Ticket Booking

Control structure

Task 1: Conditional Statements

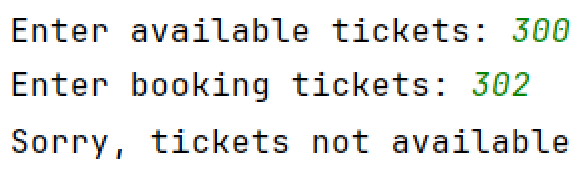
In a Booking System, you have been given the task is to create a program to book tickets. if available t ickets more than noOfTicket to book then display the remaining tickets or ticket unavailable:

Tasks: 1. Write a program that takes the available Ticket and noOfBookingTicket as input.

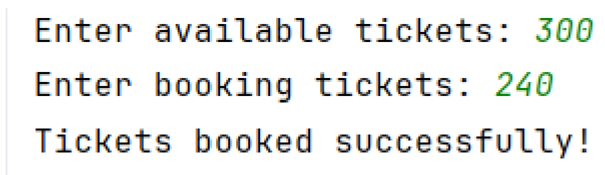
2. Use conditional statements (if-else) to determine if the ticket is available or not.

3. Display an appropriate message based on ticket availability.

```
def check_ticket_availability(available_tickets,booking_tickets):
    if(available_tickets>booking_tickets):
        return "Tickets booked successfully!"
    else:
        return "Sorry, tickets not available"
available_tickets=int(input("Enter available tickets: "))
booking_tickets=int(input("Enter booking tickets: "))
print(check_ticket_availability(available_tickets,booking_tickets))
```



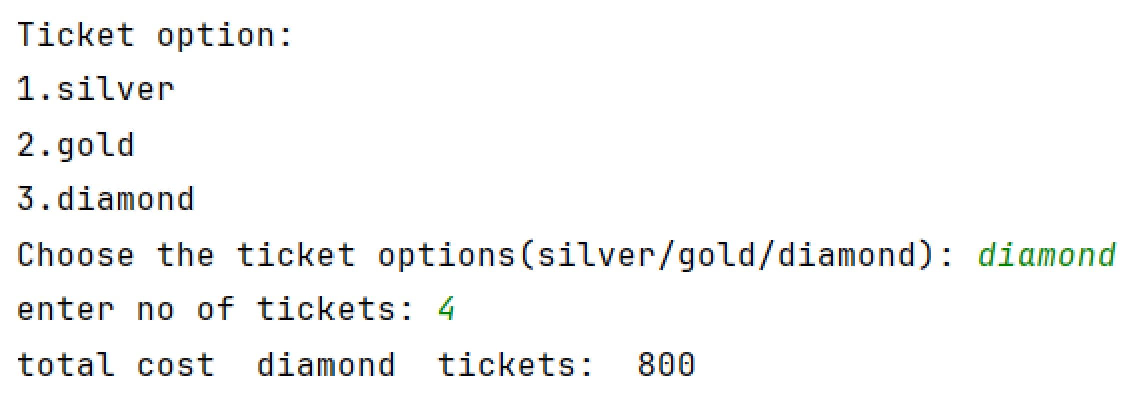
```
Enter available tickets: 300
Enter booking tickets: 302
Sorry, tickets not available
```



```
Enter available tickets: 300
Enter booking tickets: 240
Tickets booked successfully!
```

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.

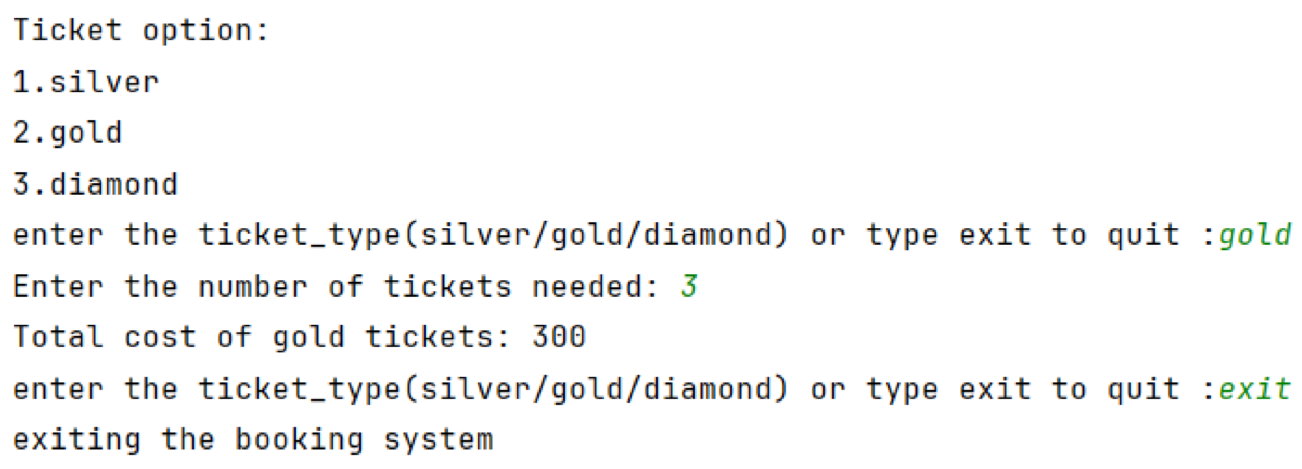
```
silver_ticket_price=50
gold_ticket_price=100
diamond_ticket_price=200

print("Ticket option: ")
print("1.silver")
print("2.gold")
print("3.diamond")

ticket_type=input("Choose the ticket options(silver/gold/diamond): ")
no_of_tickets=int(input("enter no of tickets: "))
total_cost=0
if(ticket_type=="silver"or"gold"or"diamond"):
 if(ticket_type=="silver"):
    total_cost=no_of_tickets*silver_ticket_price
 elif (ticket_type == "gold"):
    total_cost = no_of_tickets * gold_ticket_price
 elif (ticket_type == "diamond"):
    total_cost = no_of_tickets * diamond_ticket_price
else:
    print("invalid ticket type")
if(total_cost>0):
    print("total cost ",ticket_type," tickets: ",total_cost)
```



```
Ticket option:
1.silver
2.gold
3.diamond
Choose the ticket options(silver/gold/diamond): diamond
enter no of tickets: 4
total cost  diamond  tickets:  800
```

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

```
silver_ticket_price=50
gold_ticket_price=100
diamond_ticket_price=200

print("Ticket option: ")
print("1.silver")
print("2.gold")
print("3.diamond")

while True:
    ticket_type=input("enter the ticket_type(silver/gold/diamond) or type exit
to quit :")
    if(ticket_type=="exit"):
        print("exiting the booking system")
        break

    no_of_tickets = int(input("Enter the number of tickets needed: "))
    if ticket_type== "silver":
        total_cost = silver_ticket_price * no_of_tickets
    elif ticket_type == "gold":
        total_cost = gold_ticket_price * no_of_tickets
    elif ticket_type== "diamond":
        total_cost = diamond_ticket_price * no_of_tickets
    print(f"Total cost of {ticket_type} tickets: {total_cost}")
```



```
Ticket option:
1.silver
2.gold
3.diamond
enter the ticket_type(silver/gold/diamond) or type exit to quit :gold
Enter the number of tickets needed: 3
Total cost of gold tickets: 300
enter the ticket_type(silver/gold/diamond) or type exit to quit :exit
exiting the 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, available_seats, o ticket_price DECIMAL, o event_type ENUM('Movie', 'Sports', 'Concert') • Methods and Constuctors: 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.

```
from datetime import datetime

class Events:
    def __init__(self, event_name, event_date, event_time,
venue,total_seats,available_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=available_seats
        self.ticket_price=ticket_price
        self.event_type=event_type
        self.booked_tickets=0

    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(self):
            return self.venue

    def set_venue(self, venue):
            self.venue = venue

    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):
```

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

    def get_booked_no_of_tickets(self):
        return self.booked_tickets

    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.booked_tickets += num_tickets
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for the event {self.event_name}.")
        else:
            print("Not enough seats available.")

    def cancel_booking(self, num_tickets):
        if num_tickets <= self.booked_tickets:
            self.booked_tickets -= num_tickets
            self.available_seats += num_tickets
            print(f"{num_tickets} tickets canceled for the event {self.event_name}.")
        else:
            print("Invalid number of tickets to cancel.")

    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 Name: {self.venue}")
        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}")

event = Events("Movie Night", datetime(2024, 5, 10), "18:00", "Cinema Hall 1", 100, 54,45,
"Movie")
event.display_event_details()
print("Total Revenue:", event.calculate_total_revenue())
event.book_tickets(5)
event.cancel_booking(2)
```

2.Venue Class • Attributes: o venue_name, o address • Methods and Constuctors: o display_venue_details(): Display venue details.

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


venue = venue("Stadium", "123 Main Street")
venue.display_venue_details()
```

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

```
class Customers:
    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 Name:", self.customer_name)
        print("Email:", self.email)
        print("Phone Number:", self.phone_number)

    # Example usage:
customer_details = Customers("John Doe", "john@example.com", "123-456-7890")
customer_details.display_customer_details()
```

4.Booking Class to represent the Tiket booking system. Perform the following operation in main method. Note:- Use Event class object for the following operation. • Methods and Constuctors: 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

```
from event import Events
from datetime import datetime
class TicketBooking:
    def __init__(self, event):
        self.event = event
        self.total_cost = 0

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

    def book_tickets(self, num_tickets):
        if num_tickets <= self.event.get_available_seats():
            self.event.book_tickets(num_tickets)
            self.calculate_booking_cost(num_tickets)
            print(f"{num_tickets} tickets booked for the event
{self.event.get_event_name()}.")
        else:
            print("Not enough seats available.")

    def cancel_booking(self, num_tickets):
        self.event.cancel_booking(num_tickets)
        self.total_cost = 0

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

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

# Example usage:
# Assuming `event` is an instance of the Event class
eventobj=Events("Movie Night", datetime(2024, 5, 10), "18:00", "Cinema Hall 1", 100,
54,45, "Movie")
booking = TicketBooking(eventobj)

# Perform operations using the booking object
booking.book_tickets(5)
print("Available Tickets:", booking.get_available_no_of_tickets())
```

```python
print("Event Details:")
booking.get_event_details()
```

Task 5:Inheritance and Polymorphism

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.

```python
class Movie(Events):
    def __init__(self, event_name, event_date, event_time, venue, total_seats, 
available_seats, ticket_price, event_type,genre, actor_name, actress_name, customer=None):
        super().__init__(event_name, event_date, event_time, venue, total_seats, 
available_seats, ticket_price, "Movie", customer)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

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

action_movie = Movie("The Matrix", "2024-05-15", "18:00", "Cinema Hall 1", 100, 80, 10.00, 
"Action", "Keanu Reeves", "Carrie-Anne Moss")
action_movie.display_event_details()

horror_movie = Movie("The Conjuring", "2024-05-15", "18:00", "Scary Cinema", 150, 100, 
12.00, "Horror", "Vera Farmiga", "Patrick Wilson")
comedy_movie = Movie("The Hangover", "2024-06-20", "20:00", "Funny Theater", 200, 150, 
15.00, "Comedy", "Bradley Cooper", "Zach Galifianakis")

horror_movie.display_event_details()
comedy_movie.display_event_details()
```

2.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.

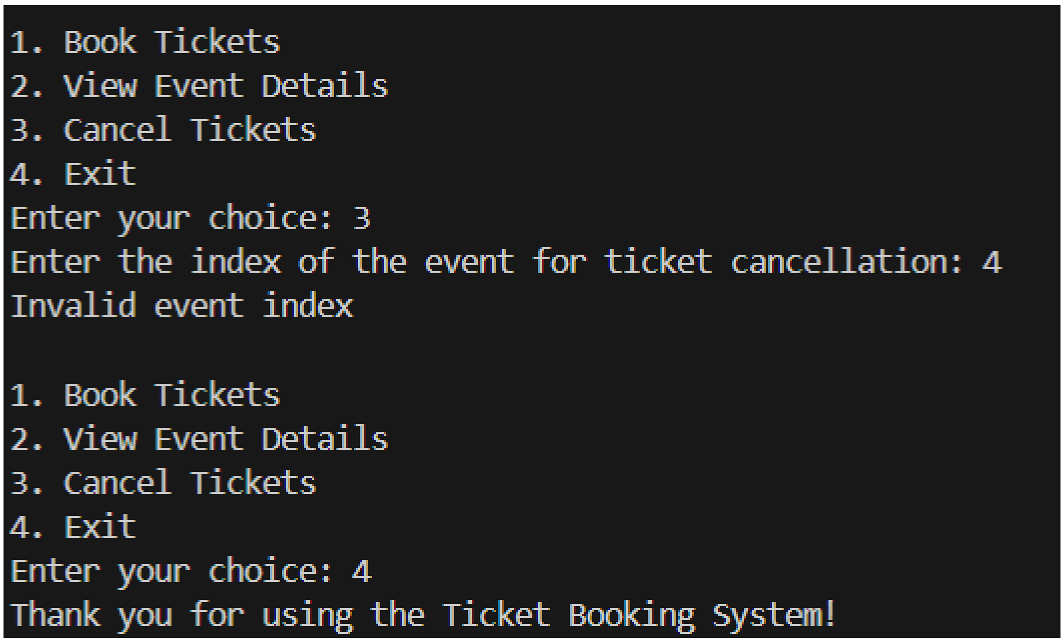
```python
class Concert(Events):
    class ConcertType(Enum):
        Theatrical = "Theatrical"
        Classical = "Classical"
        Rock = "Rock"
        Recital = "Recital"
    def __init__(self, event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price, artist=None, concert_type=None):
        super().__init__(event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price, "Concert")
        self.artist = artist
        self.concert_type = concert_type

    def display_concert_details(self):
        self.display_event_details()
        print(f"Artist: {self.artist}\nType: {self.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

concert1 = Concert("Rock On", "2024-05-15", "20:00:00", "Stadium A", 1000, 800, 50.00)
concert1.set_artist("The Rolling Stones")
concert1.set_concert_type(Concert.ConcertType.Rock.value)
concert1.display_concert_details()


concert2 = Concert("Classical Night", "2024-06-20", "19:30:00", "Opera House", 500, 400,
75.00, "Beethoven", Concert.ConcertType.Classical.value)
concert2.display_concert_details()
```

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

```python
#from event import Events
class Sports(Events):
    def __init__(self, event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price, sport_name=None, teams_name=None):
        super().__init__(event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price, "Sports")
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_sport_details(self):
        self.display_event_details()
        print(f"Sport: {self.sport_name}\nTeams: {self.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
# Example usage
if __name__ == "__main__":
    # Creating a Sports instance using default constructor
    sports_event1 = Sports("Football Match", "2024-05-20", "18:00:00", "Stadium B", 50000,
35000, 30.00)
    sports_event1.set_sport_name("Football")
    sports_event1.set_teams_name("Real Madrid vs Barcelona")
    sports_event1.display_sport_details()

    # Creating a Sports instance using overloaded constructor
    sports_event2 = Sports("Cricket Match", "2024-06-10", "15:00:00", "Cricket Ground",
20000, 15000, 20.00, "Cricket", "India vs Pakistan")
    sports_event2.display_sport_details()
```

Create a class TicketBookingSystem with the following methods: o create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: f loat, 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:
    def __init__(self):
        self.events = []

    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type,
venue_name):
        if event_type.lower() == 'movie':
            event = Movie(event_name, date, time, venue_name, total_seats, total_seats,
ticket_price)
        elif event_type.lower() == 'concert':
            event = Concert(event_name, date, time, venue_name, total_seats, total_seats,
ticket_price)
        elif event_type.lower() == 'sports':
            event = Sports(event_name, date, time, venue_name, total_seats, total_seats,
ticket_price)
        else:
            raise ValueError("Invalid 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):
        if event.available_seats >= num_tickets:
            event.available_seats -= num_tickets
            return num_tickets * event.ticket_price
        else:
            print("Sorry, the event is sold out.")
            return 0

    def cancel_tickets(self, event, num_tickets):
        event.available_seats += num_tickets

    def main(self):
        while True:
            print("\n1. Book Tickets")
            print("2. View Event Details")
            print("3. Cancel Tickets")
            print("4. Exit")

            choice = input("Enter your choice: ")

            if choice == '1':
                event_type = input("Enter event type (Movie/Sports/Concert): ")
                event_name = input("Enter event name: ")
                date = input("Enter date (YYYY-MM-DD): ")
                time = input("Enter time (HH:MM:SS): ")
                venue_name = input("Enter venue name: ")
                total_seats = int(input("Enter total seats: "))
                ticket_price = float(input("Enter ticket price: "))
                num_tickets = int(input("Enter number of tickets to book: "))

                event = self.create_event(event_name, date, time, total_seats,
ticket_price, event_type, venue_name)
                total_cost = self.book_tickets(event, num_tickets)
                print(f"Booking successful! Total cost: ${total_cost}")

            elif choice == '2':
                event_index = int(input("Enter the index of the event you want to view:
"))
                if 0 <= event_index < len(self.events):
                    self.display_event_details(self.events[event_index])
                else:
                    print("Invalid event index")

            elif choice == '3':
                event_index = int(input("Enter the index of the event for ticket 
cancellation: "))
                if 0 <= event_index < len(self.events):
                    event = self.events[event_index]
                    num_tickets = int(input("Enter number of tickets to cancel: "))
                    self.cancel_tickets(event, num_tickets)
                    print("Tickets cancelled successfully!")
                else:
                    print("Invalid event index")

            elif choice == '4':
                print("Thank you for using the Ticket Booking System!")
                break

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


# Example usage
if __name__ == "__main__":
    ticket_system = TicketBookingSystem()
    ticket_system.main()
```



```
1. Book Tickets
2. View Event Details
3. Cancel Tickets
4. Exit
Enter your choice: 3
Enter the index of the event for ticket cancellation: 4
Invalid event index

1. Book Tickets
2. View Event Details
3. Cancel Tickets
4. Exit
Enter your choice: 4
Thank you for using the Ticket Booking System!
```

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:

```
from abc import ABC, abstractmethod

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

    def __str__(self):
        return (f"Event Name: {self.event_name}\n"
                f"Date: {self.event_date}\n"
                f"Time: {self.event_time}\n"
                f"Venue: {self.venue_name}\n"
                f"Total Seats: {self.total_seats}\n"
                f"Available Seats: {self.available_seats}\n"
                f"Ticket Price: {self.ticket_price}\n"
                f"Event Type: {self.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):
```

```python
        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

    @abstractmethod
    def calculate_total_revenue(self):
        pass

    @abstractmethod
    def get_booked_no_of_tickets(self):
        pass

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, num_tickets):
        pass

    @abstractmethod
    def display_event_details(self):
        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.

```python
from enum import Enum

class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price):
        super().__init__(event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price, "Movie")
        self.booked_tickets = 0

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

    def get_booked_no_of_tickets(self):
        return self.booked_tickets

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

    def display_event_details(self):
        print(self)


class Concert(Event):
    class ConcertType(Enum):
        Theatrical = "Theatrical"
        Classical = "Classical"
        Rock = "Rock"
        Recital = "Recital"

    def __init__(self, event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price, artist=None, concert_type=None):
        super().__init__(event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price, "Concert")
        self.artist = artist
        self.concert_type = concert_type
        self.booked_tickets = 0

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

    def get_booked_no_of_tickets(self):
        return self.booked_tickets

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

    def cancel_booking(self, num_tickets):
        self.available_seats += num_tickets
        self.booked_tickets -= num_tickets

    def display_event_details(self):
        print(self)


class Sports(Event):
    def __init__(self, event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price, sport_name=None, teams_name=None):
        super().__init__(event_name, event_date, event_time, venue_name, total_seats,
available_seats, ticket_price, "Sports")
        self.sport_name = sport_name
        self.teams_name = teams_name
        self.booked_tickets = 0
    def calculate_total_revenue(self):
        return self.ticket_price * (self.total_seats - self.available_seats)

    def get_booked_no_of_tickets(self):
        return self.booked_tickets

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

    def cancel_booking(self, num_tickets):
        self.available_seats += num_tickets
        self.booked_tickets -= num_tickets

    def display_event_details(self):
        print(self)
```

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

```python
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 calculate_booking_cost(self, num_tickets):
        pass

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

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

    @abstractmethod
    def get_available_no_of_tickets(self):
        pass

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

4.Create a concrete class TicketBookingSystem that inherits from BookingSystem: • TicketBookingSystem: Implement the abstract methods to create events, book t ickets, 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 __init__(self):
        self.events = []

    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, 
venue_name):
        event = Event(event_name, date, time, total_seats, total_seats, ticket_price, 
event_type, venue_name)
        self.events.append(event)
        return event

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

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

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

    def cancel_booking(self, event, num_tickets):
        event.set_available_seats(event.get_available_seats() + num_tickets)

    def get_available_no_of_tickets(self):
        return self.events[0].get_available_seats()  # Assuming only one event for 
simplicity

    def get_event_details(self):
        if self.events:
            return self.events[0]
        else:
            print("No events available.")
            return None

    def main(self):
        while True:
            print("\n1. Create Event")
            print("2. Book Tickets")
            print("3. Cancel Tickets")
            print("4. Get Available Seats")
            print("5. Exit")

            choice = input("Enter your choice: ")
```

```python
            if choice == '1':
                event_name = input("Enter event name: ")
                date = input("Enter date (YYYY-MM-DD): ")
                time = input("Enter 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.create_event(event_name, date, time, total_seats, ticket_price,
event_type, venue_name)
                print("Event created successfully!")

            elif choice == '2':
                event = self.get_event_details()
                if event:
                    num_tickets = int(input("Enter number of tickets to book: "))
                    if self.book_tickets(event, num_tickets):
                        print(f"{num_tickets} tickets booked successfully!")
                    else:
                        print("Not enough available seats.")
                else:
                    print("No events available.")

            elif choice == '3':
                event = self.get_event_details()
                if event:
                    num_tickets = int(input("Enter number of tickets to cancel: "))
                    self.cancel_booking(event, num_tickets)
                    print("Tickets cancelled successfully!")
                else:
                    print("No events available.")

            elif choice == '4':
                available_seats = self.get_available_no_of_tickets()
                if available_seats is not None:
                    print(f"Available seats: {available_seats}")
                else:
                    print("No events available.")

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

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


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

task 7:

1.Create a Following classes with the following attributes and methods: 1. Venue Class • Attributes: o venue_name, o address • Methods and Constuctors: 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 display_venue_details(self):
        print("Venue Name:", self.venue_name)
        print("Address:", self.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
```

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 Constuctors: 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.

```
class Event:
    def __init__(self, event_name, event_date, event_time, venue, total_seats,
available_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 = available_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    def __str__(self):
        return (f"Event Name: {self.event_name}\n"
                f"Date: {self.event_date}\n"
                f"Time: {self.event_time}\n"
```

```python
                f"Venue: {self.venue.get_venue_name()}\n"
                f"Total Seats: {self.total_seats}\n"
                f"Available Seats: {self.available_seats}\n"
                f"Ticket Price: {self.ticket_price}\n"
                f"Event Type: {self.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(self):
        return self.venue

    def set_venue(self, venue):
        self.venue = venue

    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 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 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

    def display_event_details(self):
        print(self)
```

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.

4. Customer Class

• Attributes:

o customer_name,

o email,

o phone_number,

• Methods and Constuctors:

o Implement default constructors and overload the constructor with Customer

attributes, generate getter and setter methods.

o display_customer_details(): Display customer details.

```
class Event:
    def __init__(self, event_name, event_date, event_time, venue, total_seats,
available_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 = available_seats
        self.ticket_price = ticket_price
```

```python
        self.event_type = event_type

    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 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

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


# Movie subclass
class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price, movie_type):
        super().__init__(event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price, "Movie")
        self.movie_type = movie_type


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


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


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

    # 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 Name:", self.customer_name)
        print("Email:", self.email)
        print("Phone Number:", self.phone_number)

venue = "Cinema Hall"
date = "2024-05-10"
time = "18:00:00"
total_seats = 100
available_seats = 100
ticket_price = 10.00

movie_event = Movie("Movie Night", date, time, venue, total_seats, available_seats,
ticket_price, "Comedy")
concert_event = Concert("Rock Concert", date, time, venue, total_seats, available_seats,
ticket_price, "Linkin Park", "Rock")
sport_event = Sport("Football Match", date, time, venue, total_seats, available_seats,
ticket_price, "Football", "Barcelona vs Real Madrid")
```

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

```
customer = Customer("John Doe", "john@example.com", "1234567890")
```

```python
print("Movie Event Details:")
movie_event.display_event_details()

print("\nConcert Event Details:")
concert_event.display_event_details()

print("\nSport Event Details:")
sport_event.display_event_details()

print("\nCustomer Details:")
customer.display_customer_details()
```

5. Create a class Booking with the following attributes: • bookingId (should be incremented for each booking) • array of customer (reference to the customer who made the booking) • event (reference to the event booked) • num_tickets(no of tickets and array of customer must equal) • total_cost • booking_date (timestamp of when the booking was made) • Methods and Constuctors: 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_counter = 0

    def __init__(self, customers, event, num_tickets, total_cost, booking_date=None):
        self.booking_id = Booking.booking_counter + 1
        self.customers = customers
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date if booking_date else datetime.now()

        Booking.booking_counter += 1

    def display_booking_details(self):
        print("Booking ID:", self.booking_id)
        print("Booking Date:", self.booking_date)
        print("Event Details:")
        self.event.display_event_details()
        print("Total Cost:", self.total_cost)
        print("Number of Tickets:", self.num_tickets)
        print("Customers:")
        for customer in self.customers:
            customer.display_customer_details()
        print("\n")

    # Getter and setter methods
    def get_booking_id(self):
        return self.booking_id

    def set_booking_id(self, booking_id):
        self.booking_id = booking_id

    def get_customers(self):
        return self.customers

    def set_customers(self, customers):
        self.customers = customers

    def get_event(self):
        return self.event

    def set_event(self, event):
        self.event = event

    def get_num_tickets(self):
        return self.num_tickets

    def set_num_tickets(self, num_tickets):
        self.num_tickets = num_tickets

    def get_total_cost(self):
        return self.total_cost

    def set_total_cost(self, total_cost):
        self.total_cost = total_cost

    def get_booking_date(self):
        return self.booking_date

    def set_booking_date(self, booking_date):
        self.booking_date = booking_date
```

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 Constuctors: o create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: f loat, 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."

```
from datetime import datetime

class Booking:
    booking_counter = 0

    def __init__(self, customers, event, num_tickets, total_cost, booking_date=None):
        self.booking_id = Booking.booking_counter + 1
        self.customers = customers
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date if booking_date else datetime.now()

        Booking.booking_counter += 1

    def display_booking_details(self):
        print("Booking ID:", self.booking_id)
        print("Booking Date:", self.booking_date)
        print("Event Details:")
        self.event.display_event_details()
        print("Total Cost:", self.total_cost)
        print("Number of Tickets:", self.num_tickets)
        print("Customers:")
        for customer in self.customers:
            customer.display_customer_details()
        print("\n")

    # Getter and setter methods
    def get_booking_id(self):
        return self.booking_id

    def set_booking_id(self, booking_id):
        self.booking_id = booking_id

    def get_customers(self):
        return self.customers

    def set_customers(self, customers):
        self.customers = customers

    def get_event(self):
        return self.event

    def set_event(self, event):
        self.event = event

    def get_num_tickets(self):
        return self.num_tickets

    def set_num_tickets(self, num_tickets):
        self.num_tickets = num_tickets

    def get_total_cost(self):
        return self.total_cost

    def set_total_cost(self, total_cost):
        self.total_cost = total_cost

    def get_booking_date(self):
        return self.booking_date

    def set_booking_date(self, booking_date):
        self.booking_date = booking_date

#6
class BookingSystem:
    def __init__(self):
        self.events = []

    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type,
venue):
        event = Event(event_name, date, time, venue, total_seats, total_seats,
ticket_price, event_type)
```

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

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

    def book_tickets(self, event_name, num_tickets, customers):
        for event in self.events:
            if event.get_event_name() == event_name:
                if event.book_tickets(num_tickets):
                    total_cost = self.calculate_booking_cost(num_tickets,
event.get_ticket_price())
                    booking = Booking(customers, event, num_tickets, total_cost)
                    return booking
                else:
                    print("Tickets are not available for booking.")
                    return None
        print("Event not found.")
        return None

    def cancel_booking(self, booking_id):
        for event in self.events:
            for booking in event.get_bookings():
                if booking.get_booking_id() == booking_id:
                    event.cancel_booking(booking.get_num_tickets())
                    event.remove_booking(booking)
                    print("Booking cancelled successfully.")
                    return
        print("Booking not found.")

    def get_available_no_of_tickets(self, event_name):
        for event in self.events:
            if event.get_event_name() == event_name:
                return event.get_available_seats()
        return 0

    def get_event_details(self, event_name):
        for event in self.events:
            if event.get_event_name() == event_name:
                return event
        return None

    def display_menu(self):
        print("\n1. Create Event")
        print("2. Book Tickets")
        print("3. Cancel Booking")
        print("4. Get Available Seats")
        print("5. Get Event Details")
        print("6. Exit")

    def main(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")
            if choice == '1':
                event_name = input("Enter event name: ")
                date = input("Enter date (YYYY-MM-DD): ")
                time = input("Enter time (HH:MM:SS): ")
                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 = Venue(venue_name, venue_address)
                self.create_event(event_name, date, time, total_seats, ticket_price,
event_type, venue)
                print("Event created successfully!")
            elif choice == '2':
                event_name = input("Enter event name: ")
                num_tickets = int(input("Enter number of tickets to book: "))
                customer_name = input("Enter customer name: ")
                email = input("Enter email: ")
                phone_number = input("Enter phone number: ")
                customers = [Customer(customer_name, email, phone_number) for _ in
range(num_tickets)]
                booking = self.book_tickets(event_name, num_tickets, customers)
                if booking:
                    print("Booking successful!")
                    booking.display_booking_details()
            elif choice == '3':
                booking_id = int(input("Enter booking ID to cancel: "))
                self.cancel_booking(booking_id)
            elif choice == '4':
                event_name = input("Enter event name: ")
                available_seats = self.get_available_no_of_tickets(event_name)
                print("Available seats:", available_seats)
            elif choice == '5':
                event_name = input("Enter event name: ")
                event_details = self.get_event_details(event_name)
                if event_details:
                    event_details.display_event_details()
                else:
                    print("Event not found.")
            elif choice == '6':
                print("Exiting...")
                break
            else:
                print("Invalid choice. Please try again.")


# Example usage:
if __name__ == "__main__":
    booking_system = BookingSystem()
    booking_system.main()
```

Task 8: Interface/abstract class, and Single Inheritance, static variable 1. Create Venue, class as mentioned above Task 4.

```
from abc import ABC, abstractmethod

class AbstractVenue(ABC):
```

```python
    @abstractmethod
    def display_venue_details(self):
        pass

    @abstractmethod
    def get_venue_name(self):
        pass

    @abstractmethod
    def set_venue_name(self, venue_name):
        pass

    @abstractmethod
    def get_address(self):
        pass

    @abstractmethod
    def set_address(self, address):
        pass


class AbstractUser(ABC):
    @abstractmethod
    def display_user_details(self):
        pass

    @abstractmethod
    def get_username(self):
        pass

    @abstractmethod
    def set_username(self, username):
        pass

    @abstractmethod
    def get_email(self):
        pass

    @abstractmethod
    def set_email(self, email):
        pass


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

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

    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


class User(AbstractUser):
    def __init__(self, username, email):
        self.username = username
        self.email = email

    def display_user_details(self):
        print("Username:", self.username)
        print("Email:", self.email)

    def get_username(self):
        return self.username

    def set_username(self, username):
        self.username = username

    def get_email(self):
        return self.email

    def set_email(self, email):
        self.email = email


# Example usage:
if __name__ == "__main__":
    venue = Venue("Example Venue", "123 Main Street")
    venue.display_venue_details()

    user = User("example_user", "user@example.com")
    user.display_user_details()
```

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 Constuctors: 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.

```
from abc import ABC, abstractmethod

class AbstractEvent(ABC):
```

```python
    def __init__(self, event_name, event_date, event_time, venue, total_seats,
available_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 = available_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    @abstractmethod
    def display_event_details(self):
        pass

    @abstractmethod
    def calculate_total_revenue(self):
        pass

    @abstractmethod
    def get_booked_no_of_tickets(self):
        pass

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, num_tickets):
        pass


class Event(AbstractEvent):
    def __init__(self, event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price, event_type):
        super().__init__(event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price, event_type)

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

    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 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


# Example usage:
if __name__ == "__main__":
    # Creating a venue
    venue = Venue("Example Venue", "123 Main Street")

    # Creating an event
    event = Event("Concert", "2024-05-10", "18:00:00", venue, 100, 100, 10.00, "Concert")
    event.display_event_details()
```

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. Concert. o Sport. abstract method

```python
class Movie(AbstractEvent):
    def __init__(self, event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price):
        super().__init__(event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price, "Movie")

    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 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

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

```python
class Concert(AbstractEvent):
    def __init__(self, event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price):
        super().__init__(event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price, "Concert")

    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 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

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


class Sport(AbstractEvent):
    def __init__(self, event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price):
        super().__init__(event_name, event_date, event_time, venue, total_seats,
available_seats, ticket_price, "Sport")

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


# Example usage:
if __name__ == "__main__":
    # Creating a venue
    venue = Venue("Example Venue", "123 Main Street")

    # Creating instances of Movie, Concert, and Sport events
    movie_event = Movie("Movie Night", "2024-05-15", "20:00:00", venue, 200, 200, 15.00)
    concert_event = Concert("Rock Concert", "2024-06-20", "18:00:00", venue, 100, 100, 
20.00)
    sport_event = Sport("Football Match", "2024-07-10", "16:00:00", venue, 500, 500, 
25.00)

    # Display event details
    movie_event.display_event_details()
    concert_event.display_event_details()
    sport_event.display_event_details()
```

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

```python
from abc import ABC, abstractmethod

class AbstractCustomer(ABC):
    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

    @abstractmethod
    def display_customer_details(self):
        pass


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

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


# Example usage:
if __name__ == "__main__":
    # Creating a customer
    customer = Customer("John Doe", "john@example.com", "123-456-7890")

    # Display customer details
    customer.display_customer_details()
```

5. 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 abc import ABC, abstractmethod

class AbstractBooking(ABC):
    def __init__(self, booking_id, customers, 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

    # Getter and setter methods
    def get_booking_id(self):
        return self.booking_id

    def set_booking_id(self, booking_id):
        self.booking_id = booking_id
    def get_customers(self):
        return self.customers

    def set_customers(self, customers):
        self.customers = customers

    def get_event(self):
        return self.event

    def set_event(self, event):
        self.event = event

    def get_num_tickets(self):
        return self.num_tickets

    def set_num_tickets(self, num_tickets):
        self.num_tickets = num_tickets

    def get_total_cost(self):
        return self.total_cost

    def set_total_cost(self, total_cost):
        self.total_cost = total_cost

    def get_booking_date(self):
        return self.booking_date

    def set_booking_date(self, booking_date):
        self.booking_date = booking_date

    @abstractmethod
    def display_booking_details(self):
        pass


class Booking(AbstractBooking):
    def __init__(self, booking_id, customers, event, num_tickets, total_cost,
booking_date):
        super().__init__(booking_id, customers, event, num_tickets, total_cost,
booking_date)

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


# Example usage:
if __name__ == "__main__":
    # Creating a booking
    booking = Booking(1, ["John Doe"], "Concert", 2, 40.00, "2024-05-10")

    # Display booking details
    booking.display_booking_details()
```
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 t ickets 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 AbstractBookingSystem(ABC):
    def __init__(self):
        self.events = []

    @abstractmethod
    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, 
venue):
        pass

    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        pass

    @abstractmethod
    def book_tickets(self, eventname, num_tickets, array_of_customer):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

    @abstractmethod
    def get_available_no_of_tickets(self):
        pass

    @abstractmethod
    def get_event_details(self):
        pass

    def main(self):
        while True:
            print("\nWelcome to the Ticket Booking System")
            print("1. Create Event")
            print("2. Book Tickets")
            print("3. Cancel Booking")
            print("4. Get Available Seats")
            print("5. Get Event Details")
            print("6. Exit")

            choice = input("Enter your choice: ")

            if choice == "1":
                event_name = input("Enter event name: ")
                date = input("Enter date: ")
                time = input("Enter time: ")
```

```python
                total_seats = int(input("Enter total seats: "))
                ticket_price = float(input("Enter ticket price: "))
                event_type = input("Enter event type (movie, sport, concert): ")
                venue = input("Enter venue: ")
                event = self.create_event(event_name, date, time, total_seats,
ticket_price, event_type, venue)
                self.events.append(event)
                print("Event created successfully!")

            elif choice == "2":
                event_name = input("Enter event name: ")
                num_tickets = int(input("Enter number of tickets: "))
                array_of_customer = input("Enter names of customers separated by commas:
").split(",")
                success = self.book_tickets(event_name, num_tickets, array_of_customer)
                if success:
                    print("Tickets booked successfully!")
                else:
                    print("Failed to book tickets. Not enough available seats.")

            elif choice == "3":
                booking_id = int(input("Enter booking ID to cancel: "))
                self.cancel_booking(booking_id)
                print("Booking canceled successfully!")

            elif choice == "4":
                print("Total available seats:", self.get_available_no_of_tickets())

            elif choice == "5":
                print("Event Details:")
                for event in self.events:
                    event.display_event_details()

            elif choice == "6":
                print("Exiting program...")
                break

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


# Example usage:
if __name__ == "__main__":
    class BookingSystem(AbstractBookingSystem):
        def create_event(self, event_name, date, time, total_seats, ticket_price,
event_type, venue):
            # Here, you should create an Event object using the provided information and
return it
            pass

        def calculate_booking_cost(self, num_tickets):
            # Calculate total cost of booking
            pass

        def book_tickets(self, eventname, num_tickets, array_of_customer):
            # Book tickets for the specified event
            pass

        def cancel_booking(self, booking_id):
            # Cancel the booking with the given ID
            pass

        def get_available_no_of_tickets(self):
            # Get the total available tickets
            pass

        def get_event_details(self):
            # Get details of all events
            pass

    booking_system = BookingSystem()
    booking_system.main()
```

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

```
from abc import ABC, abstractmethod

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

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

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

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

    @abstractmethod
    def get_available_no_of_tickets(self, event):
        pass

    @abstractmethod
    def get_event_details(self):
        pass


class EventServiceProviderImpl(IEventServiceProvider):
    def __init__(self):
        self.events = []
    def create_event(self, event_name, date, time, venue, total_seats, ticket_price,
event_type):
        event = Event(event_name, date, time, venue, total_seats, total_seats,
ticket_price, event_type)
        self.events.append(event)
        return event

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

    def book_tickets(self, event, num_tickets, customers):
        if event.get_available_seats() >= num_tickets:
            event.set_available_seats(event.get_available_seats() - num_tickets)
            print(f"{num_tickets} tickets booked successfully for
{event.get_event_name()}")
            return True
        else:
            print("Failed to book tickets. Not enough available seats.")
            return False

    def cancel_booking(self, event, booking_id):
        # Implement cancellation logic here
        pass

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

    def get_event_details(self):
        for event in self.events:
            event.display_event_details()


# Example usage:
if __name__ == "__main__":
    event_provider = EventServiceProviderImpl()

    # Create an event
    venue = Venue("Example Venue", "123 Main Street")
    event = event_provider.create_event("Movie Night", "2024-05-15", "20:00:00", venue,
200, 15.00, "Movie")

    # Book tickets for the event
    event_provider.book_tickets(event, 5, ["John", "Jane", "Doe", "Smith", "Alice"])

    # Display event details
    event_provider.get_event_details()
```

8.Create BookingSystemServiceProviderImpl class which implements IBookingSystemServiceProvider provide all implementation methods and inherits EventServiceProviderImpl class with following attributes. • Attributes o array of events

```
from abc import ABC, abstractmethod

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


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

    def create_event(self, event_name, date, time, venue, total_seats, ticket_price,
event_type):

        event = Event(event_name, date, time, venue, total_seats, ticket_price,
event_type)
        self.events.append(event)
        return event




class IBookingSystemServiceProvider(ABC):
    @abstractmethod
    def book_tickets(self, event, num_tickets, customers):
        pass




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

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

        pass



if __name__ == "__main__":

    booking_system_provider = BookingSystemServiceProviderImpl()


    event = booking_system_provider.create_event("Movie Night", "2022-05-15", "20:00:00",
"Example Venue", 200, 10.00, "Movie")
```

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):
```

```python
        self.event_provider = EventServiceProviderImpl()

    def run(self):
        while True:
            print("\n*** Ticket Booking System ***")
            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")

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

    def create_event(self):
        event_name = input("Enter event name: ")
        date = input("Enter event date (YYYY-MM-DD): ")
        time = input("Enter event time (HH:MM:SS): ")
        venue_name = input("Enter venue name: ")
        venue_address = input("Enter venue address: ")
        total_seats = int(input("Enter total seats: "))
        ticket_price = float(input("Enter ticket price: "))
        event_type = input("Enter event type (Movie/Sports/Concert): ")

        venue = Venue(venue_name, venue_address)
        event = self.event_provider.create_event(event_name, date, time, venue,
total_seats, ticket_price, event_type)
        print("Event created successfully!")

    def book_tickets(self):
        event_name = input("Enter event name: ")
        num_tickets = int(input("Enter number of tickets to book: "))

        # Get the event object based on its name
        event = None
        for e in self.event_provider.events:
            if e.get_event_name() == event_name:
                event = e
                break
        if event:
            # Book tickets for the event
            # Implement the book_tickets method in EventServiceProviderImpl and use it
here
            pass
        else:
            print("Event not found!")

    def cancel_tickets(self):
        # Implement cancel_tickets method
        pass

    def get_available_seats(self):
        # Implement get_available_seats method
        pass

    def get_event_details(self):
        # Implement get_event_details method
        pass


# Example usage
if __name__ == "__main__":
    ticket_booking_system = TicketBookingSystem()
    ticket_booking_system.run()
```

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

```
class TicketBookingSystem:
    def __init__(self):
        self.event_provider = EventServiceProviderImpl()

    def run(self):
        while True:
            print("\n*** Ticket Booking System ***")
            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")

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

    def create_event(self):
        event_name = input("Enter event name: ")
        date = input("Enter event date (YYYY-MM-DD): ")
        time = input("Enter event time (HH:MM:SS): ")
        venue_name = input("Enter venue name: ")
        venue_address = input("Enter venue address: ")
        total_seats = int(input("Enter total seats: "))
        ticket_price = float(input("Enter ticket price: "))
        event_type = input("Enter event type (Movie/Sports/Concert): ")

        venue = Venue(venue_name, venue_address)
        event = self.event_provider.create_event(event_name, date, time, venue,
total_seats, ticket_price, event_type)
        print("Event created successfully!")

    def book_tickets(self):
        event_name = input("Enter event name: ")
        num_tickets = int(input("Enter number of tickets to book: "))

        event = self.find_event_by_name(event_name)
        if event:
            # Book tickets for the event
            # Implement the book_tickets method in EventServiceProviderImpl and use it
here
            pass
        else:
            print("Event not found!")

    def find_event_by_name(self, event_name):
        for event in self.event_provider.events:
            if event.get_event_name() == event_name:
                return event
        return None

    # Implement cancel_tickets, get_available_seats, and get_event_details methods
similarly


# Example usage
if __name__ == "__main__":
    ticket_booking_system = TicketBookingSystem()
    ticket_booking_system.run()
```

Task 9: Exception Handling throw the exception whenever needed and Handle in main method,

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.

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

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

class NullPointerException(Exception):
    def __init__(self, message):
        super().__init__(f"Null pointer exception: {message}")


class TicketBookingSystem:
    def __init__(self):
        self.event_provider = EventServiceProviderImpl()

    def run(self):
        while True:
            print("\n*** Ticket Booking System ***")
            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")

            choice = input("Enter your choice: ")

            try:
                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...")
                    break
                else:
                    print("Invalid choice! Please try again.")
            except EventNotFoundException as e:
                print(f"Error: {e}")
            except InvalidBookingIDException as e:
                print(f"Error: {e}")
            except NullPointerException as e:
                print(f"Error: {e}")
    def create_event(self):
        event_name = input("Enter event name: ")
        date = input("Enter event date (YYYY-MM-DD): ")
        time = input("Enter event time (HH:MM:SS): ")
        venue_name = input("Enter venue name: ")
        venue_address = input("Enter venue address: ")
        total_seats = int(input("Enter total seats: "))
        ticket_price = float(input("Enter ticket price: "))
        event_type = input("Enter event type (Movie/Sports/Concert): ")

        venue = Venue(venue_name, venue_address)
        event = self.event_provider.create_event(event_name, date, time, venue,
total_seats, ticket_price, event_type)
        print("Event created successfully!")

    def book_tickets(self):
        event_name = input("Enter event name: ")
        num_tickets = int(input("Enter number of tickets to book: "))

        event = self.find_event_by_name(event_name)
        if event:

            pass
        else:
            raise EventNotFoundException(event_name)

    def find_event_by_name(self, event_name):
        for event in self.event_provider.events:
            if event.get_event_name() == event_name:
                return event
        return None

    # Implement cancel_tickets, get_available_seats, and get_event_details methods
similarly


# Example usage
if __name__ == "__main__":
    ticket_booking_system = TicketBookingSystem()
    ticket_booking_system.run()
```

Task 10.Collection

From the previous task change the Booking class attribute customers to List of customers and BookingSystem class attribute events to List of events and perform the same operation. 2. From the previous task change all list type of attribute to type Set in Booking and Booking system

```
from typing import List

class Booking:
    booking_id = 0

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

    def display_booking_details(self):
        print("Booking ID:", self.booking_id)
        print("Event:", self.event.get_event_name())
        print("Number of Tickets:", self.num_tickets)
        print("Total Cost:", self.total_cost)
        print("Booking Date:", self.booking_date)
        print("Customers:")
        for customer in self.customers:
            customer.display_customer_details()


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

    def create_event(self, event_name: str, date: str, time: str, total_seats: int,
ticket_price: float, event_type: str, venue: Venue):
        event = Event(event_name, date, time, venue, total_seats, total_seats,
ticket_price, event_type)
        self.events.append(event)
        return event

    def calculate_booking_cost(self, num_tickets: int):
        pass  # Add implementation

    def book_tickets(self, event_name: str, num_tickets: int, customers: List[Customer]):
        event = self.find_event_by_name(event_name)
        if event:
            # Book tickets for the event
            # Add implementation
            pass
        else:
            raise EventNotFoundException(event_name)

    def cancel_booking(self, booking_id: int):
        pass  # Add implementation

    def get_available_no_of_tickets(self):
        pass  # Add implementation

    def get_event_details(self):
        pass  # Add implementation

    def find_event_by_name(self, event_name: str):
        for event in self.events:
            if event.get_event_name() == event_name:
                return event
        return None
```

```

# Example usage
if __name__ == "__main__":
    booking_system = BookingSystem()
    booking_system.create_event("Concert", "2024-05-15", "19:00", 1000, 50.0, "Concert",
Venue("City Hall", "Main Street"))
    booking_system.create_event("Movie", "2024-06-01", "15:00", 500, 20.0, "Movie",
Venue("Cinema", "Broadway"))

    booking_system.book_tickets("Concert", 2, [Customer("Alice", "alice@example.com",
"1234567890")])
    booking_system.book_tickets("Movie", 3, [Customer("Bob", "bob@example.com",
"9876543210")])

    for event in booking_system.events:
        print("Event:", event.get_event_name())
        print("Available Seats:", event.get_available_seats())
```

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 object to sort the event based on event name and location in alphabetical order.

```
from typing import Set
from functools import cmp_to_key

class Booking:
    booking_id = 0

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

    def display_booking_details(self):
        print("Booking ID:", self.booking_id)
        print("Event:", self.event.get_event_name())
        print("Number of Tickets:", self.num_tickets)
        print("Total Cost:", self.total_cost)
        print("Booking Date:", self.booking_date)
        print("Customers:")
        for customer in self.customers:
            customer.display_customer_details()
```

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:
    booking_id = 0
    def __init__(self, customers: Dict[int, Customer], event: Event, num_tickets: int, 
total_cost: float, booking_date: str):
        self.booking_id = Booking.booking_id
        Booking.booking_id += 1
        self.customers = customers
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date

    def display_booking_details(self):
        print("Booking ID:", self.booking_id)
        print("Event:", self.event.get_event_name())
        print("Number of Tickets:", self.num_tickets)
        print("Total Cost:", self.total_cost)
        print("Booking Date:", self.booking_date)
        print("Customers:")
        for customer_id, customer in self.customers.items():
            print("Customer ID:", customer_id)
            customer.display_customer_details()
```

Task 11: Database Connectivity

Create Venue, Event, Customer and Booking class as mentioned above Task 5.

```
from datetime import datetime

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

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

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

```
class Booking:
    def __init__(self, booking_id, customer_id, event_id, num_tickets, total_cost,
booking_date):
        self.booking_id = booking_id
        self.customer_id = customer_id
        self.event_id = event_id
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date
```

2.Create Event sub classes as mentioned in above Task 4.

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

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

class Concert(Event):
    def __init__(self, event_id, event_name, event_date, event_time, venue_id,
total_seats, available_seats, ticket_price, event_type, artist, concert_type):
        super().__init__(event_id, event_name, event_date, event_time, venue_id,
total_seats, available_seats, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type

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

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

    def display_event_details(self):
        print(f"Sports Event: {self.event_name}, Sport: {self.sport_name}, Teams:
{self.teams_name}")
```

3. Create interface/abstract class IEventServiceProvider, IBookingSystemServiceProvider and its implementation classes as mentioned in above Task 5.

```python
from abc import ABC, abstractmethod

class IEventServiceProvider(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

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

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

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

    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type,
venue_name):
        # Assuming venue_id is known
        venue_id = 1  # Example venue_id
        event_id = len(self.events) + 1  # Generating event_id
        available_seats = total_seats
        event_date = datetime.strptime(date, '%Y-%m-%d').date()
        event_time = datetime.strptime(time, '%H:%M').time()

        new_event = None
        if event_type == 'Movie':
            new_event = Movie(event_id, event_name, event_date, event_time, venue_id,
total_seats, available_seats, ticket_price, event_type, genre="", actor_name="",
actress_name="")
        elif event_type == 'Concert':
            new_event = Concert(event_id, event_name, event_date, event_time, venue_id,
total_seats, available_seats, ticket_price, event_type, artist="", concert_type="")
        elif event_type == 'Sports':
            new_event = Sports(event_id, event_name, event_date, event_time, venue_id,
total_seats, available_seats, ticket_price, event_type, sport_name="", teams_name="")

        self.events.append(new_event)
        return new_event

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

    def book_tickets(self, event, num_tickets):
        if event.available_seats >= num_tickets:
            event.available_seats -= num_tickets
            total_cost = num_tickets * event.ticket_price
            return total_cost
        else:
            print("Event is sold out.")

    def cancel_tickets(self, event, num_tickets):
        event.available_seats += num_tickets
```

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 t ickets 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.

```
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: Venue):
        # Dummy implementation, replace with actual database interaction
        event_id = generate_event_id()  # Example function to generate event_id
        new_event = Event(event_id, event_name, date, time, venue.venue_id, total_seats,
total_seats, ticket_price, event_type)
        store_event_in_database(new_event)  # Example function to store event in database
        return new_event

    @abstractmethod
    def get_event_details(self):
        # Dummy implementation, replace with actual database interaction
        return retrieve_all_events_from_database()  # Example function to retrieve all
events

    @abstractmethod
    def get_available_no_of_tickets(self):
        # Dummy implementation, replace with actual database interaction
        return retrieve_available_tickets_from_database()  # Example function to retrieve
available tickets

    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        # Dummy implementation, replace with actual calculation logic
        return num_tickets * TICKET_PRICE  # Example calculation for total cost
    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, list_of_customers):

        event_id = retrieve_event_id_by_name(event_name)
        booking_id = generate_booking_id()
        total_cost = self.calculate_booking_cost(num_tickets)
        for customer in list_of_customers:
            new_booking = Booking(booking_id, customer.customer_id, event_id, num_tickets,
total_cost, datetime.now())
            store_booking_in_database(new_booking)
        return booking_id

    @abstractmethod
    def cancel_booking(self, booking_id):

        cancel_booking_in_database(booking_id)
    @abstractmethod
    def get_booking_details(self, booking_id):

        return retrieve_booking_details_from_database(booking_id)
```

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

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

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: Venue):
        pass

    @abstractmethod
    def get_event_details(self):
        pass

    @abstractmethod
    def get_available_no_of_tickets(self):
        pass

    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        pass

    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, list_of_customers):
        pass
    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

    @abstractmethod
    def get_booking_details(self, booking_id):
        pass

class BookingSystemRepositoryImpl(IBookingSystemRepository):
    def __init__(self, database):
        self.database = database  # Assuming database is an instance of a database
connection or ORM

    def create_event(self, event_name: str, date: str, time: str, total_seats: int,
ticket_price: float, event_type: str, venue: Venue):
        event_id = self.generate_event_id()  # Example function to generate event_id
        event_date = datetime.strptime(date, '%Y-%m-%d')
        event_time = datetime.strptime(time, '%H:%M')
        available_seats = total_seats
        event = Event(event_id, event_name, event_date, event_time, venue.venue_id,
total_seats, available_seats, ticket_price, event_type)
        self.database.store_event(event)  # Example function to store event in database
        return event

    def get_event_details(self):
        return self.database.retrieve_all_events()  # Example function to retrieve all
events

    def get_available_no_of_tickets(self):
        return self.database.retrieve_available_tickets()
    def calculate_booking_cost(self, num_tickets):
        return num_tickets * ticket_price

    def book_tickets(self, event_name: str, num_tickets: int, list_of_customers:
List[Customer]):
        event_id = self.database.retrieve_event_id_by_name(event_name)
        booking_id = self.generate_booking_id()
        total_cost = self.calculate_booking_cost(num_tickets)
        for customer in list_of_customers:
            booking = Booking(booking_id, customer.customer_id, event_id, num_tickets,
total_cost, datetime.now())
            self.database.store_booking(booking)
        return booking_id

    def cancel_booking(self, booking_id):
        self.database.cancel_booking(booking_id)
    def get_booking_details(self, booking_id):
        return self.database.retrieve_booking_details(booking_id)
    def generate_event_id(self):

        pass

    def generate_booking_id(self):
        pass
```

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

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="diana",
    database="ordermanagement"
)

cursor = conn.cursor()

try:
    cursor.execute("select * from venu")
    rows = cursor.fetchall()
    for row in rows:
        print(row)
except mysql.connector.Error as e:
    print("Error:", e)
finally:
    cursor.close()
    conn.close()
```

7. Place the interface/abstract class in service package and interface implementation class, concrete class in bean package and TicketBookingSystemRepository class in app package.

```
from bean.BookingSystemRepositoryImpl import BookingSystemRepositoryImpl

from service.IEventServiceProvider import IEventServiceProvider
from service.IBookingSystemServiceProvider import IBookingSystemServiceProvider
from service.IBookingSystemRepository import IBookingSystemRepository
from datetime import datetime
from typing import List
from abc import ABC, abstractmethod
from typing import List

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



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


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: Venue):
        pass

    @abstractmethod
    def get_event_details(self):
        pass

    @abstractmethod
    def get_available_no_of_tickets(self):
        pass

    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        pass

    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, list_of_customers):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

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

8. Should throw appropriate exception as mentioned in above task along with handle SQLException.

```
from service.IBookingSystemRepository import IBookingSystemRepository
from datetime import datetime
from typing import List
import sqlite3

class BookingSystemRepositoryImpl(IBookingSystemRepository):
    def __init__(self, database):
        self.database = database  # Assuming database is an instance of a database
connection or ORM

    def create_event(self, event_name: str, date: str, time: str, total_seats: int,
ticket_price: float, event_type: str, venue: Venue):
        try:
            event_id = self.generate_event_id()  # Example function to generate event_id
            event_date = datetime.strptime(date, '%Y-%m-%d')
            event_time = datetime.strptime(time, '%H:%M')
            available_seats = total_seats
            event = Event(event_id, event_name, event_date, event_time, venue.venue_id,
total_seats, available_seats, ticket_price, event_type)
            self.database.store_event(event)  # Example function to store event in
database
            return event
        except Exception as e:
            raise EventCreationException("Failed to create event.") from e

    def get_event_details(self):
        try:
            return self.database.retrieve_all_events()
```

9. Create TicketBookingSystem class and perform following operations: • Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats,", "get_event_details," and "exit."

```
from bean.TicketBookingSystem import TicketBookingSystem

class TicketBookingSystemUI:
    @staticmethod
    def main():
        ticket_booking_system = TicketBookingSystem()

        while True:
            print("\nWelcome to the Ticket Booking System")
            print("Available commands:")
            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")

            command = input("Enter command: ")

            if command == "create_event":
                TicketBookingSystemUI.create_event(ticket_booking_system)
            elif command == "book_tickets":
                TicketBookingSystemUI.book_tickets(ticket_booking_system)
            elif command == "cancel_tickets":
                TicketBookingSystemUI.cancel_tickets(ticket_booking_system)
            elif command == "get_available_seats":
                TicketBookingSystemUI.get_available_seats(ticket_booking_system)
            elif command == "get_event_details":
                TicketBookingSystemUI.get_event_details(ticket_booking_system)
            elif command == "exit":
                print("Exiting...")
                break
            else:
                print("Invalid command. Please enter a valid command.")

    @staticmethod
    def create_event(ticket_booking_system):
        # Placeholder for create_event functionality
        pass

    @staticmethod
    def book_tickets(ticket_booking_system):
        # Placeholder for book_tickets functionality
        pass

    @staticmethod
    def cancel_tickets(ticket_booking_system):
        # Placeholder for cancel_tickets functionality
```

```python
        pass

    @staticmethod
    def get_available_seats(ticket_booking_system):
        # Placeholder for get_available_seats functionality
        pass

    @staticmethod
    def get_event_details(ticket_booking_system):
        # Placeholder for get_event_details functionality
        pass

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

```
Welcome to the Ticket Booking System
Available commands:
1. create_event
2. book_tickets
3. cancel_tickets
4. get_available_seats
5. get_event_details
6. exit
Enter command: create_event
[Placeholder] Enter event details:
Event created successfully.

Welcome to the Ticket Booking System
Available commands:
1. create_event
2. book_tickets
3. cancel_tickets
4. get_available_seats
5. get_event_details
6. exit
Enter command: book_tickets
[Placeholder] Enter event name and number of tickets:
Booking successful! Total cost: $100

Welcome to the Ticket Booking System
Available commands:
1. create_event
2. book_tickets
3. cancel_tickets
4. get_available_seats
5. get_event_details
6. exit
Enter command: cancel_tickets
[Placeholder] Enter booking ID and number of tickets to cancel:
Tickets canceled successfully.
```