Assignment-5

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:

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

```
def book_tickets(available_tickets, num_booking_tickets):
    if available_tickets >= num_booking_tickets:
        remaining_tickets = available_tickets - num_booking_tickets
        print(f"Tickets booked successfully! Remaining tickets: {remaining_tickets}")
    else:
        print("Sorry, tickets unavailable.")

available_tickets = int(input("Enter the number of available tickets: "))
num_booking_tickets = int(input("Enter the number of tickets to book: "))
book_tickets(available_tickets, num_booking_tickets)
```

```
Enter the number of available tickets: 5
Enter the number of tickets to book: 4
Tickets booked successfully! Remaining tickets: 1

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.

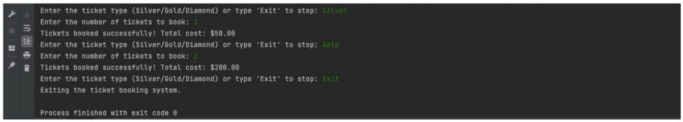
```python
def calculate_ticket_cost(ticket_type, num_tickets):
    silver_price = 50.0
    gold_price = 100.0
    diamond_price = 150.0

    if ticket_type == "Silver":
        total_cost = silver_price * num_tickets
    elif ticket_type == "Gold":
        total_cost = gold_price * num_tickets
    elif ticket_type == "Diamond":
        total_cost = diamond_price * num_tickets
    else:
        print("Invalid ticket type. Please choose from Silver, Gold, or Diamond.")
        return None

    return total_cost


ticket_type = input("Enter the ticket type (Silver/Gold/Diamond): ")
num_tickets = int(input("Enter the number of tickets to book: "))
total_cost = calculate_ticket_cost(ticket_type, num_tickets)
if total_cost is not None:
    print(f"Tickets booked successfully! Total cost: ${total_cost:.2f}")
```

```
Enter the ticket type (Silver/Gold/Diamond): Silver
Enter the number of tickets to book: 2
Tickets booked successfully! Total cost: $100.00

Process finished with exit code 0
```

Task 3: Looping

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

```python
def calculate_ticket_cost(ticket_type, num_tickets):
    silver_price = 50.0
    gold_price = 100.0
    diamond_price = 150.0

    if ticket_type == "Silver":
        total_cost = silver_price * num_tickets
    elif ticket_type == "Gold":
        total_cost = gold_price * num_tickets
    elif ticket_type == "Diamond":
        total_cost = diamond_price * num_tickets
    else:
        print("Invalid ticket type. Please choose from Silver, Gold, or Diamond.")
        return None

    return total_cost

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

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

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

    if total_cost is not None:
        print(f"Tickets booked successfully! Total cost: ${total_cost:.2f}")
```



Task 4: Class & Object
Create a Following classes with the following attributes and methods:

1. Event Class**:**
• Attributes:

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

• Methods and Constuctors:
- Implement default constructors and overload the constructor with Customer

attributes, generate getter and setter, (print all information of attribute) methods for

the attributes.
- **calculate_total_revenue()**: Calculate and return the total revenue based on the

number of tickets sold.
- **getBookedNoOfTickets()**: return the total booked tickets
- **book_tickets(num_tickets)**: Book a specified number of tickets for an event. Initially

available seats are equal to the total seats when tickets are booked available seats

number should be reduced.
- **cancel_booking(num_tickets)**: Cancel the booking and update the available seats. - **display_event_details():** Display event details, including event name, date time seat

```python
from datetime import date, time, datetime
from enum import Enum

class EventType(Enum):
    MOVIE = 'Movie'
    SPORTS = 'Sports'
    CONCERT = 'Concert'

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
        self.booked_tickets = 0

    # 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
```

```python
    def set_event_date(self, event_date):
        self.event_date = event_date

    def get_event_time(self):
        return self.event_

    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

    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 Name: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Ticket Price: ${self.ticket_price:.2f}")
        print(f"Event Type: {self.event_type}")

    def calculate_total_revenue(self):
        return self.ticket_price * self.booked_tickets

    def get_booked_no_of_tickets(self):
        return self.booked_tickets

    def book_tickets(self, num_tickets):
        if num_tickets > 0 and num_tickets <= self.available_seats:
            self.booked_tickets += num_tickets
            self.available_seats -= num_tickets
```

```
    def book_tickets(self, num_tickets):
        if num_tickets > 0 and num_tickets <= self.available_seats:
            self.booked_tickets += num_tickets
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked successfully!")
        else:
            print("Invalid number of tickets or not enough available seats.")

    def cancel_booking(self, num_tickets):
        if num_tickets > 0 and num_tickets <= self.booked_tickets:
            self.booked_tickets -= num_tickets
            self.available_seats += num_tickets
            print(f"{num_tickets} tickets canceled successfully!")
        else:
            print("Invalid number of tickets or not enough booked tickets.")
```

```
Event Details:
Event Name: New Year Concert
Event Date: 2023-12-31
Event Time: 18:30:00
Venue Name: Concert Hall
Total Seats: 1000
Available Seats: 1000
Ticket Price: $75.00
Event Type: EventType.CONCERT
5 tickets booked successfully!
```

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.

```
from datetime import date, time
from enum import Enum

class EventType(Enum):
    MOVIE = 'Movie'
    SPORTS = 'Sports'
    CONCERT = 'Concert'

class Venue:
    def __init__(self, venue_name, address):
        self.venue_name = venue_name
        self.address = address
    # Getter and Setter methods
    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:**

  - 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.

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

    # Getter and Setter methods
    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"Customer 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:**
○ **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

```python
class Booking:
    def __init__(self, event, customer, num_tickets):
        self.event = event
        self.customer = customer
        self.num_tickets = num_tickets
        self.total_cost = 0.0

    def calculate_booking_cost(self):
        self.total_cost = self.event.ticket_price * self.num_tickets

    def book_tickets(self):
        if self.num_tickets > 0 and self.num_tickets <= self.event.available_seats:
            self.event.book_tickets(self.num_tickets)
            self.calculate_booking_cost()
            print(f"{self.num_tickets} tickets booked successfully! Total cost: ${self.total_cost:.2f}")
        else:
            print("Invalid number of tickets or not enough available seats.")

    def cancel_booking(self):
        if self.num_tickets > 0 and self.num_tickets <= self.event.booked_tickets:
            self.event.cancel_booking(self.num_tickets)
            self.calculate_booking_cost()  # Update total cost after canceling tickets
            print(f"{self.num_tickets} tickets canceled successfully! Total cost: ${self.total_cost:.2f}")
        else:
            print("Invalid number of tickets or not enough booked tickets.")

    def get_available_no_of_tickets(self):
        return self.event.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: ○ **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.

```python
class EventType(Enum):
    MOVIE = 'Movie'
    SPORTS = 'Sports'
    CONCERT = 'Concert'

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

    def get_genre(self):
        return self.genre

    def set_genre(self, genre):
        self.genre = genre

    def get_actor_name(self):
        return self.actor_name

    def set_actor_name(self, actor_name):
        self.actor_name = actor_name

    def get_actress_name(self):
        return self.actress_name

    def set_actress_name(self, actress_name):
        self.actress_name = actress_name
```

```python
    def set_actor_name(self, actor_name):
        self.actor_name = actor_name

    def get_actress_name(self):
        return self.actress_name

    def set_actress_name(self, actress_name):
        self.actress_name = actress_name

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

if __name__ == "__main__":
    movie_date = date(2023, 1, 15)
    movie_time = time(20, 0)
    movie_type = EventType.MOVIE
    movie = Movie("Blockbuster Movie", movie_date, movie_time, "Cinema City", 500, 10.0, movie_type,
                  "Action", "John Doe", "Jane Doe")

    movie.display_event_details()
```

- 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:**

    3. Implement default constructors and overload the constructor with Customer

       attributes, generate getter and setter methods.

    4. **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, event_type,
                 artist, concert_type):
        super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type

    # Getter and Setter methods
    def get_artist(self):
        return self.artist

    def set_artist(self, artist):
        self.artist = artist

    def get_concert_type(self):
        return self.concert_type

    def set_concert_type(self, concert_type):
        self.concert_type = concert_type

    def display_event_details(self):
        super().display_event_details()
        print(f"Artist: {self.artist}")
        print(f"Concert Type: {self.concert_type}")

if __name__ == "__main__":
    concert_date = date(2023, 2, 20)
    concert_time = time(19, 30)
    concert_type = EventType.CONCERT
    concert = Concert("Live Concert", concert_date, concert_time, "Music Arena", 1000, 50.0, concert_type,
                      "Rock Band", "Rock")
    concert.display_event_details()
```

- 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.

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

    # Getter and Setter methods
    def get_sport_name(self):
        return self.sport_name

    def set_sport_name(self, sport_name):
        self.sport_name = sport_name

    def get_teams_name(self):
        return self.teams_name

    def set_teams_name(self, teams_name):
        self.teams_name = teams_name
```

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.

```
from datetime import date, time

class EventType:
    MOVIE = 'Movie'
    SPORTS = 'Sports'
    CONCERT = 'Concert'

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

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 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 Name: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Ticket Price: ${self.ticket_price:.2f}")
        print(f"Event Type: {self.event_type}")

    def book_tickets(self, num_tickets):
        if 0 < num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            return self.ticket_price * num_tickets
        else:
            print("Not enough available seats.")
            return 0.0
```

```
    def cancel_tickets(self, num_tickets):
        if 0 < num_tickets <= (self.total_seats - self.available_seats):
            self.available_seats += num_tickets
            return self.ticket_price * num_tickets
        else:
            print("Invalid number of tickets to cancel.")
            return 0.0

class TicketBookingSystem:
    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue_name):
        event_date = date(*map(int, date.split('-')))
        event_time = time(*map(int, time.split(':')))
        return Event(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)

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

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

    def cancel_tickets(self, event, num_tickets):
        return event.cancel_tickets(num_tickets)

def main():
    ticket_booking_system = TicketBookingSystem()

    while True:
        print("\nMenu:")
        print("1. Create Event")
        print("2. Display Event Details")
        print("3. Book Tickets")
        print("4. Cancel Tickets")
        print("5. Exit")

        choice = input("Enter your choice (1-5): ")
```

```
        if choice == "1":
            event_name = input("Enter event name: ")
            date = input("Enter event date (YYYY-MM-DD): ")
            time_str = input("Enter event time (HH:MM): ")
            total_seats = int(input("Enter total seats: "))
            ticket_price = float(input("Enter ticket price: "))
            event_type = input("Enter event type (Movie/Sports/Concert): ")
            venue_name = input("Enter venue name: ")

            event = ticket_booking_system.create_event(event_name, date, time_str, total_seats, ticket_price, event_type, venue_name)
            print("Event created successfully!")

        elif choice == "2":
            if 'event' in locals():
                ticket_booking_system.display_event_details(event)
            else:
                print("No event created yet. Please create an event first.")

        elif choice == "3":
            if 'event' in locals():
                num_tickets = int(input("Enter the number of tickets to book: "))
                total_cost = ticket_booking_system.book_tickets(event, num_tickets)
                if total_cost > 0:
                    print(f"Tickets booked successfully! Total cost: ${total_cost:.2f}")
            else:
                print("No event created yet. Please create an event first.")

        elif choice == "4":
            if 'event' in locals():
                num_tickets = int(input("Enter the number of tickets to cancel: "))
                total_refund = ticket_booking_system.cancel_tickets(event, num_tickets)
                if total_refund > 0:
                    print(f"Tickets canceled successfully! Total refund: ${total_refund:.2f}")
            else:
                print("No event created yet. Please create an event first.")

        elif choice == "5":
            print("Exiting the Ticket Booking System.")
            break

        else:
            print("Invalid choice. Please enter a number between 1 and 5.")

if __name__ == "__main__":
    main()
```

```
Menu:
1. Create Event
2. Display Event Details
3. Book Tickets
4. Cancel Tickets
5. Exit
Enter your choice (1-5):
```

Task 6: Abstraction Requirements:

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.

```python
from abc import ABC, abstractmethod
from enum import Enum

class EventType(Enum):
    MOVIE = 'Movie'
    SPORTS = 'Sports'
    CONCERT = 'Concert'

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_tickets(self, num_tickets):
        pass
```

```python
class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
                 genre, actor_name, actress_name):
        super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
        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: {self.actor_name}")
        print(f"Actress: {self.actress_name}")

    def book_tickets(self, num_tickets):
        if 0 < num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            return self.ticket_price * num_tickets
        else:
            print("Not enough available seats.")
            return 0.0

    def cancel_tickets(self, num_tickets):
        if 0 < num_tickets <= (self.total_seats - self.available_seats):
            self.available_seats += num_tickets
            return self.ticket_price * num_tickets
        else:
            print("Invalid number of tickets to cancel.")
            return 0.0
```

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

    def display_event_details(self):
        super().display_event_details()
        print(f"Artist: {self.artist}")
        print(f"Concert Type: {self.concert_type}")

    def book_tickets(self, num_tickets):
        if 0 < num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            return self.ticket_price * num_tickets
        else:
            print("Not enough available seats.")
            return 0.0

    def cancel_tickets(self, num_tickets):
        if 0 < num_tickets <= (self.total_seats - self.available_seats):
            self.available_seats += num_tickets
            return self.ticket_price * num_tickets
        else:
            print("Invalid number of tickets to cancel.")
            return 0.0

class Sports(Event):
    def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
                 sport_name, teams_name):
        super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
        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}")
        print(f"Teams: {self.teams_name}")

    def book_tickets(self, num_tickets):
        if 0 < num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            return self.ticket_price * num_tickets
        else:
            print("Not enough available seats.")
            return 0.0

    def cancel_tickets(self, num_tickets):
        if 0 < num_tickets <= (self.total_seats - self.available_seats):
            self.available_seats += num_tickets
            return self.ticket_price * num_tickets
        else:
            print("Invalid number of tickets to cancel.")
            return 0.0
```

3. **BookingSystem** Abstraction:
• Create an abstract class **BookingSystem** that represents the ticket booking system. It should

include the methods of TASK 2 **TicketBookingSystem**:

```
from abc import ABC, abstractmethod
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

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

    def display_event_details(self, event):
        pass

    def book_tickets(self, event, num_tickets):
        pass

    def cancel_tickets(self, event, num_tickets):
        pass

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

. 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."

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

    def create_event(self, event_name, date, time_str, total_seats, ticket_price, event_type, venue_name):
        event_date = date(*map(int, date.split('-')))
        event_time = time(*map(int, time_str.split(':')))
        event = Event(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
        self.events.append(event)
        return event

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

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

    def cancel_tickets(self, event, num_tickets):
        return event.cancel_tickets(num_tickets)

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

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

```python
if __name__ == "__main__":
    ticket_booking_system = TicketBookingSystem()

    while True:
        print("\nMenu:")
        print("1. Create Event")
        print("2. Display Event Details")
        print("3. Book Tickets")
        print("4. Cancel Tickets")
        print("5. Get Available Seats")
        print("6. Exit")

        choice = input("Enter your choice (1-6): ")

        if choice == "1":
            event_name = input("Enter event name: ")
            date = input("Enter event date (YYYY-MM-DD): ")
            time_str = input("Enter event time (HH:MM): ")
            total_seats = int(input("Enter total seats: "))
            ticket_price = float(input("Enter ticket price: "))
            event_type_str = input("Enter event type (Movie/Concert/Sports): ")
            venue_name = input("Enter venue name: ")

            event_type = EventType[event_type_str.upper()]

            event = ticket_booking_system.create_event(event_name, date, time_str, total_seats, ticket_price, event_type, venue_name)
            print("Event created successfully!")

        elif choice == "2":
            event_index = int(input("Enter the index of the event to display details: "))
            if 0 <= event_index < len(ticket_booking_system.events):
                ticket_booking_system.display_event_details(ticket_booking_system.events[event_index])
```

```
            else:
                print("Invalid event index.")

        elif choice == "3":
            event_index = int(input("Enter the index of the event to book tickets: "))
            if 0 <= event_index < len(ticket_booking_system.events):
                num_tickets = int(input("Enter the number of tickets to book: "))
                total_cost = ticket_booking_system.book_tickets(ticket_booking_system.events[event_index], num_tickets)
                if total_cost > 0:
                    print(f"Tickets booked successfully! Total cost: ${total_cost:.2f}")
            else:
                print("Invalid event index.")

        elif choice == "4":
            event_index = int(input("Enter the index of the event to cancel tickets: "))
            if 0 <= event_index < len(ticket_booking_system.events):
                num_tickets = int(input("Enter the number of tickets to cancel: "))
                total_refund = ticket_booking_system.cancel_tickets(ticket_booking_system.events[event_index], num_tickets)
                if total_refund > 0:
                    print(f"Tickets canceled successfully! Total refund: ${total_refund:.2f}")
            else:
                print("Invalid event index.")

        elif choice == "5":
            event_index = int(input("Enter the index of the event to get available seats: "))
            if 0 <= event_index < len(ticket_booking_system.events):
                available_seats = ticket_booking_system.get_available_seats(ticket_booking_system.events[event_index])
                print(f"Available seats: {available_seats}")
            else:
                print("Invalid event index.")
```

```
        elif choice == "6":
            print("Exiting the Ticket Booking System.")
            break
```

```
Menu:
1. Create Event
2. Display Event Details
3. Book Tickets
4. Cancel Tickets
5. Get Available Seats
6. Exit
Enter your choice (1-6):
6. Exit
Enter your choice (1-6): 6
Exiting the Ticket Booking System.

Process finished with exit code 0
```

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.

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

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

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.

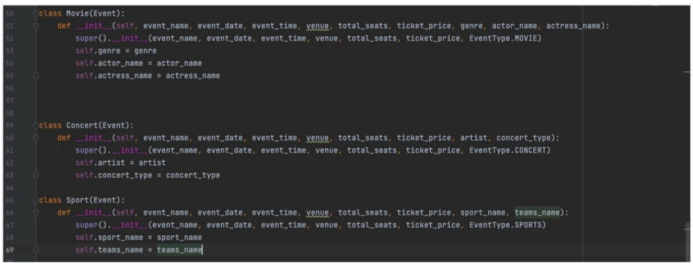
```
from enum import Enum
from datetime import date, time

class Event:
    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
        self.booked_tickets = 0


    def calculate_total_revenue(self):
        return self.ticket_price * self.booked_tickets

    def get_booked_no_of_tickets(self):
        return self.booked_tickets

    def book_tickets(self, num_tickets):
        if 0 < num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            self.booked_tickets += num_tickets
            return True
        else:
            print("Not enough available seats.")
            return False
```

```
    def cancel_booking(self, num_tickets):
        if 0 < num_tickets <= self.booked_tickets:
            self.available_seats += num_tickets
            self.booked_tickets -= num_tickets
            return True
        else:
            print("Invalid number of tickets to cancel.")
            return False
```

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.



```
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, EventType.MOVIE)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name



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, EventType.CONCERT)
        self.artist = artist
        self.concert_type = concert_type

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, EventType.SPORTS)
        self.sport_name = sport_name
        self.teams_name = teams_name
```

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.

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

    # Getter and Setter methods
    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
```

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:**

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

o **display_booking_details()**: Display customer details.

```python
from datetime import datetime
class Booking:
    booking_id_counter = 1

    def __init__(self, customers, event, num_tickets):
        self.booking_id = Booking.booking_id_counter
        Booking.booking_id_counter += 1

        if len(customers) != num_tickets:
            raise ValueError("Number of customers and number of tickets must be equal.")

        self.customers = customers
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = 0.0
        self.booking_date = datetime.now()

    def calculate_total_cost(self):
        self.total_cost = self.event.ticket_price * self.num_tickets
        return self.total_cost
```

6. **BookingSystem** Class to represent the Ticket booking system. Perform the following operation in

main method. Note: - Use Event class object for the following operation.

• **Attributes**
○ array of events

• **Methods and Constuctors:**
○ **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price:**

float, event_type: str, venu:Venu): Create a new event with the specified details and

event type (movie, sport or concert) and return event object.
○ **calculate_booking_cost(num_tickets)**: Calculate and set the total cost of the

booking.
○ **book_tickets(eventname:str, num_tickets, arrayOfCustomer)**: Book a specified

number of tickets for an event. for each tickets customer object should be created

and stored in array also should update the attributes of **Booking** class.
○ **cancel_booking(booking_id)**: Cancel the booking and update the available seats. ○ **getAvailableNoOfTickets()**: return the total available tickets
○ **getEventDetails()**: return event details from the event class
○ Create a simple user interface in a **main method** that allows users to interact with

the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats,", "get_event_details," and "exit."

```python
from typing import List

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

    def create_event(self, event_name, date, time_str, total_seats, ticket_price, event_type, venue):
        event_date = date(*map(int, date.split('-')))
        event_time = time(*map(int, time_str.split(':')))
        event = Event(event_name, event_date, event_time, venue, total_seats, ticket_price, event_type)
        self.events.append(event)
        return event

    def calculate_booking_cost(self, num_tickets):
        if num_tickets > 0:
            return num_tickets * self.selected_event.ticket_price
        else:
            print("Invalid number of tickets.")
            return 0.0

    def book_tickets(self, event_name, num_tickets, array_of_customers):
        for _ in range(num_tickets):
            customer_name = input("Enter customer name: ")
            email = input("Enter email: ")
            phone_number = input("Enter phone number: ")
            customer = Customer(customer_name, email, phone_number)
            array_of_customers.append(customer)

        total_cost = self.calculate_booking_cost(num_tickets)
        booking = Booking(array_of_customers, self.selected_event, num_tickets)
```

```python
        total_cost = self.calculate_booking_cost(num_tickets)
        booking = Booking(array_of_customers, self.selected_event, num_tickets)
        booking.total_cost = total_cost

        print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
        booking.display_booking_details()

    def cancel_booking(self, booking_id):
        for event in self.events:
            for booking in event.bookings:
                if booking.booking_id == booking_id:
                    event.available_seats += booking.num_tickets
                    event.bookings.remove(booking)
                    print(f"\nBooking with ID {booking_id} canceled successfully.")
                    return
        print(f"\nBooking with ID {booking_id} not found.")

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

    def get_event_details(self):
        self.selected_event.display_event_details()

    def display_menu(self):
        print("\nMenu:")
        print("1. Create Event")
        print("2. Get Event Details")
        print("3. Book Tickets")
        print("4. Cancel Booking")
        print("5. Get Available Seats")
```

```
    def main(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice (1-6): ")

            if choice == "1":
                event_name = input("Enter event name: ")
                date = input("Enter event date (YYYY-MM-DD): ")
                time_str = input("Enter event time (HH:MM): ")
                total_seats = int(input("Enter total seats: "))
                ticket_price = float(input("Enter ticket price: "))
                event_type_str = input("Enter event type (Movie/Concert/Sports): ")
                venue_name = input("Enter venue name: ")

                event_type = EventType[event_type_str.upper()]
                venue = Venue(venue_name, "Venue Address")  # You can provide the actual address

                event = self.create_event(event_name, date, time_str, total_seats, ticket_price, event_type, venue)
                print("Event created successfully!")

            elif choice == "2":
                event_index = int(input("Enter the index of the event to get details: "))
                if 0 <= event_index < len(self.events):
                    self.selected_event = self.events[event_index]
                    self.get_event_details()
```

```
            elif choice == "2":
                event_index = int(input("Enter the index of the event to get details: "))
                if 0 <= event_index < len(self.events):
                    self.selected_event = self.events[event_index]
                    self.get_event_details()
                else:
                    print("Invalid event index.")

            elif choice == "3":
                event_index = int(input("Enter the index of the event to book tickets: "))
                if 0 <= event_index < len(self.events):
                    self.selected_event = self.events[event_index]
                    num_tickets = int(input("Enter the number of tickets to book: "))
                    array_of_customers = []
                    self.book_tickets(self.selected_event.event_name, num_tickets, array_of_customers)
                else:
                    print("Invalid event index.")

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

            elif choice == "5":
                print(f"\nAvailable Seats: {self.get_available_no_of_tickets()}")

            elif choice == "6":
                print("Exiting the Ticket Booking System.")
                break

if __name__ == "__main__":
```

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

3. 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.

```python
from abc import ABC, abstractmethod
from typing import List

class IEventServiceProvider(ABC):
    @abstractmethod
    def create_event(self, event_name: str, date: str, time_str: str, total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue) -> Event:
        pass

    @abstractmethod
    def get_event_details(self) -> List[str]:
        pass

    @abstractmethod
    def get_available_no_of_tickets(self) -> int:
        pass
```

3. 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.

```python

from abc import ABC, abstractmethod
from typing import List

class IBookingSystemServiceProvider(ABC):
    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int) -> float:
        pass

    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, array_of_customers: List[Customer]) -> None:
        pass

    @abstractmethod
    def cancel_booking(self, booking_id: int) -> None:
        pass

    @abstractmethod
    def get_booking_details(self, booking_id: int) -> None:
        pass
```

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

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

    def create_event(self, event_name: str, date: str, time_str: str, total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue) -> Event:
        event_date = date(*map(int, date.split('-')))
        event_time = time(*map(int, time_str.split(':')))
        event = Event(event_name, event_date, event_time, venue, total_seats, ticket_price, event_type)
        self.events.append(event)
        return event

    def get_event_details(self) -> List[str]:
        event_details = []
        for event in self.events:
            event_details.append(event.event_name)
        return event_details

    def get_available_no_of_tickets(self) -> int:
        if not self.events:
            return 0
        return self.events[0].available_seats  # Assuming the first event in the list
```

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

• **Attributes**
- array of events

```
from typing import List

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

    def calculate_booking_cost(self, num_tickets: int) -> float:
        if num_tickets > 0:
            return num_tickets * self.selected_event.ticket_price
        else:
            print("Invalid number of tickets.")
            return 0.0

    def book_tickets(self, event_name: str, num_tickets: int, array_of_customers: List[Customer]) -> None:
        self.selected_event = None
        for event in self.events:
            if event.event_name == event_name:
                self.selected_event = event
                break

        if self.selected_event is not None:
            for _ in range(num_tickets):
                customer_name = input("Enter customer name: ")
                email = input("Enter email: ")
                phone_number = input("Enter phone number: ")
                customer = Customer(customer_name, email, phone_number)
                array_of_customers.append(customer)
```

```
            total_cost = self.calculate_booking_cost(num_tickets)
            booking = Booking(array_of_customers, self.selected_event, num_tickets)
            booking.total_cost = total_cost

            print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
            booking.display_booking_details()
        else:
            print(f"\nEvent with name '{event_name}' not found.")

    def cancel_booking(self, booking_id: int) -> None:
        for event in self.events:
            for booking in event.bookings:
                if booking.booking_id == booking_id:
                    event.available_seats += booking.num_tickets
                    event.bookings.remove(booking)
                    print(f"\nBooking with ID {booking_id} canceled successfully.")
                    return
        print(f"\nBooking with ID {booking_id} not found.")

    def get_booking_details(self, booking_id: int) -> None:
        for event in self.events:
            for booking in event.bookings:
                if booking.booking_id == booking_id:
                    print("\nBooking Details:")
                    booking.display_booking_details()
                    return
        print(f"\nBooking with ID {booking_id} not found.")
```

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."

```
class TicketBookingSystem:
    def __init__(self, booking_system_provider):
        self.booking_system_provider = booking_system_provider

    def display_menu(self):
        print("\nMenu:")
        print("1. Create Event")
        print("2. Get Event Details")
        print("3. Book Tickets")
        print("4. Cancel Booking")
        print("5. Get Available Seats")
        print("6. Exit")

    def main(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice (1-6): ")

            if choice == "1":
                event_name = input("Enter event name: ")
                date = input("Enter event date (YYYY-MM-DD): ")
                time_str = input("Enter event time (HH:MM): ")
                total_seats = int(input("Enter total seats: "))
                ticket_price = float(input("Enter ticket price: "))
                event_type_str = input("Enter event type (Movie/Concert/Sports): ")
                venue_name = input("Enter venue name: ")

                event_type = EventType[event_type_str.upper()]
                venue = Venue(venue_name, "Venue Address")  # You can provide the actual address
```

```
                self.booking_system_provider.create_event(event_name, date, time_str, total_seats, ticket_price,
                                                          event_type, venue)
                print("Event created successfully!")

            elif choice == "2":
                self.booking_system_provider.get_event_details()

            elif choice == "3":
                event_name = input("Enter the name of the event to book tickets: ")
                num_tickets = int(input("Enter the number of tickets to book: "))
                array_of_customers = []
                self.booking_system_provider.book_tickets(event_name, num_tickets, array_of_customers)

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

            elif choice == "5":
                print(f"\nAvailable Seats: {self.booking_system_provider.get_available_no_of_tickets()}")

            elif choice == "6":
                print("Exiting the Ticket Booking System.")
                break


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

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):
        super().__init__(message)

class TicketBookingSystem:
    def __init__(self, booking_system_provider):
        self.booking_system_provider = booking_system_provider
```

```
        except Exception as e:
            print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    try:
        booking_system_provider = BookingSystemServiceProviderImpl()
        ticket_booking_system = TicketBookingSystem(booking_system_provider)
        ticket_booking_system.main()
    except Exception as e:
        print(f"An unexpected 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.

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):
        super().__init__(message)

class TicketBookingSystem:
    def __init__(self, booking_system_provider):
        self.booking_system_provider = booking_system_provider
```

```
        except Exception as e:
            print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    try:
        booking_system_provider = BookingSystemServiceProviderImpl()
        ticket_booking_system = TicketBookingSystem(booking_system_provider)
        ticket_booking_system.main()
    except Exception as e:
        print(f"An unexpected 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.

```
class Booking:
    booking_id_counter = 1

    def __init__(self, customers: List[Customer], event: Event, num_tickets: int):
        self.booking_id = Booking.booking_id_counter
        Booking.booking_id_counter += 1
        self.customers = customers
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = 0.0
        self.booking_date = datetime.now()

class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__()
        self.events = []
```

1. 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.

```
class Booking:
    booking_id_counter = 1

    def __init__(self, customers: Set[Customer], event: Event, num_tickets: int):
        self.booking_id = Booking.booking_id_counter
        Booking.booking_id_counter += 1
        self.customers = customers
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = 0.0
        self.booking_date = datetime.now()

class EventComparator:
    def compare(self, event1: Event, event2: Event):
        if event1.event_name != event2.event_name:
            return event1.event_name < event2.event_name
        else:
            return event1.venue.venue_name < event2.venue.venue_name

class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__()
        self.events = set()

    def calculate_booking_cost(self, num_tickets: int) -> float:
        if num_tickets > 0:
            return num_tickets * self.selected_event.ticket_price
        else:
```

```
    def book_tickets(self, event_name: str, num_tickets: int, set_of_customers: Set[Customer]) -> None:
        self.selected_event = None
        for event in self.events:
            if event.event_name == event_name:
                self.selected_event = event
                break

        if self.selected_event is not None:
            for _ in range(num_tickets):
                customer_name = input("Enter customer name: ")
                email = input("Enter email: ")
                phone_number = input("Enter phone number: ")
                customer = Customer(customer_name, email, phone_number)
                set_of_customers.add(customer)

            total_cost = self.calculate_booking_cost(num_tickets)
            booking = Booking(set_of_customers, self.selected_event, num_tickets)
            booking.total_cost = total_cost

            self.selected_event.available_seats -= num_tickets
            self.selected_event.bookings.add(booking)

            print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
            booking.display_booking_details()
        else:
            raise EventNotFoundException(event_name)
```

```
    def cancel_booking(self, booking_id: int) -> None:
        for event in self.events:
            for booking in event.bookings:
                if booking.booking_id == booking_id:
                    event.available_seats += booking.num_tickets
                    event.bookings.remove(booking)
                    print(f"\nBooking with ID {booking_id} canceled successfully.")
                    return
        raise InvalidBookingIDException(booking_id)

    def get_booking_details(self, booking_id: int) -> None:
        for event in self.events:
            for booking in event.bookings:
                if booking.booking_id == booking_id:
                    print("\nBooking Details:")
                    booking.display_booking_details()
                    return
        raise InvalidBookingIDException(booking_id)

    def sort_events(self) -> List[Event]:
        comparator = EventComparator()
        return sorted(self.events, key=functools.cmp_to_key(comparator.compare))
```

Task 11: Database Connectivity.

1. 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.

```python
from typing import List
from abc import ABC, abstractmethod

class IBookingSystemRepository(ABC):
    @abstractmethod
    def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
                     event_type: str, venue: Venu) -> Event:
        pass

    @abstractmethod
    def get_event_details(self) -> List[Event]:
        pass

    @abstractmethod
    def get_available_no_of_tickets(self) -> int:
        pass

    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int) -> float:
        pass

    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, list_of_customers: List[Customer]) -> None:
        pass

    @abstractmethod
    def cancel_booking(self, booking_id: int) -> None:
        pass
```

```python

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

```

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

```
    def book_tickets(self, event_name: str, num_tickets: int, list_of_customers: List[Customer]) -> None:
        self.selected_event = None
        for event in self.events:
            if event.event_name == event_name:
                self.selected_event = event
                break

        if self.selected_event is not None:
            if self.selected_event.available_seats >= num_tickets:
                for _ in range(num_tickets):
                    customer_name = input("Enter customer name: ")
                    email = input("Enter email: ")
                    phone_number = input("Enter phone number: ")
                    customer = Customer(customer_name, email, phone_number)
                    list_of_customers.append(customer)

                total_cost = self.calculate_booking_cost(num_tickets)
                booking = Booking(set(list_of_customers), self.selected_event, num_tickets)
                booking.total_cost = total_cost

                self.selected_event.available_seats -= num_tickets
                self.bookings.add(booking)

                print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
                booking.display_booking_details()
            else:
                raise ValueError("Not enough available seats for booking.")
        else:
            raise EventNotFoundException(event_name)
```

```
from typing import List, Set

class BookingSystemRepositoryImpl(IBookingSystemRepository):
    def __init__(self):
        # Simulating a database with in-memory data structures
        self.events: List[Event] = []
        self.bookings: Set[Booking] = set()

    def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
                     event_type: str, venue: Venu) -> Event:
        new_event = Event(event_name, date, time, venue, total_seats, total_seats, ticket_price, EventType[event_type.upper()])
        self.events.append(new_event)
        return new_event

    def get_event_details(self) -> List[Event]:
        return self.events

    def get_available_no_of_tickets(self) -> int:
        total_available_tickets = sum(event.available_seats for event in self.events)
        return total_available_tickets

    def calculate_booking_cost(self, num_tickets: int) -> float:
        if num_tickets > 0 and self.selected_event:
            return num_tickets * self.selected_event.ticket_price
        else:
            raise ValueError("Invalid number of tickets or event not selected.")
```

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**.

```
from abc import ABC, abstractmethod

class IBookingSystemRepository(ABC):
    @abstractmethod
    def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
                     event_type: str, venue: Venu) -> Event:
        pass

    @abstractmethod
    def get_event_details(self) -> List[Event]:
        pass

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

    def handle_sql_exception(self):
        try:
            pass
        except SQLException as e:
            print(f"SQLException: {e}")
            raise e
```

7. 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."

```python
class TicketBookingSystem:
    def __init__(self, booking_system_repository: IBookingSystemRepository):
        self.booking_system_repository = booking_system_repository

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

    def create_event(self):
        event_name = input("Enter event name: ")
        date = input("Enter event date: ")
        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/Sports/Concert): ")
        venue_name = input("Enter venue name: ")
        venue_address = input("Enter venue address: ")
        venue = Venu(venue_name, venue_address)

        self.booking_system_repository.create_event(event_name, date, time, total_seats, ticket_price, event_type, venue)
        print("Event created successfully!")
```

```python
    def get_available_seats(self):
        try:
            # Get available seats using the repository
            available_seats = self.booking_system_repository.get_available_no_of_tickets()
            print(f"Available seats: {available_seats}")
        except SQLException as e:
            print(f"Error: {e}")

    def get_event_details(self):
        # Get event details using the repository
        event_details = self.booking_system_repository.get_event_details()
        print("\n===== Event Details =====")
        for event in event_details:
            print(event.display_event_details())

    def run_ticket_booking_system(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")

            if choice == "1":
                self.create_event()
            elif choice == "2":
                self.book_tickets()
            elif choice == "3":
                self.cancel_tickets()
            elif choice == "4":
                self.get_available_seats()
            elif choice == "5":
                self.get_event_details()
            elif choice == "6":
                print("Exiting Ticket Booking System. Goodbye!")
                break
            else:
                print("Invalid choice. Please enter a valid option.")
```