Ticket Booking System

Instructions

- Submitting assignments should be a single file or through git hub link shared with trainer and hexavarsity.
- Each assignment builds upon the previous one, and by the end, you will have a comprehensive Banking System implemented in Java/C#/Python with a strong focus on SQL schema design, control flow statements, loops, arrays, collections, and database interaction.
- Follow object-oriented principles throughout the Java programming assignments. Use classes and objects to model real-world entities, encapsulate data and behaviour, and ensure code reusability.
- Throw user defined exception from method and handle in the main method**.**
- The following **Directory structure** is to be followed in the application.
    - **entity/model**
        - Create entity classes in this package. All entity class should not have any business logic.
    - **dao**
        - Create Service Provider interface/abstract class to showcase functionalities.
        - Create the implementation class for the above interface with db interaction.
    - **exception**
        - Create user defined exceptions in this package and handle exceptions whenever needed.
    - **util**
        - Create a **DBPropertyUtil** class with a static function which takes property file name as parameter and returns connection string.
        - Create a **DBConnUtil** class which holds **static method** which takes connection string as parameter file and returns **connection object**.
    - **main**
        - Create a class MainModule and demonstrate the functionalities in a menu driven application.

You are tasked with creating a ticket booking system for a Event. The system should support booking tickets for different types of events, such as movies, concerts, and plays. Each event has its own pricing strategy, and the system should also track available seats and customer bookings.

Database Tables

1. **Venu Table**
    - **venue_id (Primary Key)**
    - venue_name,
    - address
2. **Event Table**
    - **event_id (Primary Key)**
    - event_name,
    - event_date DATE,
    - event_time TIME,

- venue_id **(Foreign Key)**,
- total_seats,
- available_seats,
- ticket_price DECIMAL,
- event_type ('Movie', 'Sports', 'Concert')
- booking_id **(Foreign Key)**

3. **Customer Table**
   - **customer_id (Primary key)**
   - customer_name,
   - email,
   - phone_number,
   - booking_id **(Foreign Key)**
4. **Booking Table**
   - booking_id **(Primary Key)**,
   - customer_id **(Foreign Key)**,
   - event_id **(Foreign Key)**,
   - num_tickets,
   - total_cost,
   - booking_date,

Tasks 1: Database Design:

1. Create the database named "TicketBookingSystem"
2. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.
   - Venu
   - Event
   - Customers
   - Booking
3. Create an ERD (Entity Relationship Diagram) for the database.
4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

Tasks 2: Select, Where, Between, AND, LIKE:

1. Write a SQL query to insert at least 10 sample records into each table.
2. Write a SQL query to list all Events.
3. Write a SQL query to select events with available tickets.
4. Write a SQL query to select events name partial match with ‘cup’.
5. Write a SQL query to select events with ticket price range is between 1000 to 2500.
6. Write a SQL query to retrieve events with dates falling within a specific range.
7. Write a SQL query to retrieve events with available tickets that also have "Concert" in their name.
8. Write a SQL query to retrieve users in batches of 5, starting from the 6th user.
9. Write a SQL query to retrieve bookings details contains booked no of ticket more than 4.

10. Write a SQL query to retrieve customer information whose phone number end with '000'
11. Write a SQL query to retrieve the events in order whose seat capacity more than 15000.
12. Write a SQL query to select events name not start with 'x', 'y', 'z'

Tasks 3: Aggregate functions, Having, Order By, GroupBy and Joins:

1. Write a SQL query to List Events and Their Average Ticket Prices.
2. Write a SQL query to Calculate the Total Revenue Generated by Events.
3. Write a SQL query to find the event with the highest ticket sales.
4. Write a SQL query to Calculate the Total Number of Tickets Sold for Each Event.
5. Write a SQL query to Find Events with No Ticket Sales.
6. Write a SQL query to Find the User Who Has Booked the Most Tickets.
7. Write a SQL query to List Events and the total number of tickets sold for each month.
8. Write a SQL query to calculate the average Ticket Price for Events in Each Venue.
9. Write a SQL query to calculate the total Number of Tickets Sold for Each Event Type.
10. Write a SQL query to calculate the total Revenue Generated by Events in Each Year.
11. Write a SQL query to list users who have booked tickets for multiple events.
12. Write a SQL query to calculate the Total Revenue Generated by Events for Each User.
13. Write a SQL query to calculate the Average Ticket Price for Events in Each Category and Venue.
14. Write a SQL query to list Users and the Total Number of Tickets They've Purchased in the Last 30 Days.

Tasks 4: Subquery and its types

1. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery.
2. Find Events with More Than 50% of Tickets Sold using subquery.
3. Calculate the Total Number of Tickets Sold for Each Event.
4. Find Users Who Have Not Booked Any Tickets Using a NOT EXISTS Subquery.
5. List Events with No Ticket Sales Using a NOT IN Subquery.
6. Calculate the Total Number of Tickets Sold for Each Event Type Using a Subquery in the FROM Clause.
7. Find Events with Ticket Prices Higher Than the Average Ticket Price Using a Subquery in the WHERE Clause.
8. Calculate the Total Revenue Generated by Events for Each User Using a Correlated Subquery.
9. List Users Who Have Booked Tickets for Events in a Given Venue Using a Subquery in the WHERE Clause.
10. Calculate the Total Number of Tickets Sold for Each Event Category Using a Subquery with GROUP BY.
11. Find Users Who Have Booked Tickets for Events in each Month Using a Subquery with DATE_FORMAT.
12. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery

Control structure

Task 1: Conditional Statements

In a BookingSystem, you have been given the task is to create a program to book tickets. if available tickets more than noOfTicket to book then display the remaining tickets or ticket unavailable:

Tasks:

1. Write a program that takes the availableTicket and noOfBookingTicket as input.
2. Use conditional statements (if-else) to determine if the ticket is available or not.
3. Display an appropriate message based on ticket availability.

Task 2: Nested Conditional Statements

Create a program that simulates a Ticket booking and calculating cost of tickets. Display tickets options such as "Silver", "Gold", "Dimond". Based on ticket category fix the base ticket price and get the user input for ticket type and no of tickets need and calculate the total cost of tickets booked.

Task 3: Looping

From the above task book the tickets for repeatedly until user type "Exit"

Task 4: 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 5: 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 6: 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 7: 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 8: 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 9: 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 10: 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.

Task 11: Database Connectivity.

1. Create **Venue, Event, Customer** and **Booking** class as mentioned above Task 5.
2. Create **Event** sub classes as mentioned in above Task 4**.**
3. Create interface/abstract class **IEventServiceProvider, IBookingSystemServiceProvider** and its implementation classes as mentioned in above Task 5**.**
4. Create I**BookingSystemRepository** interface/abstract class which include following methods to interact with database.
    - **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 and should store in database.
    - **getEventDetails():** return array of event details from the database.
    - **getAvailableNoOfTickets()**: return the total available tickets from the database.
    - **calculate_booking_cost(num_tickets):** Calculate and set the total cost of the booking.
    - **book_tickets(eventname:str, num_tickets, listOfCustomer):** 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 and stored in database.
    - **cancel_booking(booking_id):** Cancel the booking and update the available seats and stored in database.
    - **get_booking_details(booking_id):** get the booking details from database.

5. Create **BookingSystemRepositoryImpl** interface/abstract class which implements I**BookingSystemRepository** interface/abstract class and provide implementation of all methods and perform the database operations.
6. Create **DBUtil** class and add the following method.
    - **static getDBConn():Connection** Establish a connection to the database and return Connection reference
7. Place the interface/abstract class in service package and interface implementation class, concrete class in bean package and **TicketBookingSystemRepository** class in app package.
8. Should throw appropriate exception as mentioned in above task along with handle **SQLException**.
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."