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.

```python
def book_tickets(available_tickets, no_of_booking_tickets):
    if available_tickets >= no_of_booking_tickets:
        remaining_tickets = available_tickets - no_of_booking_tickets
        print("Tickets booked successfully! Remaining tickets:", remaining_tickets)
    else:
        print("Tickets unavailable!")

# Input
available_tickets = int(input("Enter the number of available tickets: "))
no_of_booking_tickets = int(input("Enter the number of tickets to book: "))

# Calling the function
book_tickets(available_tickets, no_of_booking_tickets)
```

```
Enter the number of available tickets: 10
Enter the number of tickets to book: 13
Tickets unavailable!

Process finished with exit code 0
```

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.

```
1 usage
def calculate_ticket_cost(ticket_type, no_of_tickets):
    base_prices = {"Silver": 50, "Gold": 100, "Diamond": 150}

    if ticket_type in base_prices:
        base_price = base_prices[ticket_type]
        total_cost = base_price * no_of_tickets
        print("Total cost for", no_of_tickets, ticket_type, "tickets:", total_cost)
    else:
        print("Invalid ticket type!")

# Input
ticket_type = input("Enter the ticket type (Silver/Gold/Diamond): ")
no_of_tickets = int(input("Enter the number of tickets needed: "))

# Calling the function
calculate_ticket_cost(ticket_type, no_of_tickets)
```

```
C:\Users\HP\PycharmProjects\ticketbooking\.venv\Scripts\py
Enter the ticket type (Silver/Gold/Diamond): Silver
Enter the number of tickets needed: 12
Total cost for 12 Silver tickets: 600

Process finished with exit code 0
```

Task 3: Looping From the above task book the tickets for repeatedly until user type "Exit"

```
1 usage
def calculate_ticket_cost(ticket_type, no_of_tickets):
    base_prices = {"Silver": 50, "Gold": 100, "Diamond": 150}

    if ticket_type in base_prices:
        base_price = base_prices[ticket_type]
        total_cost = base_price * no_of_tickets
        print("Total cost for", no_of_tickets, ticket_type, "tickets:", total_cost)
    else:
        print("Invalid ticket type!")
while True:
    ticket_type = input("Enter the ticket type (Silver/Gold/Diamond) or type 'Exit' to quit: ")

    if ticket_type.lower() == "exit":
        break

    no_of_tickets = int(input("Enter the number of tickets needed: "))

    calculate_ticket_cost(ticket_type, no_of_tickets)
```

```
C:\Users\HP\PycharmProjects\ticketbooking\.venv\Scripts\python.exe "C:\Users\HP
Enter the ticket type (Silver/Gold/Diamond) or type 'Exit' to quit: Silver
Enter the number of tickets needed: 12
Total cost for 12 Silver tickets: 600
Enter the ticket type (Silver/Gold/Diamond) or type 'Exit' to quit: Gold
Enter the number of tickets needed: 12
Total cost for 12 Gold tickets: 1200
Enter the ticket type (Silver/Gold/Diamond) or type 'Exit' to quit: Exit

Process finished with exit code 0
```

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. 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

```
from datetime import datetime

class Event:
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, event_type):
        self.event_name = event_name
        self.event_date = event_date
        self.event_time = event_time
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    def calculate_total_revenue(self):
        return (self.total_seats - self.available_seats) *
self.ticket_price

    def get_booked_no_of_tickets(self):
        return self.total_seats - self.available_seats

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, num_tickets):
        self.available_seats += num_tickets
        print(f"{num_tickets} tickets cancelled for {self.event_name}.")

    def display_event_details(self):
        print("Event Name:", self.event_name)
        print("Event Date:", self.event_date)
        print("Event Time:", self.event_time)
```

```
        print("Venue Name:", self.venue_name)
        print("Total Seats:", self.total_seats)
        print("Available Seats:", self.available_seats)
        print("Ticket Price:", self.ticket_price)
        print("Event Type:", self.event_type)
class Venue:
    def __init__(self, venue_name, address):
        self.venue_name = venue_name
        self.address = address

    def display_venue_details(self):
        print("Venue Name:", self.venue_name)
        print("Address:", self.address)

class Customer:
    def __init__(self, customer_name, email, phone_number):
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number

    def display_customer_details(self):
        print("Customer Name:", self.customer_name)
        print("Email:", self.email)
        print("Phone Number:", self.phone_number)

class Booking:
    def __init__(self, event):
        self.event = event

    def calculate_booking_cost(self, num_tickets):
        return num_tickets * self.event.ticket_price

    def book_tickets(self, num_tickets):
        self.event.book_tickets(num_tickets)

    def cancel_booking(self, num_tickets):
        self.event.cancel_booking(num_tickets)

    def get_available_no_of_tickets(self):
        return self.event.available_seats

    def get_event_details(self):
        return self.event.display_event_details()

class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, genre, actor_name, actress_name):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "Movie")
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def display_event_details(self):
        super().display_event_details()
        print("Genre:", self.genre)
        print("Actor Name:", self.actor_name)
        print("Actress Name:", self.actress_name)

class Concert(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
```

```
total_seats, ticket_price, artist, concert_type):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "Concert")
        self.artist = artist
        self.concert_type = concert_type

    def display_event_details(self):
        super().display_event_details()
        print("Artist:", self.artist)
        print("Concert Type:", self.concert_type)

class Sports(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, sport_name, teams_name):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "Sports")
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        super().display_event_details()
        print("Sport Name:", self.sport_name)
        print("Teams Name:", self.teams_name)

class TicketBookingSystem:
    def __init__(self):
        self.events = []

    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue_name):
        event_date = datetime.strptime(date, "%Y-%m-%d")
        event_time = datetime.strptime(time, "%H:%M").time()

        if event_type.lower() == "movie":
            event = Movie(event_name, event_date, event_time, venue_name,
total_seats, ticket_price,"","","")
        elif event_type.lower() == "concert":
            event = Concert(event_name, event_date, event_time, venue_name,
total_seats, ticket_price,"","")
        elif event_type.lower() == "sports":
            event = Sports(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "", "")
        else:
            print("Invalid event type!")
            return None

        self.events.append(event)
        return event

    def display_event_details(self, event):
        event.display_event_details()

    def book_tickets(self, event, num_tickets):
        event.book_tickets(num_tickets)

    def cancel_tickets(self, event, num_tickets):
        event.cancel_booking(num_tickets)
```

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."

```
from abc import ABC, abstractmethod
from datetime import datetime

class Event(ABC):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, event_type):
        self.event_name = event_name
        self.event_date = event_date
        self.event_time = event_time
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    @abstractmethod
    def display_event_details(self):
        pass

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, num_tickets):
        pass
```

```
class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, genre, actor_name, actress_name):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "Movie")
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def display_event_details(self):
        print("Event Type: Movie")
        print("Event Name:", self.event_name)
        print("Event Date:", self.event_date)
        print("Event Time:", self.event_time)
        print("Venue Name:", self.venue_name)
        print("Total Seats:", self.total_seats)
        print("Available Seats:", self.available_seats)
        print("Ticket Price:", self.ticket_price)
        print("Genre:", self.genre)
        print("Actor Name:", self.actor_name)
        print("Actress Name:", self.actress_name)

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, num_tickets):
        self.available_seats += num_tickets
        print(f"{num_tickets} tickets cancelled for {self.event_name}.")

class Concert(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, artist, concert_type):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "Concert")
        self.artist = artist
        self.concert_type = concert_type

    def display_event_details(self):
        print("Event Type: Concert")
        print("Event Name:", self.event_name)
        print("Event Date:", self.event_date)
        print("Event Time:", self.event_time)
        print("Venue Name:", self.venue_name)
        print("Total Seats:", self.total_seats)
        print("Available Seats:", self.available_seats)
        print("Ticket Price:", self.ticket_price)
        print("Artist:", self.artist)
        print("Concert Type:", self.concert_type)

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, num_tickets):
```

```
        self.available_seats += num_tickets
        print(f"{num_tickets} tickets cancelled for {self.event_name}.")

class Sport(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, sport_name, teams_name):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "Sports")
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        print("Event Type: Sports")
        print("Event Name:", self.event_name)
        print("Event Date:", self.event_date)
        print("Event Time:", self.event_time)
        print("Venue Name:", self.venue_name)
        print("Total Seats:", self.total_seats)
        print("Available Seats:", self.available_seats)
        print("Ticket Price:", self.ticket_price)
        print("Sport Name:", self.sport_name)
        print("Teams Name:", self.teams_name)

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, num_tickets):
        self.available_seats += num_tickets
        print(f"{num_tickets} tickets cancelled for {self.event_name}.")

class BookingSystem(ABC):
    @abstractmethod
    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue_name):
        pass

    @abstractmethod
    def display_event_details(self, event):
        pass

    @abstractmethod
    def book_tickets(self, event, num_tickets):
        pass

    @abstractmethod
    def cancel_tickets(self, event, num_tickets):
        pass

    @abstractmethod
    def main(self):
        pass

class TicketBookingSystem(BookingSystem):
    def __init__(self):
        self.events = []

    def create_event(self, event_name, date, time, total_seats,
```

```
ticket_price, event_type, venue_name):
        event_date = datetime.strptime(date, "%Y-%m-%d")
        event_time = datetime.strptime(time, "%H:%M").time()

        if event_type.lower() == "movie":
            genre = input("Enter movie genre: ")
            actor_name = input("Enter actor's name: ")
            actress_name = input("Enter actress's name: ")
            event = Movie(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, genre, actor_name, actress_name)
        elif event_type.lower() == "concert":
            artist = input("Enter artist's name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, artist, concert_type)
        elif event_type.lower() == "sports":
            sport_name = input("Enter sport name: ")
            teams_name = input("Enter teams playing: ")
            event = Sport(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, sport_name, teams_name)


        else:
            print("Invalid event type!")
            return None

        self.events.append(event)
        return event

    def display_event_details(self, event):
        event.display_event_details()

    def book_tickets(self, event, num_tickets):
        event.book_tickets(num_tickets)

    def cancel_tickets(self, event, num_tickets):
        event.cancel_booking(num_tickets)
```

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**

- **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."

```
from abc import ABC, abstractmethod
from datetime import datetime

class Venue:
    def __init__(self, venue_name, address):
        self.venue_name = venue_name
        self.address = address

    def display_venue_details(self):
```

```
        print("Venue Name:", self.venue_name)
        print("Address:", self.address)


class Customer:
    def __init__(self, customer_name, email, phone_number):
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number

    def display_customer_details(self):
        print("Customer Name:", self.customer_name)
        print("Email:", self.email)
        print("Phone Number:", self.phone_number)


class Booking:
    booking_id_counter = 1

    def __init__(self, event, customer, num_tickets, total_cost):
        self.booking_id = Booking.booking_id_counter
        Booking.booking_id_counter += 1
        self.event = event
        self.customer = customer
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = datetime.now()

    def display_booking_details(self):
        print("Booking ID:", self.booking_id)
        print("Event Name:", self.event.event_name)
        print("Event Date:", self.event.event_date)
        print("Number of Tickets:", self.num_tickets)
        print("Total Cost:", self.total_cost)
        print("Booking Date:", self.booking_date)


class Event(ABC):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, event_type):
        self.event_name = event_name
        self.event_date = event_date
        self.event_time = event_time
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    @abstractmethod
    def display_event_details(self):
        pass

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, num_tickets):
        pass
```

```
class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, genre, actor_name, actress_name):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "Movie")
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def display_event_details(self):
        print("Event Type: Movie")
        print("Event Name:", self.event_name)
        print("Event Date:", self.event_date)
        print("Event Time:", self.event_time)
        print("Venue Name:", self.venue_name)
        print("Total Seats:", self.total_seats)
        print("Available Seats:", self.available_seats)
        print("Ticket Price:", self.ticket_price)
        print("Genre:", self.genre)
        print("Actor Name:", self.actor_name)
        print("Actress Name:", self.actress_name)

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, num_tickets):
        self.available_seats += num_tickets
        print(f"{num_tickets} tickets cancelled for {self.event_name}.")

class Concert(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, artist, concert_type):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "Concert")
        self.artist = artist
        self.concert_type = concert_type

    def display_event_details(self):
        print("Event Type: Concert")
        print("Event Name:", self.event_name)
        print("Event Date:", self.event_date)
        print("Event Time:", self.event_time)
        print("Venue Name:", self.venue_name)
        print("Total Seats:", self.total_seats)
        print("Available Seats:", self.available_seats)
        print("Ticket Price:", self.ticket_price)
        print("Artist:", self.artist)
        print("Concert Type:", self.concert_type)

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, num_tickets):
```

```
        self.available_seats += num_tickets
        print(f"{num_tickets} tickets cancelled for {self.event_name}.")

class Sport(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, sport_name, teams_name):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, "Sports")
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        print("Event Type: Sports")
        print("Event Name:", self.event_name)
        print("Event Date:", self.event_date)
        print("Event Time:", self.event_time)
        print("Venue Name:", self.venue_name)
        print("Total Seats:", self.total_seats)
        print("Available Seats:", self.available_seats)
        print("Ticket Price:", self.ticket_price)
        print("Sport Name:", self.sport_name)
        print("Teams Name:", self.teams_name)

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, num_tickets):
        self.available_seats += num_tickets
        print(f"{num_tickets} tickets cancelled for {self.event_name}.")

class BookingSystem(ABC):
    @abstractmethod
    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue_name):
        pass

    @abstractmethod
    def display_event_details(self, event):
        pass

    @abstractmethod
    def book_tickets(self, event, num_tickets):
        pass

    @abstractmethod
    def cancel_tickets(self, event, num_tickets):
        pass

    @abstractmethod
    def main(self):
        pass

class TicketBookingSystem(BookingSystem):
    def __init__(self):
        super().__init__()
        self.bookings = []
```

```
    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue_name):
        event_date = datetime.strptime(date, "%Y-%m-%d")
        event_time = datetime.strptime(time, "%H:%M").time()

        if event_type.lower() == "movie":
            genre = input("Enter movie genre: ")
            actor_name = input("Enter actor's name: ")
            actress_name = input("Enter actress's name: ")
            event = Movie(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, genre, actor_name, actress_name)
        elif event_type.lower() == "concert":
            artist = input("Enter artist's name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, artist, concert_type)
        elif event_type.lower() == "sports":
            sport_name = input("Enter sport name: ")
            teams_name = input("Enter teams playing: ")
            event = Sport(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, sport_name, teams_name)
        else:
            print("Invalid event type!")
            return None

        self.events.append(event)
        return event

    def display_event_details(self, event):
        event.display_event_details()

    def book_tickets(self, event_name, num_tickets, customers):
        event = next((e for e in self.events if e.event_name ==
event_name), None)
        if event:
            if event.available_seats >= num_tickets:
                total_cost = num_tickets * event.ticket_price
                booking = Booking(event, customers, num_tickets,
total_cost)
                self.bookings.append(booking)
                event.available_seats -= num_tickets
                print(f"{num_tickets} tickets booked for {event_name}.")
            else:
                print("Tickets unavailable!")
        else:
            print("Event not found!")

    def cancel_tickets(self, event_name, booking_id):
        booking = next((b for b in self.bookings if b.booking_id ==
booking_id), None)
        if booking:
            event = booking.event
            event.available_seats += booking.num_tickets
            self.bookings.remove(booking)
            print(f"Booking {booking_id} cancelled.")
        else:
            print("Booking not found!")
```

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.
    - **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.

```python
from abc import ABC, abstractmethod
from datetime import datetime

# Venue class
class Venue:
    def __init__(self, venue_name, address):
        self.venue_name = venue_name
        self.address = address

    def display_venue_details(self):
        print("Venue Name:", self.venue_name)
        print("Address:", self.address)

# Event class
class Event(ABC):
    def __init__(self, event_name, event_date, event_time, venue,
total_seats, ticket_price, event_type):
        self.event_name = event_name
        self.event_date = event_date
        self.event_time = event_time
        self.venue = venue
        self.total_seats = total_seats
        self.available_seats = total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    def display_event_details(self):
        print("Event Name:", self.event_name)
        print("Event Date:", self.event_date)
        print("Event Time:", self.event_time)
        print("Venue:", self.venue)
        print("Total Seats:", self.total_seats)
        print("Available Seats:", self.available_seats)
        print("Ticket Price:", self.ticket_price)
        print("Event Type:", self.event_type)

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

# Movie class
class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue,
total_seats, ticket_price, genre, actor_name, actress_name):
        super().__init__(event_name, event_date, event_time, venue,
total_seats, ticket_price, "Movie")
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")
```

```
    def cancel_booking(self, booking_id):
        print("Cancellation not supported for movie tickets.")

# Concert class
class Concert(Event):
    def __init__(self, event_name, event_date, event_time, venue,
total_seats, ticket_price, artist, concert_type):
        super().__init__(event_name, event_date, event_time, venue,
total_seats, ticket_price, "Concert")
        self.artist = artist
        self.concert_type = concert_type

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, booking_id):
        print("Cancellation not supported for concert tickets.")

# Sport class
class Sport(Event):
    def __init__(self, event_name, event_date, event_time, venue,
total_seats, ticket_price, sport_name, teams_name):
        super().__init__(event_name, event_date, event_time, venue,
total_seats, ticket_price, "Sports")
        self.sport_name = sport_name
        self.teams_name = teams_name

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, booking_id):
        print("Cancellation not supported for sports tickets.")

# Customer class
class Customer:
    def __init__(self, customer_name, email, phone_number):
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number

    def display_customer_details(self):
        print("Customer Name:", self.customer_name)
        print("Email:", self.email)
        print("Phone Number:", self.phone_number)

# Booking class
class Booking:
    def __init__(self, booking_id, event, customer, num_tickets):
        self.booking_id = booking_id
        self.event = event
        self.customer = customer
        self.num_tickets = num_tickets
```

```
    def display_booking_details(self):
        print("Booking ID:", self.booking_id)
        print("Event:", self.event.event_name)
        print("Customer:", self.customer.customer_name)
        print("Number of Tickets:", self.num_tickets)

# Interface for Event Service Provider
class IEventServiceProvider(ABC):
    @abstractmethod
    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue):
        pass

    @abstractmethod
    def get_event_details(self):
        pass

    @abstractmethod
    def get_available_no_of_tickets(self):
        pass

# Interface for Booking System Service Provider
class IBookingSystemServiceProvider(ABC):
    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        pass

    @abstractmethod
    def book_tickets(self, event_name, num_tickets, customer_array):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

    @abstractmethod
    def get_booking_details(self, booking_id):
        pass

# Implementation of Event Service Provider
class EventServiceProviderImpl(IEventServiceProvider):
    events = []

    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue):
        event_date = datetime.strptime(date, "%Y-%m-%d")
        event_time = datetime.strptime(time, "%H:%M").time()
        event = None

        if event_type.lower() == "movie":
            genre = input("Enter movie genre: ")
            actor_name = input("Enter actor's name: ")
            actress_name = input("Enter actress's name: ")
            event = Movie(event_name, event_date, event_time, venue,
total_seats, ticket_price, genre, actor_name, actress_name)
        elif event_type.lower() == "concert":
            artist = input("Enter artist's name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue,
total_seats, ticket_price, artist, concert_type)
        elif event_type.lower() == "sports":
```

```
            sport_name = input("Enter sport name: ")
            teams_name = input("Enter teams playing: ")
            event = Sport(event_name, event_date, event_time, venue,
total_seats, ticket_price, sport_name, teams_name)
        else:
            print("Invalid event type!")

        if event:
            self.events.append(event)
            return event

    def get_event_details(self):
        return self.events

    def get_available_no_of_tickets(self):
        available_tickets = sum([event.available_seats for event in
self.events])
        return available_tickets

# Implementation of Booking System Service Provider
class BookingSystemServiceProviderImpl(IBookingSystemServiceProvider,
EventServiceProviderImpl):
    bookings = []
    booking_id_counter = 1

    def calculate_booking_cost(self, num_tickets):
        return num_tickets * self.events[0].ticket_price  # Assuming only
one event for simplicity

    def book_tickets(self, event_name, num_tickets, customer_array):
        event = next((e for e in self.events if e.event_name ==
event_name), None)
        if event:
            if event.available_seats >= num_tickets:
                total_cost = num_tickets * event.ticket_price
                booking = Booking(self.booking_id_counter, event,
customer_array, num_tickets)
                self.bookings.append(booking)
                event.available_seats -= num_tickets
                self.booking_id_counter += 1
                print(f"{num_tickets} tickets booked for {event_name}.")
            else:
                print("Tickets unavailable!")
        else:
            print("Event not found!")

    def cancel_booking(self, booking_id):
        booking = next((b for b in self.bookings if b.booking_id ==
booking_id), None)
        if booking:
            event = booking.event
            event.available_seats += booking.num_tickets
            self.bookings.remove(booking)
            print(f"Booking {booking_id} cancelled.")
        else:
            print("Booking not found!")

    def get_booking_details(self, booking_id):
        booking = next((b for b in self.bookings if b.booking_id ==
booking_id), None)
        if booking:
```

```
            booking.display_booking_details()
        else:
            print("Booking not found!")

# Ticket Booking System
class TicketBookingSystem:
    def __init__(self):
        self.booking_system_service_provider =
BookingSystemServiceProviderImpl()
```

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.

```
from abc import ABC, abstractmethod
from datetime import datetime

# Custom exceptions
class EventNotFoundException(Exception):
    pass

class InvalidBookingIDException(Exception):
    pass
```

```
            else:
                print("Invalid choice. Please try again.")
        except EventNotFoundException as e:
            print(f"Event not found: {e}")
        except InvalidBookingIDException as e:
            print(f"Invalid booking ID: {e}")
        except Exception as e:
            print(f"An error occurred: {e}")
```

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.

```
class BookingSystemServiceProviderImpl(IBookingSystemServiceProvider,
EventServiceProviderImpl):
```

```
    bookings: Set[Booking] = set()
    booking_id_counter = 1

    def calculate_booking_cost(self, num_tickets):
        pass

    def book_tickets(self, event_name, num_tickets, customer_array):
        event = next((e for e in self.events if e.event_name ==
event_name), None)
        if event:
            if event.available_seats >= num_tickets:
                total_cost = num_tickets * event.ticket_price
                booking = Booking(self.booking_id_counter, event,
customer_array, num_tickets)
                self.bookings.add(booking)
                event.available_seats -= num_tickets
                self.booking_id_counter += 1
                print(f"{num_tickets} tickets booked for {event_name}.")
            else:
                print("Tickets unavailable!")
        else:
            raise EventNotFoundException("Event not found!")

    def cancel_booking(self, booking_id):
        booking = next((b for b in self.bookings if b.booking_id ==
booking_id), None)
        if booking:
            event = booking.event
            event.available_seats += booking.num_tickets
            self.bookings.remove(booking)
            print(f"Booking {booking_id} cancelled.")
        else:
            raise InvalidBookingIDException("Booking not found!")

    def get_booking_details(self, booking_id):
        booking = next((b for b in self.bookings if b.booking_id ==
booking_id), None)
        if booking:
            booking.display_booking_details()
        else:
            raise InvalidBookingIDException("Booking not found!")
```

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."

```
from abc import ABC, abstractmethod
from datetime import datetime
from typing import List, Set, Optional
import mysql.connector

# Custom exceptions
class EventNotFoundException(Exception):
    pass

class InvalidBookingIDException(Exception):
    pass

# Venue class
class Venue:
```

```
    def __init__(self, venue_name, address):
        self.venue_name = venue_name
        self.address = address

    def display_venue_details(self):
        print("Venue Name:", self.venue_name)
        print("Address:", self.address)

# Event class
class Event(ABC):
    def __init__(self, event_name, event_date, event_time, venue,
total_seats, ticket_price, event_type):
        self.event_name = event_name
        self.event_date = event_date
        self.event_time = event_time
        self.venue = venue
        self.total_seats = total_seats
        self.available_seats = total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    def display_event_details(self):
        print("Event Name:", self.event_name)
        print("Event Date:", self.event_date)
        print("Event Time:", self.event_time)
        print("Venue:", self.venue)
        print("Total Seats:", self.total_seats)
        print("Available Seats:", self.available_seats)
        print("Ticket Price:", self.ticket_price)
        print("Event Type:", self.event_type)

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

# Movie class
class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue,
total_seats, ticket_price, genre, actor_name, actress_name):
        super().__init__(event_name, event_date, event_time, venue,
total_seats, ticket_price, "Movie")
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, booking_id):
        raise InvalidBookingIDException("Cancellation not supported for
movie tickets.")

# Concert class
```

```
class Concert(Event):
    def __init__(self, event_name, event_date, event_time, venue,
total_seats, ticket_price, artist, concert_type):
        super().__init__(event_name, event_date, event_time, venue,
total_seats, ticket_price, "Concert")
        self.artist = artist
        self.concert_type = concert_type

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, booking_id):
        raise InvalidBookingIDException("Cancellation not supported for
concert tickets.")

# Sport class
class Sport(Event):
    def __init__(self, event_name, event_date, event_time, venue,
total_seats, ticket_price, sport_name, teams_name):
        super().__init__(event_name, event_date, event_time, venue,
total_seats, ticket_price, "Sports")
        self.sport_name = sport_name
        self.teams_name = teams_name

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}.")
        else:
            print("Tickets unavailable!")

    def cancel_booking(self, booking_id):
        raise InvalidBookingIDException("Cancellation not supported for
sports tickets.")

# Customer class
class Customer:
    def __init__(self, customer_name, email, phone_number):
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number

    def display_customer_details(self):
        print("Customer Name:", self.customer_name)
        print("Email:", self.email)
        print("Phone Number:", self.phone_number)

# Booking class
class Booking:
    def __init__(self, booking_id, event, customers: Set[Customer],
num_tickets):
        self.booking_id = booking_id
        self.event = event
        self.customers = customers
        self.num_tickets = num_tickets

    def display_booking_details(self):
```

```
        print("Booking ID:", self.booking_id)
        print("Event:", self.event.event_name)
        print("Number of Tickets:", self.num_tickets)

# Interface for Event Service Provider
class IEventServiceProvider(ABC):
    @abstractmethod
    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue):
        pass

    @abstractmethod
    def get_event_details(self):
        pass

    @abstractmethod
    def get_available_no_of_tickets(self):
        pass

# Interface for Booking System Service Provider
class IBookingSystemServiceProvider(ABC):
    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        pass

    @abstractmethod
    def book_tickets(self, event_name, num_tickets, customer_array):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

    @abstractmethod
    def get_booking_details(self, booking_id):
        pass

# Implementation of Event Service Provider
class EventServiceProviderImpl(IEventServiceProvider):
    events: Set[Event] = set()

    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue):
        event_date = datetime.strptime(date, "%Y-%m-%d")
        event_time = datetime.strptime(time, "%H:%M").time()
        event = None

        if event_type.lower() == "movie":
            genre = input("Enter movie genre: ")
            actor_name = input("Enter actor's name: ")
            actress_name = input("Enter actress's name: ")
            event = Movie(event_name, event_date, event_time, venue,
total_seats, ticket_price, genre, actor_name, actress_name)
        elif event_type.lower() == "concert":
            artist = input("Enter artist's name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue,
total_seats, ticket_price, artist, concert_type)
        elif event_type.lower() == "sports":
            sport_name = input("Enter sport name: ")
            teams_name = input("Enter teams playing: ")
```

```
            event = Sport(event_name, event_date, event_time, venue,
total_seats, ticket_price, sport_name, teams_name)
        else:
            print("Invalid event type!")

        if event:
            self.events.add(event)
            return event

    def get_event_details(self):
        return self.events

    def get_available_no_of_tickets(self):
        available_tickets = sum([event.available_seats for event in
self.events])
        return available_tickets

# Implementation of Booking System Service Provider
# Implementation of Booking System Service Provider
import mysql.connector

class DBUtil:
    @staticmethod
    def getDBConn():
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",
                password="root",
                database="ticketbooking"
            )
            print("Connected to MySQL database")
            return conn
        except mysql.connector.Error as e:
            print("Error connecting to MySQL database:", e)
            return None

from typing import List
from mysql.connector import Error

class BookingSystemServiceProviderImpl(IBookingSystemServiceProvider):
    def __init__(self):
        self.conn = DBUtil.getDBConn()

    def create_event(self, event_id, event_name, date, time, venue_id,
total_seats, ticket_price, event_type):
        try:
            cursor = self.conn.cursor()
            ticket_price = round(float(ticket_price), 2)
            query = "INSERT INTO event (event_id, event_name, event_date,
event_time, venue_id, total_seats, available_seats, ticket_price,
event_type) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
            cursor.execute(query, (
            event_id, event_name, date, time, venue_id, total_seats,
total_seats, ticket_price, event_type))
            self.conn.commit()
            cursor.close()
            print("Event created successfully.")
        except Error as e:
            print("Error creating event:", e)
```

```
    def get_event_details(self) -> List[Event]:
        try:
            cursor = self.conn.cursor()
            query = "SELECT * FROM event"
            cursor.execute(query)
            events = []
            for event_data in cursor.fetchall():
                print(event_data)
            cursor.close()
            return events
        except Error as e:
            print("Error getting event details:", e)
            return []

    def get_available_no_of_tickets(self) -> int:
        try:
            cursor = self.conn.cursor()
            query = "SELECT SUM(available_seats) FROM event"
            cursor.execute(query)
            result = cursor.fetchone()
            cursor.close()
            return result[0] if result else 0
        except Error as e:
            print("Error getting available tickets:", e)
            return 0

    def calculate_booking_cost(self, num_tickets, ticket_price) -> float:
        return num_tickets * ticket_price

    def book_tickets(self, booking_id,eventname, num_tickets,
list_of_customers):
        try:
            cursor = self.conn.cursor()
            # Update available seats in the event table
            update_query = "UPDATE event SET available_seats =
available_seats - %s WHERE event_name = %s"
            cursor.execute(update_query, (num_tickets, eventname))

            # Insert a row into the booking table for each ticket booked
            for customer in list_of_customers:
                insert_customer_query = "INSERT INTO customer
(customer_name, email, phone_number) VALUES (%s, %s, %s)"
                cursor.execute(insert_customer_query,
(customer.customer_name, customer.email, customer.phone_number))
                customer_id = cursor.lastrowid

                # Fetch event_id based on event_name
                select_event_query = "SELECT event_id FROM event WHERE
event_name = %s"
                cursor.execute(select_event_query, (eventname,))
                event_id = cursor.fetchone()[0]  # Fetch the event_id

                # Insert a row into the booking table for each customer
                insert_booking_query = "INSERT INTO booking
(booking_id,customer_id, event_id, num_tickets) VALUES (%s,%s, %s, %s)"
                cursor.execute(insert_booking_query,
(booking_id,customer_id, event_id, num_tickets))

            self.conn.commit()
            cursor.close()
            print(f"{num_tickets} tickets booked for {eventname}.")
```

```
        except Error as e:
            print("Error booking tickets:", e)

    def cancel_booking(self, booking_id):
        try:
            cursor = self.conn.cursor()
            query = "SELECT * FROM booking WHERE booking_id = %s"
            cursor.execute(query, (booking_id,))
            booking = cursor.fetchone()
            if booking:
                eventname, num_tickets = booking[2], booking[3]
                query = "UPDATE event SET available_seats = available_seats
+ %s WHERE event_id = %s"
                cursor.execute(query, (num_tickets, eventname))
                query = "DELETE FROM booking WHERE booking_id = %s"
                cursor.execute(query, (booking_id,))
                self.conn.commit()
                print(f"Booking {booking_id} cancelled.")
            else:
                print("Booking not found.")
            cursor.close()
        except Error as e:
            print("Error cancelling booking:", e)

    def get_booking_details(self, booking_id) -> Optional[Booking]:
        try:
            cursor = self.conn.cursor()
            query = "SELECT * FROM booking WHERE booking_id = %s"
            cursor.execute(query, (booking_id,))
            booking_data = cursor.fetchone()
            cursor.close()

            if booking_data:
                print("num tickets:",booking_data[3])
                #print("total cost",booking_data[4])
                cursor = self.conn.cursor()
                query = "SELECT customer_name FROM customer WHERE
customer_id = %s"
                cursor.execute(query, (booking_data[2],))
                customernow = cursor.fetchone()[0]
                print("event booked for:",customernow)
            else:
                print("Booking not found.")
        except Error as e:
            print("Error retrieving booking details:", e)


# Ticket Booking System
class TicketBookingSystem:
    def __init__(self):
        self.booking_system_service_provider =
BookingSystemServiceProviderImpl()

    def main(self):
        while True:
            print("\nMenu:")
            print("1. Create Event")
            print("2. Display Event Details")
            print("3. Book Tickets")
            print("4. Cancel Booking")
            print("5. Get Booking Details")
```

```
            print("6. Get Available Seats")
            print("7. Exit")

            choice = input("Enter your choice: ")

            try:
                if choice == "1":
                    event_id = int(input("Enter event ID: "))
                    event_name = input("Enter event name: ")
                    date = input("Enter event date (YYYY-MM-DD): ")
                    time = input("Enter event time (HH:MM): ")
                    venue_name = input("Enter venue name: ")
                    total_seats = int(input("Enter total seats: "))
                    ticket_price = float(input("Enter ticket price: "))
                    event_type = input("Enter event type
(Movie/Sports/Concert): ")
                    # Call the create_event method with event_id parameter

                    # Fetch venue ID from the database based on the
provided venue name
                    venue_id = None
                    try:
                        cursor =
self.booking_system_service_provider.conn.cursor()
                        query = "SELECT venue_id FROM venu WHERE venue_name
= %s"
                        cursor.execute(query, (venue_name,))
                        venue_id = cursor.fetchone()[0]
                        cursor.close()
                    except Error as e:
                        print("Error fetching venue ID:", e)

                    if venue_id:

self.booking_system_service_provider.create_event(event_id,event_name,
date, time, venue_id,total_seats,

ticket_price, event_type)
                    else:
                        print("Venue not found. Please make sure the venue
name is correct.")


                elif choice == "2":
                    events =
self.booking_system_service_provider.get_event_details()
                    for event in events:
                        print("\nEvent Details:")
                        event.display_event_details()

                elif choice == "3":
                    event_name = input("Enter the event name to book
tickets: ")
                    num_tickets = int(input("Enter number of tickets to
book: "))
                    customers = []
                    for _ in range(num_tickets):
                        customer_name = input("Enter customer name: ")
                        email = input("Enter email: ")
                        phone_number = input("Enter phone number: ")
                        booking_id = int(input("enter booking id:"))
```

```
                        customer = Customer(customer_name, email,
phone_number)
                        customers.append(customer)

self.booking_system_service_provider.book_tickets(booking_id,event_name,
num_tickets, customers)



                elif choice == "4":
                    booking_id = int(input("Enter the booking ID to cancel:
"))

self.booking_system_service_provider.cancel_booking(booking_id)

                elif choice == "5":
                    booking_id = int(input("Enter the booking ID to get
details: "))

self.booking_system_service_provider.get_booking_details(booking_id)

                elif choice == "6":
                    available_seats =
self.booking_system_service_provider.get_available_no_of_tickets()
                    print("Available Seats:", available_seats)

                elif choice == "7":
                    print("Exiting...")
                    break

                else:
                    print("Invalid choice. Please try again.")
            except EventNotFoundException as e:
                print(f"Event not found: {e}")
            except InvalidBookingIDException as e:
                print(f"Invalid booking ID: {e}")
            except Exception as e:
                print(f"An error occurred: {e}")

if __name__ == "__main__":
    ticket_booking_system = TicketBookingSystem()
    ticket_booking_system.main()
```

```
Menu:
1. Create Event
2. Display Event Details
3. Book Tickets
4. Cancel Booking
5. Get Booking Details
6. Get Available Seats
7. Exit
Enter your choice: 1
Enter event ID: 67
Enter event name: Charlie puth concert
Enter event date (YYYY-MM-DD): 2024-05-05
Enter event time (HH:MM): 07:08
Enter venue name: Venue 1
Enter total seats: 56
Enter ticket price: 45
Enter event type (Movie/Sports/Concert): Concert
Event created successfully.
```

```
2. Display Event Details
3. Book Tickets
4. Cancel Booking
5. Get Booking Details
6. Get Available Seats
7. Exit
Enter your choice: 2
(1, 'Event 1', datetime.date(2024, 4, 8), datetime.timedelta(seconds=50400), 1, 200, 200, Decimal('25.00'), 'Sports', None)
(2, 'Event 2', datetime.date(2024, 4, 9), datetime.timedelta(seconds=54000), 2, 150, 100, Decimal('35.00'), 'Concert', None)
(3, 'Event 3', datetime.date(2024, 4, 10), datetime.timedelta(seconds=57600), 3, 300, 250, Decimal('20.00'), 'Movie', None)
(4, 'Event 4', datetime.date(2024, 4, 11), datetime.timedelta(seconds=61200), 4, 100, 50, Decimal('50.00'), 'Concert', None)
(5, 'Event 5', datetime.date(2024, 4, 12), datetime.timedelta(seconds=64800), 5, 250, 200, Decimal('30.00'), 'Sports', None)
(6, 'Event 6', datetime.date(2024, 4, 13), datetime.timedelta(seconds=68400), 6, 150, 100, Decimal('40.00'), 'Concert', None)
(7, 'Event 7', datetime.date(2024, 4, 14), datetime.timedelta(seconds=72000), 7, 200, 150, Decimal('45.00'), 'Movie', None)
(8, 'Event 8', datetime.date(2024, 4, 15), datetime.timedelta(seconds=75600), 8, 300, 250, Decimal('20.00'), 'Concert', None)
(9, 'Event 9', datetime.date(2024, 4, 16), datetime.timedelta(seconds=79200), 9, 100, 50, Decimal('60.00'), 'Sports', None)
(10, 'Event 10', datetime.date(2024, 4, 17), datetime.timedelta(seconds=82800), 10, 150, 100, Decimal('35.00'), 'Concert', None)
(11, 'Taylor Concert', datetime.date(2024, 4, 8), datetime.timedelta(seconds=50400), 1, 200, 200, Decimal('1500.00'), 'Concert', None)
(34, 'taylor', datetime.date(2024, 6, 6), datetime.timedelta(seconds=21960), 1, 789, 788, Decimal('45.65'), 'Concert', None)
(35, 'tlor', datetime.date(2024, 6, 6), datetime.timedelta(seconds=21960), 2, 78, 78, Decimal('45.65'), 'Sports', None)
(67, 'Charlie puth concert', datetime.date(2024, 5, 5), datetime.timedelta(seconds=25680), 1, 56, 56, Decimal('45.00'), 'Concert', None)
(88, 'football', datetime.date(2023, 9, 9), datetime.timedelta(seconds=32940), 2, 66, 66, Decimal('66.00'), 'Movie', None)
```

```
Menu:
1. Create Event
2. Display Event Details
3. Book Tickets
4. Cancel Booking
5. Get Booking Details
6. Get Available Seats
7. Exit
Enter your choice: 3
Enter the event name to book tickets: Charlie puth concert
Enter number of tickets to book: 1
Enter customer name: keerthika
Enter email: keeerthika@gmail.com
Enter phone number: 9483421568
enter booking id:98
1 tickets booked for Charlie puth concert.
```

```
Menu:
1. Create Event
2. Display Event Details
3. Book Tickets
4. Cancel Booking
5. Get Booking Details
6. Get Available Seats
7. Exit
Enter your choice: 4
Enter the booking ID to cancel: 98
Booking 98 cancelled.
```

```
Menu:
1. Create Event
2. Display Event Details
3. Book Tickets
4. Cancel Booking
5. Get Booking Details
6. Get Available Seats
7. Exit
Enter your choice: 5
Enter the booking ID to get details: 1
num tickets: 2
event booked for: Rahul Kumar
```

```
Menu:
1. Create Event
2. Display Event Details
3. Book Tickets
4. Cancel Booking
5. Get Booking Details
6. Get Available Seats
7. Exit
Enter your choice: 6
Available Seats: 2638
```

```
Menu:
1. Create Event
2. Display Event Details
3. Book Tickets
4. Cancel Booking
5. Get Booking Details
6. Get Available Seats
7. Exit
Enter your choice: 7
Exiting...
```