

Assignment 5

Control Structure

SaiPrabath Chowdary S

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.

```
def check_ticket_availability(available_tickets, no_of_tickets_to_book):  
    if available_tickets >= no_of_tickets_to_book:  
        remaining_tickets = available_tickets - no_of_tickets_to_book  
        print(f"Tickets available! Remaining tickets:  
        {remaining_tickets}")  
    else:  
        print("Tickets unavailable!")
```

```
# Example 1  
available_tickets = 50  
no_of_tickets_to_book = 3  
check_ticket_availability(available_tickets, no_of_tickets_to_book)
```

```
Tickets available! Remaining tickets: 47
```

```
# Example 2  
available_tickets = 5  
no_of_tickets_to_book = 7  
check_ticket_availability(available_tickets, no_of_tickets_to_book)
```

```
Tickets unavailable!
```

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", "Diamond". 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.

```
def calculate_ticket_cost(ticket_type, num_tickets):
    base_prices = {"Silver": 100, "Gold": 200, "Diamond": 300}
    if ticket_type in base_prices:
        base_price = base_prices[ticket_type]
        total_cost = base_price * num_tickets
        return total_cost
    else:
        return None

ticket_type = input("Enter ticket type (Silver/Gold/Diamond): ")
num_tickets = int(input("Enter number of tickets: "))
total_cost = calculate_ticket_cost(ticket_type, num_tickets)

if total_cost is not None:
    print(f"Total cost for {num_tickets} {ticket_type} tickets: {total_cost}")
else:
    print("Invalid ticket type!")
```

```
Enter ticket type (Silver/Gold/Diamond): Gold
Enter number of tickets: 4
Total cost for 4 Gold tickets: 800
```

Task 3: Looping

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

```
def calculate_ticket_cost(ticket_type, num_tickets):
    base_prices = {"Silver": 100, "Gold": 200, "Diamond": 300}
    if ticket_type in base_prices:
        base_price = base_prices[ticket_type]
        total_cost = base_price * num_tickets
        return total_cost
    else:
        return None
```

```

while True:
    ticket_type = input("Enter ticket type (Silver/Gold/Diamond), or type 'Exit' to quit: ")

    if ticket_type.lower() == "exit":
        print("Exiting ticket booking system.")
        break

    num_tickets = int(input("Enter number of tickets: "))
    total_cost = calculate_ticket_cost(ticket_type, num_tickets)

    if total_cost is not None:
        print(f"Total cost for {num_tickets} {ticket_type} tickets: {total_cost}")
    else:
        print("Invalid ticket type!")

```

```

Enter ticket type (Silver/Gold/Diamond), or type 'Exit' to quit: Diamond
Enter number of tickets: 5
Total cost for 5 Diamond tickets: 1500
Enter ticket type (Silver/Gold/Diamond), or type 'Exit' to quit: exit
Exiting ticket booking system.

```

Task 4: Class & Object

Create a Following classes with the following attributes and methods:

1. Event Class:

- **Attributes:**

- o event_name,
- o event_date DATE,
- o event_time TIME,
- o venue_name,
- o total_seats,
- o available_seats,
- o ticket_price DECIMAL,
- o event_type ENUM('Movie', 'Sports', 'Concert')

- **Methods and Constructors:**

- o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

- o **calculate_total_revenue()**: Calculate and return the total revenue based on the number of tickets sold.
- o **getBookedNoOfTickets()**: return the total booked tickets
- o **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.
- o **cancel_booking(num_tickets)**: Cancel the booking and update the available seats.
- o **display_event_details()**: Display event details, including event name, date time seat availability.

```
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

# Getter and Setter methods
def get_event_name(self):
    return self.event_name
def set_event_name(self, event_name):
    self.event_name = event_name

def get_event_date(self):
    return self.event_date
def set_event_date(self, event_date):
    self.event_date = event_date

def get_event_time(self):
    return self.event_time
def set_event_time(self, event_time):
    self.event_time = event_time
```

```
def get_venue_name(self):
    return self.venue_name
def set_venue_name(self, venue_name):
    self.venue_name = venue_name

def get_total_seats(self):
    return self.total_seats
def set_total_seats(self, total_seats):
    self.total_seats = total_seats

def get_available_seats(self):
    return self.available_seats
def set_available_seats(self, available_seats):
    self.available_seats = available_seats

def get_ticket_price(self):
    return self.ticket_price
def set_ticket_price(self, ticket_price):
    self.ticket_price = ticket_price

def get_event_type(self):
    return self.event_type
def set_event_type(self, event_type):
    self.event_type = event_type
```

```
# methods
```

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

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

```
def book_tickets(self, num_tickets):
    if num_tickets > self.available_seats:
        print("Insufficient seats available!")
        return False
    else:
        self.available_seats -= num_tickets
        print(f"{num_tickets} tickets booked successfully for the event '{self.event_name}'")
        return True
```

```
def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets > self.total_seats:
        print("Invalid number of tickets to cancel!")
        return False
    else:
        self.available_seats += num_tickets
        print(f"{num_tickets} tickets canceled successfully for the
event '{self.event_name}'")
        return True
```

```
def display_event_details(self):
    print("Event Details:")
    print(f"Event Name: {self.event_name}")
    print(f"Event Date: {self.event_date}")
    print(f"Event Time: {self.event_time}")
    print(f"Venue: {self.venue_name}")
    print(f"Total Seats: {self.total_seats}")
    print(f"Available Seats: {self.available_seats}")
    print(f"Ticket Price: {self.ticket_price}")
    print(f"Event Type: {self.event_type.value}")
```

2. Venue Class

- **Attributes:**

- o venue_name,
- o address

- **Methods and Constructors:**

- o **display_venue_details():** Display venue details.

- o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

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

```
def get_venue_name(self):
    return self.venue_name
def set_venue_name(self, venue_name):
    self.venue_name = venue_name
```

```
def get_address(self):  
    return self.address  
def set_address(self, address):  
    self.address = address
```

```
def display_venue_details(self):  
    print("Venue Details:")  
    print(f"Venue Name: {self.venue_name}")  
    print(f"Address: {self.address}")
```

3. Customer Class

- **Attributes:**

- o customer_name,
- o email,
- o phone_number

- **Methods and Constructors:**

- o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

- o **display_customer_details()**: Display customer details.

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

```
def get_customer_name(self):  
    return self.customer_name  
  
def set_customer_name(self, customer_name):  
    self.customer_name = customer_name  
  
def get_email(self):  
    return self.email  
  
def set_email(self, email):  
    self.email = email
```

```
def get_phone_number(self):  
    return self.phone_number  
  
def set_phone_number(self, phone_number):  
    self.phone_number = phone_number
```

```
def display_customer_details(self):  
    print("Customer Details:")  
    print(f"Name: {self.customer_name}")  
    print(f"Email: {self.email}")  
    print(f"Phone Number: {self.phone_number}")
```

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:**

- o **calculate_booking_cost(num_tickets):** Calculate and set the total cost of the booking.
- o **book_tickets(num_tickets):** Book a specified number of tickets for an event.
- o **cancel_booking(num_tickets):** Cancel the booking and update the available seats.
- o **getAvailableNoOfTickets():** return the total available tickets
- o **getEventDetails():** return event details from the event class

```
class Booking:  
    def __init__(self, event):  
        self.event = event  
        self.num_tickets = 0  
        self.total_cost = 0
```

```
def calculate_booking_cost(self, num_tickets):  
    self.num_tickets = num_tickets  
    self.total_cost = self.num_tickets * self.event.get_ticket_price()
```



```
def book_tickets(self, num_tickets):
    if self.event.book_tickets(num_tickets):
        self.num_tickets += num_tickets
        self.total_cost += num_tickets * self.event.get_ticket_price()
        print(f"{num_tickets} tickets booked successfully!")
        return True
    else:
        print("Failed to book tickets.")
        return False
```

```
def cancel_booking(self, num_tickets):
    if self.event.cancel_booking(num_tickets):
        self.num_tickets -= num_tickets
        self.total_cost -= num_tickets * self.event.get_ticket_price()
        print(f"{num_tickets} tickets canceled successfully!")
        return True
    else:
        print("Failed to cancel booking.")
        return False
```

```
def get_available_no_of_tickets(self):
    return self.event.get_available_seats()
```

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

Task 5: Inheritance and polymorphism

1. Inheritance

- Create a subclass **Movie** that inherits from **Event**. Add the following attributes and methods:

o Attributes:

1. genre: Genre of the movie (e.g., Action, Comedy, Horror).
2. ActorName
3. ActresName

o 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.

```

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(f"Genre: {self.genre}")
    print(f"Actor Name: {self.actor_name}")
    print(f"Actress Name: {self.actress_name}")

```

• Create another subclass **Concert** that inherits from **Event**. Add the following attributes and methods:

o **Attributes:**

1. artist: Name of the performing artist or band.
2. type: (Theatrical, Classical, Rock, Recital)

o **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.

```

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(f"Artist: {self.artist}")

```

- Create another subclass **Sports** that inherits from **Event**. Add the following attributes and methods:

- o **Attributes:**

1. sportName: Name of the game.
2. teamsName: (India vs Pakistan)

- o **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.

```
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(f"Sport Name: {self.sport_name}")
```

- Create a class **TicketBookingSystem** with the following methods:

- o **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.

- o **display_event_details(event: Event)**: Accepts an event object and calls its **display_event_details()** method to display event details.

- o **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.

- o **cancel_tickets(event: Event, num_tickets)**: cancel a specified number of tickets for an event.

o **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.

```
class TicketBookingSystem:
    events = []

    @classmethod
    def create_event(cls, event_name, event_date, event_time,
venue_name, total_seats, ticket_price, event_type):
        if event_type.lower() == "movie":
            genre = input("Enter movie genre: ")
            actor_name = input("Enter actor name: ")
            actress_name = input("Enter actress 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 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 name: ")
            event = Sports(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, sport_name, teams_name)

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

    cls.events.append(event)
    return event
```

```
@classmethod
def display_event_details(cls, event):
    event.display_event_details()
```

```
@classmethod
def book_tickets(cls, event, num_tickets):
    if event.book_tickets(num_tickets):
        print(f"{num_tickets} tickets booked successfully!")
        return event.calculate_total_revenue(num_tickets)
    else:
        print("Failed to book tickets.")
        return 0
```

```
@classmethod
def cancel_tickets(cls, event, num_tickets):
    if event.cancel_booking(num_tickets):
        print(f"{num_tickets} tickets canceled successfully!")
        return True
    else:
        print("Failed to cancel tickets.")
        return False
```

```
@classmethod
def main(cls):
    while True:

        print("\n1. Create Event\n2. Display Event Details\n3. Book
Tickets\n4. Cancel Tickets\n5. Exit")
        choice = input("Enter your choice: ")

        if choice == "1":
            event_name = input("Enter event name: ")
            event_date = input("Enter event date: ")
            event_time = input("Enter event time: ")
            total_seats = int(input("Enter total seats: "))
            ticket_price = float(input("Enter ticket price: "))
            event_type = input("Enter event type (movie/concert/sports): ")
            venue_name = input("Enter venue name: ")
            cls.create_event(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, event_type)
```

```

elif choice == "2":
    if cls.events:
        for index, event in enumerate(cls.events, start=1):
            print(f"\nEvent {index}:")
            cls.display_event_details(event)
    else:
        print("No events created yet.")

elif choice == "3":
    if cls.events:
        event_index = int(input("Enter event index to book tickets: "))
- 1

        if 0 <= event_index < len(cls.events):
            event = cls.events[event_index]
            num_tickets = int(input("Enter number of tickets to book:"))
            cls.book_tickets(event, num_tickets)
        else:
            print("Invalid event index.")
    else:
        print("No events created yet.")

elif choice == "4":
    if cls.events:
        event_index = int(input("Enter event index to cancel tickets:
")) - 1

        if 0 <= event_index < len(cls.events):
            event = cls.events[event_index]
            num_tickets = int(input("Enter number of tickets to cancel:"))
            cls.cancel_tickets(event, num_tickets)
        else:
            print("Invalid event index.")
    else:
        print("No events created yet.")

elif choice == "5":
    print("Exiting program.")
    break

else:
    print("Invalid choice. Please try again.")

```

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*:

```
from abc import ABC, abstractmethod

# Abstract

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 calculate_total_revenue(self, num_tickets):
    pass
```

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

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

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.

```
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, event_type="Movie")
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

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

    def calculate_total_revenue(self, num_tickets):
        return num_tickets * self.ticket_price

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            return True
        else:
            return False
```



```
def cancel_booking(self, num_tickets):
    self.available_seats += num_tickets
    return True
```

```
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, event_type="Concert")
        self.artist = artist
        self.concert_type = concert_type

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

    def calculate_total_revenue(self, num_tickets):
        return num_tickets * self.ticket_price

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            return True
        else:
            return False

    def cancel_booking(self, num_tickets):
        self.available_seats += num_tickets
        return True
```

```

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, event_type="Sports")
        self.sport_name = sport_name
        self.teams_name = teams_name

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

    def calculate_total_revenue(self, num_tickets):
        return num_tickets * self.ticket_price

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            return True
        else:
            return False

    def cancel_booking(self, num_tickets):
        self.available_seats += num_tickets
        return True

```

2. BookingSystem Abstraction:

- Create an abstract class **BookingSystem** that represents the ticket booking system. It should include the methods of TASK 2 **TicketBookingSystem**:

```

class BookingSystem(ABC):
    events = []

    @abstractmethod
    def create_event(self, event_name, event_date, event_time,
total_seats, ticket_price, event_type, venue_name):
        pass

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

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

    @abstractmethod
    def get_available_seats(self, event):
        pass

```

3. 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."

```

class TicketBookingSystem(BookingSystem):

    def create_event(self, event_name, event_date, event_time,
total_seats, ticket_price, event_type, venue_name):
        if event_type == "Movie":
            genre = input("Enter genre: ")
            actor_name = input("Enter actor name: ")
            actress_name = input("Enter actress name: ")
            event = Movie(event_name, event_date, event_time, venue_name,
total_seats, ticket_price, genre, actor_name, actress_name)

```

```
        elif event_type == "Concert":
            artist = input("Enter artist 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 == "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 book_tickets(self, event, num_tickets):
    if event.book_tickets(num_tickets):
        print(f"{num_tickets} tickets booked successfully!")
        return event.calculate_total_revenue(num_tickets)
    else:
        print("Failed to book tickets.")
        return 0
```

```
def cancel_tickets(self, event, num_tickets):
    if event.cancel_booking(num_tickets):
        print(f"{num_tickets} tickets canceled successfully!")
        return True
    else:
        print("Failed to cancel tickets.")
        return False
```

```
def get_available_seats(self, event):
    return event.available_seats
```

Task 7: Has A Relation / Association

Create a Following classes with the following attributes and methods:

1. Venue Class

- **Attributes:**

- o venue_name,
- o address

- **Methods and Constructors:**

- o **display_venue_details()**: Display venue details.
- o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

created in task 4

2. Event Class:

- **Attributes:**

- o event_name,
- o event_date DATE,
- o event_time TIME,
- o venue (reference of class **Venu**),
- o total_seats,
- o available_seats,
- o ticket_price DECIMAL,
- o event_type ENUM('Movie', 'Sports', 'Concert')

- **Methods and Constructors:**

- o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
- o **calculate_total_revenue()**: Calculate and return the total revenue based on the number of tickets sold.
- o **getBookedNoOfTickets()**: return the total booked tickets
- o **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.
- o **cancel_booking(num_tickets)**: Cancel the booking and update the available seats.
- o **display_event_details()**: Display event details, including event name, date time seat availability.

created in task 4

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*:
 - o Movie.
 - o Concert.
 - o Sport.

created in task 5

4. Customer Class

- **Attributes:**
 - o customer_name,
 - o email,
 - o phone_number,
- **Methods and Constructors:**
 - o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
 - o **display_customer_details()**: Display customer details.

created in task 4

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 Constructors:**
 - o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
 - o **display_booking_details()**: Display customer details.

```

class Booking:

    booking_id_counter = 0

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

    def display_booking_details(self):
        print(f"Booking ID: {self.booking_id}")
        print("Event Details:")
        self.event.display_event_details()
        print(f"Number of Tickets: {self.num_tickets}")
        print(f"Total Cost: {self.total_cost}")
        print(f"Booking Date: {self.booking_date}")
        self.customers.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**

- o array of events

- **Methods and Constructors:**

- o **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.

- o **calculate_booking_cost(num_tickets):** Calculate and set the total cost of the booking.

- o **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.

- o **cancel_booking(booking_id):** Cancel the booking and update the available seats.

- o **getAvailableNoOfTickets()**: return the total available tickets
- o **getEventDetails()**: return event details from the event class
- o 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."

created in task 5

Task 8: Interface/abstract class, and Single Inheritance, static variable

1. Create **Venue**, class as mentioned above Task 4.

created in task 4

2. Event Class:

• Attributes:

- o event_name,
- o event_date DATE,
- o event_time TIME,
- o venue (reference of class Venu),
- o total_seats,
- o available_seats,
- o ticket_price DECIMAL,
- o event_type ENUM('Movie', 'Sports', 'Concert')

• Methods and Constructors:

- o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

created in task 4

3. Create **Event** sub classes as mentioned in above Task 4.

created in task 4

4. Create a class **Customer** and **Booking** as mentioned in above Task 4.

created in 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.

```
from abc import ABC, abstractmethod
class IEventServiceProvider(ABC):
    @abstractmethod
    def create_event(self, event_name: str, date: str, time: str,
total_seats: int, ticket_price: float, event_type: str, venue: Venue):
        pass

    @abstractmethod
    def get_event_details(self):
        pass

    @abstractmethod
    def get_available_no_of_tickets(self):
        pass
```

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.

```
from abc import ABC, abstractmethod
class IBookingSystemServiceProvider(IEventServiceProvider):
    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int):
        pass

    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int,
customers: list):
        pass
```

```

@abstractmethod
def cancel_booking(self, booking_id: int):
    pass

@abstractmethod
def get_booking_details(self, booking_id: int):
    pass

```

7. Create **EventServiceProviderImpl** class which implements **IEventServiceProvider** provide all implementation methods.

```

from service.IEventServiceProvider import IEventServiceProvider

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

    def create_event(self, event_name, event_date,
event_time,venue_name, total_seats, ticket_price, event_type ):
        if event_type == "Movie":
            genre = input("Enter genre: ")
            actor_name = input("Enter actor name: ")
            actress_name = input("Enter actress name: ")
            event = Movie(event_name, event_date, event_time,
venue_name, total_seats, ticket_price, genre, actor_name, actress_name)

        elif event_type == "Concert":
            artist = input("Enter artist 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 == "Sports":
            sport_name = input("Enter sport name: ")
            teams_name = input("Enter teams playing: ")
            event = Sports(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 get_event_details(self):
        self.event.display_event_details()

    def get_available_no_of_tickets(self):
        self.event.get_available_seats()

```

8. Create **BookingSystemServiceProviderImpl** class which implements **IBookingSystemServiceProvider** provide all implementation methods and inherits **EventServiceImpl** class with following attributes.

- **Attributes**

- o array of events

```

from service.IBookingSystemServiceProvider import
IBookingSystemServiceProvider
from bean.EventServiceImpl import EventServiceImpl

class BookingSystemServiceProviderImpl(EventServiceImpl,
IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__()

    def calculate_booking_cost(self, num_tickets: int):
        return self.get_ticket_price() * num_tickets

    def book_tickets(self, event, num_tickets):
        if event.book_tickets(num_tickets):
            print(f"{num_tickets} tickets booked successfully!")
            return event.calculate_total_revenue(num_tickets)
        else:
            print("Failed to book tickets.")
            return 0

    def cancel_tickets(self, event, num_tickets):
        if event.cancel_booking(num_tickets):
            print(f"{num_tickets} tickets canceled successfully!")
            return True

```

```

        else:
            print("Failed to cancel tickets.")
            return False

    def get_booking_details(self, booking_id: int):
        self.display_booking_details()

```

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.

created in task 5

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.

```

class EventNotFoundException(Exception):
    def __init__(self, event_name):
        self.event_name = event_name
        super().__init__(f"Event '{event_name}' not found.")

```

```

class InvalidBookingIDException(Exception):
    def __init__(self, booking_id):
        self.booking_id = booking_id
        super().__init__(f"Invalid booking ID '{booking_id}'")

class NullPointerException(Exception):
    def __init__(self, message="Null pointer exception
occurred."):
        super().__init__(message)

```

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.

```

from typing import List

class Booking:
    def __init__(self, booking_id, customers: List[Customer], event:
Event, num_tickets, total_cost, booking_date):
        self.booking_id = booking_id
        self.customers = customers
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date

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

```

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.

```

from typing import Set

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

class BookingSystem:
    def __init__(self):
        self.events = set()

```

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.

```

from typing import Dict

class Booking:
    def __init__(self, booking_id, customers: Dict[int, Customer],
event: Event, num_tickets, total_cost, booking_date):
        self.booking_id = booking_id
        self.customers = customers
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date

class BookingSystem:
    def __init__(self):
        self.events = {}

```

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 **IBookingSystemRepository** 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.

```
from abc import ABC, abstractmethod
class IBookingSystemRepository(ABC):
    @abstractmethod
    def create_event(self, event_name, event_date, event_time,
venue_name, total_seats, ticket_price, event_type):
        pass

    @abstractmethod
    def getEventDetails(self) :
        pass

    @abstractmethod
    def getAvailableNoOfTickets(self) :
        pass

    @abstractmethod
    def calculate_booking_cost(self, num_tickets, ticket_price):
        pass
```

```

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

@abstractmethod
def cancel_booking(self, booking_id: int) :
    pass

@abstractmethod
def get_booking_details(self, booking_id: int) :
    pass

```

5. Create **BookingSystemRepositoryImpl** interface/abstract class which implements **IBookingSystemRepository** interface/abstract class and provide implementation of all methods and perform the database operations.

```

from TicketBookingSystem.entity.movie import Movie
from TicketBookingSystem.entity.concert import Concert
from TicketBookingSystem.entity.sports import Sports
from TicketBookingSystem.entity.booking import Booking
from TicketBookingSystem.service.IBookingSystemRepository import
IBookingSystemRepository

class BookingSystemRepositoryImpl(IBookingSystemRepository):

    def __init__(self, db_conn):
        self.db_conn = db_conn
        self.events = {}
        self.bookings = {}

    def create_event(self, event_name, event_date, event_time,
venue_name, total_seats, ticket_price, event_type):

        if event_type.lower() == "movie":
            genre = input("Enter movie genre: ")
            actor_name = input("Enter actor name: ")
            actress_name = input("Enter actress 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 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 name: ")
            event = Sports(event_name, event_date, event_time,
venue_name, total_seats, ticket_price, sport_name, teams_name)
        else:
            print("Invalid event type.")
            return None

    try:
        cursor = self.db_conn.cursor()
        cursor.execute(
            "INSERT INTO Event (event_id, event_name, event_date,
event_time, venue_name, total_seats, available_seats, ticket_price,
event_type) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)",
            (event.event_id, event_date, event_time, venue_name,
total_seats, total_seats, ticket_price, event_type))
        self.db_conn.commit()
        self.events[event.event_id] = event
        return event
    except mysql.connector.Error as e:
        print(f"Error creating event: {e}")

def getEventDetails(self) :
    try:
        cursor = self.db_conn.cursor()
        cursor.execute("SELECT * FROM Event")
        events = cursor.fetchall()
        cursor.close()
        return events
    except mysql.connector.Error as e:
        raise exception (f"Error fetching event details: {e}")

```

```

def getAvailableNoOfTickets(self) :
    try:
        cursor = self.db_conn.cursor()
        cursor.execute("SELECT SUM(available_seats) FROM Event")
        total_available_tickets = cursor.fetchone()[0]
        cursor.close()
        print(f"total available tickets ={total_available_tickets}")
        return total_available_tickets
    except mysql.connector.Error as e:
        raise exception(f"Error fetching available tickets: {e}")

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

def book_tickets(self, event, num_tickets, customer) :
    try:
        cursor = self.db_conn.cursor()
        cursor.execute("SELECT available_seats FROM Event WHERE
event_name = %s", (event.event_id))
        available_seats = cursor.fetchone()[0]

        if available_seats >= num_tickets:
            cursor.execute("UPDATE Event SET available_seats =
available_seats - %s WHERE event_id = %s",
(num_tickets, event.event_id))
            self.db_conn.commit()

            booking = Booking(event, num_tickets, customer)
            self.bookings[booking.booking_id] = booking

            cursor.execute(
"INSERT INTO Booking (booking_id, event_id, customer_id,
num_tickets, total_cost, booking_date) VALUES (%s, %s, %s, %s, %s, %s)",
(booking.booking_id, event.event_id, customer.customer_id,
num_tickets, booking.total_cost, booking.booking_date))

            self.db_conn.commit()
            cursor.close()
            print(f"{num_tickets} tickets booked successfully for
{event.event_name}, booking id = {booking.booking_id}")
            return booking
        else:

```

```

        print("Not enough available seats for booking")
    except mysql.connector.Error as e :
        raise exception ProcessLookupError(f"Error booking tickets:
{e}")

    def cancel_booking(self, booking_id):
        try:
            cursor = self.db_conn.cursor()
            cursor.execute("SELECT event_id, num_tickets FROM Booking
WHERE booking_id = %s", (booking_id,))
            booking_details = cursor.fetchone()

            if booking_details:
                event_id, num_tickets = booking_details
                cursor.execute("UPDATE Event SET available_seats =
available_seats + %s WHERE event_id = %s",
                (num_tickets, event_id))
                cursor.execute("DELETE FROM Booking WHERE booking_id = %s",
                (booking_id,))
                self.db_conn.commit()
                cursor.close()
            else:
                print("Booking ID not found")
        except:
            print(f"Error cancelling booking: {None}")

    def get_booking_details(self, booking_id):
        try:
            cursor = self.db_conn.cursor()
            cursor.execute("SELECT * FROM Booking WHERE booking_id = %s",
            (booking_id,))
            booking_details = cursor.fetchone()
            cursor.close()
            return booking_details
        except mysql.connector.Error as e :
            raise exception(f"Error fetching booking details: {None}")

```

6. Create **DBUtil** class and add the following method.

- **static getDBConn():Connection** Establish a connection to the database and return Connection reference

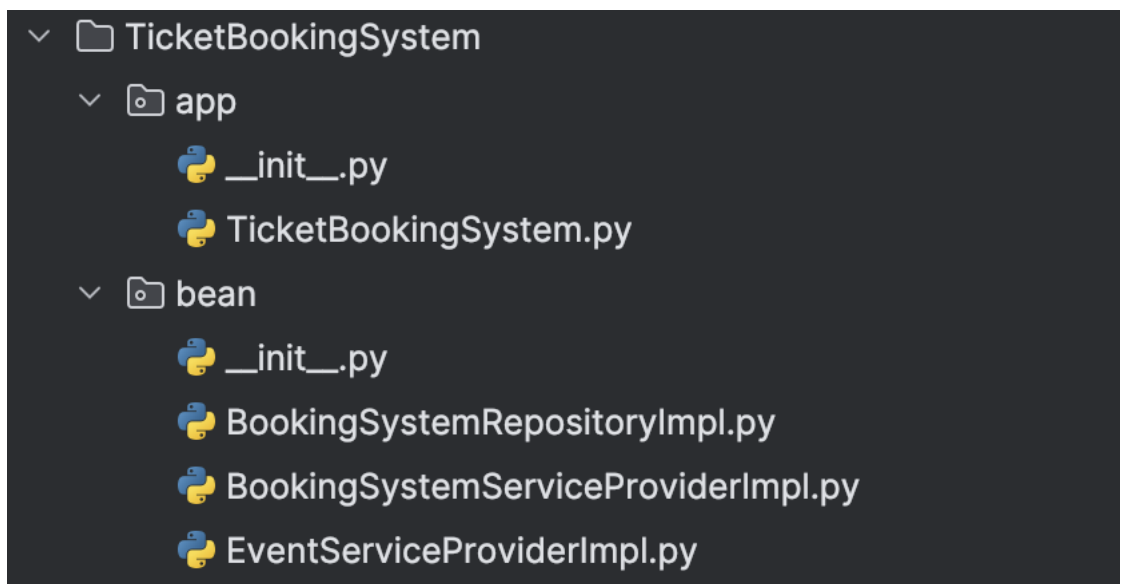
```

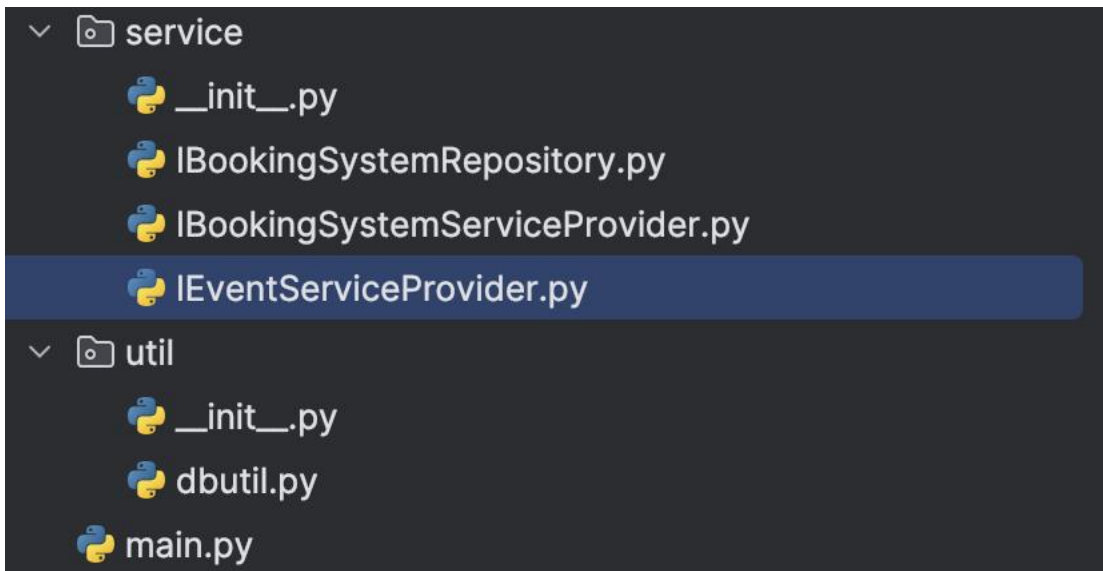
import mysql.connector

class DBUtil:
    @staticmethod
    def getDBConn():
        try:
            conn = (mysql.connector.connect(
                host="localhost",
                user="root",
                password="root",
                port='3306',
                database="TBS"
            ))
            return conn
        except mysql.connector.Error as e:
            print(f"Error connecting to MySQL database: {e}")
            return None

```

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 TicketBookingSystem.bean.BookingSystemRepositoryImpl import
BookingSystemRepositoryImpl
from TicketBookingSystem.entity.customer import Customer
```

```
class TicketBookingSystem:
def __init__(self, db_conn):
self.booking_system_repository = BookingSystemRepositoryImpl(db_conn)

def display_menu(self):
    print("==== Ticket Booking System =====")
    print("1. Create Event")
    print("2. View Event Details")
    print("3. Book Tickets")
    print("4. Cancel Booking")
    print("5. View Available Seats")
    print("6. Exit")
```

```
def create_event(self):
    event_name = input("Enter event name: ")
    event_date = input("Enter event date (YYYY-MM-DD): ")
    event_time = input("Enter event time (HH:MM:SS): ")
```

```
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): ")
self.booking_system_repository.create_event(event_name, event_date,
event_time, venue_name, total_seats, ticket_price, event_type)
```

```
def view_event_details(self):
    event_details = self.booking_system_repository.getEventDetails()
    if event_details:
        for event in event_details:
            print(event)
    else:
        print("no events found")
```

```
def book_tickets(self):
    self.view_event_details()
    event_id = int(input("Enter event ID to book tickets: "))
    try:
        event = self.booking_system_repository.events[event_id]
    except:
        print("event id not found")
        return None

    num_tickets = int(input("Enter number of tickets to book: "))

    # Creating customer
    customer_name = input("Enter customer name: ")
    email = input("Enter email: ")
    phone_number = input("Enter phone number: ")
    customer = Customer(customer_name, email, phone_number)

    # Book tickets
    self.booking_system_repository.book_tickets(event, num_tickets,
customer)
```

```
def cancel_booking(self):
    booking_id = int(input("Enter booking ID to cancel: "))
    self.booking_system_repository.cancel_booking(booking_id)
```

```
def view_available_seats(self):
    available_seats =
self.booking_system_repository.getAvailableNoOfTickets()
    print(f"Total available seats: {available_seats}")
```

```
def run(self):
while True:
self.display_menu()
choice = input("Enter your choice: ")
if choice == "1":
self.create_event()
elif choice == "2":
self.view_event_details()
elif choice == "3":
self.book_tickets()
elif choice == "4":
self.cancel_booking()
elif choice == "5":
self.view_available_seats()
elif choice == "6":
print("Exiting...")
break
else:
print("Invalid choice. Please enter a valid option.")
```