):** Create a new event with the specified details and event type (movie, sport or concert) and return event object.
        - **calculate_booking_cost(num_tickets)**: Calculate and set the total cost of the booking.
        - **book_tickets(eventname:str, num_tickets, arrayOfCustomer)**: Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of **Booking** class.
        - **cancel_booking(booking_id)**: Cancel the booking and update the available seats.
        - **getAvailableNoOfTickets()**: return the total available tickets
        - **getEventDetails()**: return event details from the event class
        - Create a simple user interface in a **main method** that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats,", "get_event_details," and "exit."

Task 5: Interface/abstract class, and Single Inheritance, static variable

1. Create **Venue,** class as mentioned above Task 4.
2. **Event** Class**:**
    - **Attributes:**
        - event_name,
        - event_date DATE,
        - event_time TIME,
        - venue (reference of class Venu),
        - total_seats,
        - available_seats,
        - ticket_price DECIMAL,
        - event_type ENUM('Movie', 'Sports', 'Concert')
    - **Methods and Constuctors:**
        - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
3. Create **Event** sub classes as mentioned in above Task 4**.**
4. Create a class **Customer** and **Booking** as mentioned in above Task 4**.**
5. Create interface/abstract class **IEventServiceProvider** with following methods:
    - **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu: Venu):** Create a new event with the specified details and event type (movie, sport or concert) and return event object.
    - **getEventDetails():** return array of event details from the event class.
    - **getAvailableNoOfTickets()**: return the total available tickets.
6. Create interface/abstract class **IBookingSystemServiceProvider** with following methods:
    - **calculate_booking_cost(num_tickets):** Calculate and set the total cost of the booking.
    - **book_tickets(eventname:str, num_tickets, arrayOfCustomer):** Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class.
    - **cancel_booking(booking_id):** Cancel the booking and update the available seats.
    - **get_booking_details(booking_id):**get the booking details.
7. Create **EventServiceProviderImpl** class which implements **IEventServiceProvider** provide all implementation methods.
8. Create **BookingSystemServiceProviderImpl** class which implements **IBookingSystemServiceProvider** provide all implementation methods and inherits **EventServiceProviderImpl** class with following attributes.
    - **Attributes**
        - array of events
9. Create **TicketBookingSystem** class and perform following operations:
    - Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats,", "get_event_details," and "exit."
10. Place the interface/abstract class in service package and interface/abstract class implementation class, all concrete class in bean package and **TicketBookingSystem** class in app package.

11. Should display appropriate message when the event or booking id is not found or any other wrong information provided.

Task 6: Exception Handling

throw the exception whenever needed and Handle in main method,

1. **EventNotFoundException** throw this exception when user try to book the tickets for Event not listed in the menu.
2. **InvalidBookingIDException** throw this exception when user entered the invalid bookingId when he tries to view the booking or cancel the booking.
3. **NullPointerException** handle in main method**.**

Throw these exceptions from the methods in **TicketBookingSystem** class. Make necessary changes to accommodate exception in the source code. Handle all these exceptions from the main program.

Task 7: Collection

1. From the previous task change the **Booking** class attribute customers to List of customers and **BookingSystem** class attribute events to List of events and perform the same operation.
2. From the previous task change all list type of attribute to type Set in **Booking** and **BookingSystem** class and perform the same operation.
    - Avoid adding duplicate Account object to the set.
    - Create Comparator<Event> object to sort the event based on event name and location in alphabetical order.
3. From the previous task change all list type of attribute to type Map object in **Booking** and **BookingSystem** class and perform the same operation.