**Assignment 5**

**Control Structure**

**SaiPrabath Chowdary S**

**Task 1: Conditional Statements**

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

**Tasks:**

1. Write a program that takes the availableTicket and noOfBookingTicket as input.
2. Use conditional statements (if-else) to determine if the ticket is available or not.

3. Display an appropriate message based on ticket availability.

def check\_ticket\_availability(available\_tickets, no\_of\_tickets\_to\_book):

if available\_tickets >= no\_of\_tickets\_to\_book:

remaining\_tickets = available\_tickets - no\_of\_tickets\_to\_book

print(f"Tickets available! Remaining tickets: {remaining\_tickets}")

else:

print("Tickets unavailable!")

# Example 1

available\_tickets = 50

no\_of\_tickets\_to\_book = 3

check\_ticket\_availability(available\_tickets, no\_of\_tickets\_to\_book)

Screenshot 2024-04-25 at 3.35.14 PM

# Example 2

available\_tickets = 5

no\_of\_tickets\_to\_book = 7

check\_ticket\_availability(available\_tickets, no\_of\_tickets\_to\_book)

Screenshot 2024-04-25 at 3.39.02 PM

**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.

def calculate\_ticket\_cost(ticket\_type, num\_tickets):

base\_prices = {"Silver": 100, "Gold": 200, "Diamond": 300}

if ticket\_type in base\_prices:

base\_price = base\_prices[ticket\_type]

total\_cost = base\_price \* num\_tickets

return total\_cost

else:

return None

ticket\_type = input("Enter ticket type (Silver/Gold/Diamond): ")

num\_tickets = int(input("Enter number of tickets: "))

total\_cost = calculate\_ticket\_cost(ticket\_type, num\_tickets)

if total\_cost is not None:

print(f"Total cost for {num\_tickets} {ticket\_type} tickets: {total\_cost}")

else:

print("Invalid ticket type!")



**Task 3: Looping**

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

def calculate\_ticket\_cost(ticket\_type, num\_tickets):

base\_prices = {"Silver": 100, "Gold": 200, "Diamond": 300}

if ticket\_type in base\_prices:

base\_price = base\_prices[ticket\_type]

total\_cost = base\_price \* num\_tickets

return total\_cost

else:

return None

while True:

ticket\_type = input("Enter ticket type (Silver/Gold/Diamond), or type 'Exit' to quit: ")

if ticket\_type.lower() == "exit":

print("Exiting ticket booking system.")

break

num\_tickets = int(input("Enter number of tickets: "))

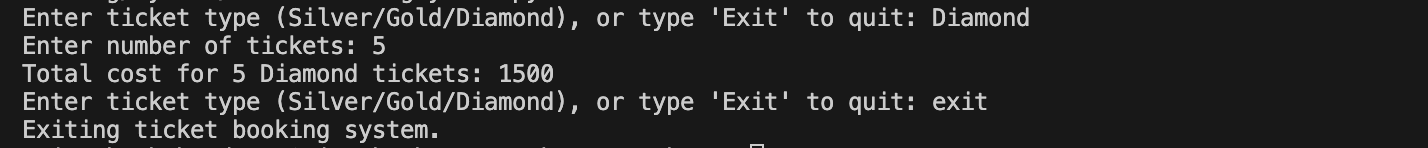
total\_cost = calculate\_ticket\_cost(ticket\_type, num\_tickets)

if total\_cost is not None:

print(f"Total cost for {num\_tickets} {ticket\_type} tickets: {total\_cost}")

else:

print("Invalid ticket type!")



**Task 4: Class & Object**

**Create a Following classes with the following attributes and methods:**

1. **Event** Class**:**

• **Attributes:**

o event\_name,

o event\_date DATE,

o event\_time TIME,

o venue\_name,

o total\_seats,

o available\_seats,

o ticket\_price DECIMAL,

o event\_type ENUM('Movie', 'Sports', 'Concert')

• **Methods and 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.

class Event:

def \_\_init\_\_(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type):

self.event\_name = event\_name

self.event\_date = event\_date

self.event\_time = event\_time

self.venue\_name = venue\_name

self.total\_seats = total\_seats

self.available\_seats = total\_seats

self.ticket\_price = ticket\_price

self.event\_type = event\_type

# Getter and Setter methods

def get\_event\_name(self):

return self.event\_name

def set\_event\_name(self, event\_name):

self.event\_name = event\_name

def get\_event\_date(self):

return self.event\_date

def set\_event\_date(self, event\_date):

self.event\_date = event\_date

def get\_event\_time(self):

return self.event\_time

def set\_event\_time(self, event\_time):

self.event\_time = event\_time

def get\_venue\_name(self):

return self.venue\_name

def set\_venue\_name(self, venue\_name):

self.venue\_name = venue\_name

def get\_total\_seats(self):

return self.total\_seats

def set\_total\_seats(self, total\_seats):

self.total\_seats = total\_seats

def get\_available\_seats(self):

return self.available\_seats

def set\_available\_seats(self, available\_seats):

self.available\_seats = available\_seats

def get\_ticket\_price(self):

return self.ticket\_price

def set\_ticket\_price(self, ticket\_price):

self.ticket\_price = ticket\_price

def get\_event\_type(self):

return self.event\_type

def set\_event\_type(self, event\_type):

self.event\_type = event\_type

# methods

def calculate\_total\_revenue(self):

return self.ticket\_price \* (self.total\_seats - self.available\_seats)

def get\_booked\_no\_of\_tickets(self):

return self.total\_seats - self.available\_seats

def book\_tickets(self, num\_tickets):

if num\_tickets > self.available\_seats:

print("Insufficient seats available!")

return False

else:

self.available\_seats -= num\_tickets

print(f"{num\_tickets} tickets booked successfully for the event '{self.event\_name}'")

return True

def cancel\_booking(self, num\_tickets):

if self.available\_seats + num\_tickets > self.total\_seats:

print("Invalid number of tickets to cancel!")

return False

else:

self.available\_seats += num\_tickets

print(f"{num\_tickets} tickets canceled successfully for the event '{self.event\_name}'")

return True

def display\_event\_details(self):

print("Event Details:")

print(f"Event Name: {self.event\_name}")

print(f"Event Date: {self.event\_date}")

print(f"Event Time: {self.event\_time}")

print(f"Venue: {self.venue\_name}")

print(f"Total Seats: {self.total\_seats}")

print(f"Available Seats: {self.available\_seats}")

print(f"Ticket Price: {self.ticket\_price}")

print(f"Event Type: {self.event\_type.value}")

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 get\_venue\_name(self):

return self.venue\_name

def set\_venue\_name(self, venue\_name):

self.venue\_name = venue\_name

def get\_address(self):

return self.address

def set\_address(self, address):

self.address = address

def display\_venue\_details(self):

print("Venue Details:")

print(f"Venue Name: {self.venue\_name}")

print(f"Address: {self.address}")

1. **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 Customer:

def \_\_init\_\_(self, customer\_name, email, phone\_number):

self.customer\_name = customer\_name

self.email = email

self.phone\_number = phone\_number

def get\_customer\_name(self):

return self.customer\_name

def set\_customer\_name(self, customer\_name):

self.customer\_name = customer\_name

def get\_email(self):

return self.email

def set\_email(self, email):

self.email = email

def get\_phone\_number(self):

return self.phone\_number

def set\_phone\_number(self, phone\_number):

self.phone\_number = phone\_number

def display\_customer\_details(self):

print("Customer Details:")

print(f"Name: {self.customer\_name}")

print(f"Email: {self.email}")

print(f"Phone Number: {self.phone\_number}")

1. **Booking** Class to represent the Tiket booking system. Perform the following operation in main

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

• **Methods and Constuctors:**

o **calculate\_booking\_cost(num\_tickets)**: Calculate and set the total cost of the

booking.

o **book\_tickets(num\_tickets)**: Book a specified number of tickets for an event.

o **cancel\_booking(num\_tickets)**: Cancel the booking and update the available seats.

o **getAvailableNoOfTickets()**: return the total available tickets

o **getEventDetails()**: return event details from the event class

class Booking:

def \_\_init\_\_(self, event):

self.event = event

self.num\_tickets = 0

self.total\_cost = 0

def calculate\_booking\_cost(self, num\_tickets):

self.num\_tickets = num\_tickets

self.total\_cost = self.num\_tickets \* self.event.get\_ticket\_price()

def book\_tickets(self, num\_tickets):

if self.event.book\_tickets(num\_tickets):

self.num\_tickets += num\_tickets

self.total\_cost += num\_tickets \* self.event.get\_ticket\_price()

print(f"{num\_tickets} tickets booked successfully!")

return True

else:

print("Failed to book tickets.")

return False

def cancel\_booking(self, num\_tickets):

if self.event.cancel\_booking(num\_tickets):

self.num\_tickets -= num\_tickets

self.total\_cost -= num\_tickets \* self.event.get\_ticket\_price()

print(f"{num\_tickets} tickets canceled successfully!")

return True

else:

print("Failed to cancel booking.")

return False

def get\_available\_no\_of\_tickets(self):

return self.event.get\_available\_seats()

def get\_event\_details(self):

return self.event.display\_event\_details()

**Task 5: Inheritance and polymorphism**

1. **Inheritance**

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

o **Attributes:**

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

2. ActorName

3. ActresName

o **Methods:**

1. Implement default constructors and overload the constructor with Customer

attributes, generate getter and setter methods.

1. **display\_event\_details():** Display movie details, including genre.

class Movie (Event):

def \_\_init\_\_(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, genre, actor\_name, actress\_name):

super().\_\_init\_\_(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, "Movie")

self.genre = genre

self.actor\_name = actor\_name

self.actress\_name = actress\_name

def display\_event\_details(self):

super().display\_event\_details()

print(f"Genre: {self.genre}")

print(f"Actor Name: {self.actor\_name}")

print(f"Actress Name: {self.actress\_name}")

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

methods:

o **Attributes:**

1. artist: Name of the performing artist or band.

2. type: (Theatrical, Classical, Rock, Recital)

o **Methods:**

1. Implement default constructors and overload the constructor with Customer

attributes, generate getter and setter methods.

2. **display\_concert\_details():** Display concert details, including the artist.

class Concert(Event):

def \_\_init\_\_(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, artist, concert\_type):

super().\_\_init\_\_(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, "Concert")

self.artist = artist

self.concert\_type = concert\_type

def display\_event\_details(self):

super().display\_event\_details()

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

• Create another subclass **Sports** that inherits from **Event**. Add the following attributes and

methods:

o **Attributes:**

1. sportName: Name of the game.

2. teamsName: (India vs Pakistan)

o **Methods:**

1. Implement default constructors and overload the constructor with Customer

attributes, generate getter and setter methods.

2. **display\_sport\_details():** Display concert details, including the artist.

class Sports(Event):

def \_\_init\_\_(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, sport\_name, teams\_name):

super().\_\_init\_\_(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, "Sports")

self.sport\_name = sport\_name

self.teams\_name = teams\_name

def display\_event\_details(self):

super().display\_event\_details()

print(f"Sport Name: {self.sport\_name}")

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

o **create\_event(event\_name: str, date:str, time:str, total\_seats: int, ticket\_price:**

**float, event\_type: str, venu\_name:str):** Create a new event with the specified details

and event type (movie, sport or concert) and return event object.

o **display\_event\_details(event: Event)**: Accepts an event object and calls its

**display\_event\_details()** method to display event details.

o **book\_tickets(event: Event, num\_tickets: int):**

1. Accepts an event object and the number of tickets to be booked.

2. Checks if there are enough available seats for the booking.

3. If seats are available, updates the available seats and returns the total cost

of the booking.

4. If seats are not available, displays a message indicating that the event is sold

out.

o **cancel\_tickets(event: Event, num\_tickets)**: cancel a specified number of tickets for

an event.

o **main():** simulates the ticket booking system

1. User can book tickets and view the event details as per their choice in menu

(movies, sports, concerts).

2. Display event details using the display\_event\_details() method without

knowing the specific event type (demonstrate polymorphism).

3. Make bookings using the book\_tickets() and cancel tickets cancel\_tickets()

method.

class TicketBookingSystem:

events = []

@classmethod

def create\_event(cls, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type):

if event\_type.lower() == "movie":

genre = input("Enter movie genre: ")

actor\_name = input("Enter actor name: ")

actress\_name = input("Enter actress name: ")

event = Movie(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, genre, actor\_name, actress\_name)

elif event\_type.lower() == "concert":

artist = input("Enter artist name: ")

concert\_type = input("Enter concert type: ")

event = Concert(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, artist, concert\_type)

elif event\_type.lower() == "sports":

sport\_name = input("Enter sport name: ")

teams\_name = input("Enter teams name: ")

event = Sports(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, sport\_name, teams\_name)

else:

print("Invalid event type.")

return None

cls.events.append(event)

return event

@classmethod

def display\_event\_details(cls, event):

event.display\_event\_details()

@classmethod

def book\_tickets(cls, event, num\_tickets):

if event.book\_tickets(num\_tickets):

print(f"{num\_tickets} tickets booked successfully!")

return event.calculate\_total\_revenue(num\_tickets)

else:

print("Failed to book tickets.")

return 0

@classmethod

def cancel\_tickets(cls, event, num\_tickets):

if event.cancel\_booking(num\_tickets):

print(f"{num\_tickets} tickets canceled successfully!")

return True

else:

print("Failed to cancel tickets.")

return False

@classmethod

def main(cls):

while True:

print("\n1. Create Event\n2. Display Event Details\n3. Book Tickets\n4. Cancel Tickets\n5. Exit")

choice = input("Enter your choice: ")

if choice == "1":

event\_name = input("Enter event name: ")

event\_date = input("Enter event date: ")

event\_time = input("Enter event time: ")

total\_seats = int(input("Enter total seats: "))

ticket\_price = float(input("Enter ticket price: "))

event\_type = input("Enter event type (movie/concert/sports): ")

venue\_name = input("Enter venue name: ")

cls.create\_event(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type)

elif choice == "2":

if cls.events:

for index, event in enumerate(cls.events, start=1):

print(f"\nEvent {index}:")

cls.display\_event\_details(event)

else:

print("No events created yet.")

elif choice == "3":

if cls.events:

event\_index = int(input("Enter event index to book tickets: ")) - 1

if 0 <= event\_index < len(cls.events):

event = cls.events[event\_index]

num\_tickets = int(input("Enter number of tickets to book:"))

cls.book\_tickets(event, num\_tickets)

else:

print("Invalid event index.")

else:

print("No events created yet.")

elif choice == "4":

if cls.events:

event\_index = int(input("Enter event index to cancel tickets: ")) - 1

if 0 <= event\_index < len(cls.events):

event = cls.events[event\_index]

num\_tickets = int(input("Enter number of tickets to cancel:"))

cls.cancel\_tickets(event, num\_tickets)

else:

print("Invalid event index.")

else:

print("No events created yet.")

elif choice == "5":

print("Exiting program.")

break

else:

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

**Task 6: Abstraction**

**Requirements:**

1. **Event Abstraction:**

• Create an abstract class **Event** that represents a generic event. It should include the

following attributes and methods as mentioned in *TASK 1*:

from abc import ABC, abstractmethod

# Abstract

class Event(ABC):

def \_\_init\_\_(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type):

self.event\_name = event\_name

self.event\_date = event\_date

self.event\_time = event\_time

self.venue\_name = venue\_name

self.total\_seats = total\_seats

self.available\_seats = total\_seats

self.ticket\_price = ticket\_price

self.event\_type = event\_type

@abstractmethod

def display\_event\_details(self):

pass

@abstractmethod

def calculate\_total\_revenue(self, num\_tickets):

pass

@abstractmethod

def book\_tickets(self, num\_tickets):

pass

@abstractmethod

def cancel\_booking(self, num\_tickets):

pass

**2.** Concrete **Event** Classes**:**

• Create three concrete classes that inherit from **Event** abstract class and override abstract

methods in concrete class should declare the variables as mentioned in above *Task 2*:

• Movie.

• Concert.

• Sport.

class Movie(Event):

def \_\_init\_\_(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, genre, actor\_name, actress\_name):

super().\_\_init\_\_(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type="Movie")

self.genre = genre

self.actor\_name = actor\_name

self.actress\_name = actress\_name

def display\_event\_details(self):

print(f"Event Name: {self.event\_name}")

print(f"Event Date: {self.event\_date}")

print(f"Event Time: {self.event\_time}")

print(f"Venue: {self.venue\_name}")

print(f"Total Seats: {self.total\_seats}")

print(f"Available Seats: {self.available\_seats}")

print(f"Ticket Price: {self.ticket\_price}")

print(f"Event Type: {self.event\_type}")

print(f"Genre: {self.genre}")

print(f"Actor: {self.actor\_name}")

print(f"Actress: {self.actress\_name}")

def calculate\_total\_revenue(self, num\_tickets):

return num\_tickets \* self.ticket\_price

def book\_tickets(self, num\_tickets):

if self.available\_seats >= num\_tickets:

self.available\_seats -= num\_tickets

return True

else:

return False

def cancel\_booking(self, num\_tickets):

self.available\_seats += num\_tickets

return True

class Concert(Event):

def \_\_init\_\_(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, artist, concert\_type):

super().\_\_init\_\_(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type="Concert")

self.artist = artist

self.concert\_type = concert\_type

def display\_event\_details(self):

print(f"Event Name: {self.event\_name}")

print(f"Event Date: {self.event\_date}")

print(f"Event Time: {self.event\_time}")

print(f"Venue: {self.venue\_name}")

print(f"Total Seats: {self.total\_seats}")

print(f"Available Seats: {self.available\_seats}")

print(f"Ticket Price: {self.ticket\_price}")

print(f"Event Type: {self.event\_type}")

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

print(f"Concert Type: {self.concert\_type}")

def calculate\_total\_revenue(self, num\_tickets):

return num\_tickets \* self.ticket\_price

def book\_tickets(self, num\_tickets):

if self.available\_seats >= num\_tickets:

self.available\_seats -= num\_tickets

return True

else:

return False

def cancel\_booking(self, num\_tickets):

self.available\_seats += num\_tickets

return True

class Sport(Event):

def \_\_init\_\_(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, sport\_name, teams\_name):

super().\_\_init\_\_(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type="Sports")

self.sport\_name = sport\_name

self.teams\_name = teams\_name

def display\_event\_details(self):

print(f"Event Name: {self.event\_name}")

print(f"Event Date: {self.event\_date}")

print(f"Event Time: {self.event\_time}")

print(f"Venue: {self.venue\_name}")

print(f"Total Seats: {self.total\_seats}")

print(f"Available Seats: {self.available\_seats}")

print(f"Ticket Price: {self.ticket\_price}")

print(f"Event Type: {self.event\_type}")

print(f"Sport: {self.sport\_name}")

print(f"Teams: {self.teams\_name}")

def calculate\_total\_revenue(self, num\_tickets):

return num\_tickets \* self.ticket\_price

def book\_tickets(self, num\_tickets):

if self.available\_seats >= num\_tickets:

self.available\_seats -= num\_tickets

return True

else:

return False

def cancel\_booking(self, num\_tickets):

self.available\_seats += num\_tickets

return True

1. **BookingSystem** Abstraction:

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

class BookingSystem(ABC):

events = []

@abstractmethod

def create\_event(self, event\_name, event\_date, event\_time, total\_seats, ticket\_price, event\_type, venue\_name):

pass

@abstractmethod

def book\_tickets(self, event, num\_tickets):

pass

@abstractmethod

def cancel\_tickets(self, event, num\_tickets):

pass

@abstractmethod

def get\_available\_seats(self, event):

pass

1. Concrete **TicketBookingSystem** Class:

• Create a concrete class **TicketBookingSystem** that inherits from BookingSystem:

• **TicketBookingSystem**: Implement the abstract methods to create events, book

tickets, and retrieve available seats. Maintain an array of events in this class.

• Create a simple user interface in a main method that allows users to interact with the ticket

booking system by entering commands such as "create\_event", "book\_tickets",

"cancel\_tickets", "get\_available\_seats," and "exit."

class TicketBookingSystem(BookingSystem):

def create\_event(self, event\_name, event\_date, event\_time, total\_seats, ticket\_price, event\_type, venue\_name):

if event\_type == "Movie":

genre = input("Enter genre: ")

actor\_name = input("Enter actor name: ")

actress\_name = input("Enter actress name: ")

event = Movie(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, genre, actor\_name, actress\_name)

elif event\_type == "Concert":

artist = input("Enter artist name: ")

concert\_type = input("Enter concert type: ")

event = Concert(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, artist, concert\_type)

elif event\_type == "Sports":

sport\_name = input("Enter sport name: ")

teams\_name = input("Enter teams playing: ")

event = Sport(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, sport\_name, teams\_name)

else:

print("Invalid event type!")

return None

self.events.append(event)

return event

def book\_tickets(self, event, num\_tickets):

if event.book\_tickets(num\_tickets):

print(f"{num\_tickets} tickets booked successfully!")

return event.calculate\_total\_revenue(num\_tickets)

else:

print("Failed to book tickets.")

return 0

def cancel\_tickets(self, event, num\_tickets):

if event.cancel\_booking(num\_tickets):

print(f"{num\_tickets} tickets canceled successfully!")

return True

else:

print("Failed to cancel tickets.")

return False

def get\_available\_seats(self, event):

return event.available\_seats

**Task 7: Has A Relation / Association**

**Create a Following classes with the following attributes and methods:**

1. **Venue** Class

• **Attributes**:

o venue\_name,

o address

• **Methods and Constuctors:**

o **display\_venue\_details():** Display venue details.

o Implement default constructors and overload the constructor with Customer

attributes, generate getter and setter methods.

created in task 4

1. **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.

created in task 4

1. **Event** sub classes**:**

• Create three sub classes that inherit from **Event** abstract class and override abstract

methods in concrete class should declare the variables as mentioned in above *Task 2*:

o Movie.

o Concert.

o Sport.

created in task 5

1. **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.

created in task 4

1. 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.

class Booking:

booking\_id\_counter = 0

def \_\_init\_\_(self, event, num\_tickets, customers):

Booking.booking\_id\_counter += 1

self.booking\_id = Booking.booking\_id\_counter

self.event = event

self.num\_tickets = num\_tickets

self.customers = customers

self.total\_cost = event.ticket\_price \* num\_tickets

self.booking\_date = datetime.now()

def display\_booking\_details(self):

print(f"Booking ID: {self.booking\_id}")

print("Event Details:")

self.event.display\_event\_details()

print(f"Number of Tickets: {self.num\_tickets}")

print(f"Total Cost: {self.total\_cost}")

print(f"Booking Date: {self.booking\_date}")

self.customers.display\_customer\_details()

6. **BookingSystem** Class to represent the Ticket booking system. Perform the following operation in main method. Note: - Use Event class object for the following operation.

• **Attributes**

o array of events

• **Methods and Constuctors:**

o **create\_event(event\_name: str, date:str, time:str, total\_seats: int, ticket\_price:**

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

event type (movie, sport or concert) and return event object.

o **calculate\_booking\_cost(num\_tickets)**: Calculate and set the total cost of the

booking.

o **book\_tickets(eventname:str, num\_tickets, arrayOfCustomer)**: Book a specified

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

and stored in array also should update the attributes of **Booking** class.

o **cancel\_booking(booking\_id)**: Cancel the booking and update the available seats.

o **getAvailableNoOfTickets()**: return the total available tickets

o **getEventDetails()**: return event details from the event class

o Create a simple user interface in a **main method** that allows users to interact with

the ticket booking system by entering commands such as "create\_event",

"book\_tickets", "cancel\_tickets", "get\_available\_seats,", "get\_event\_details," and

"exit."

created in task 5

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

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

created in task 4

1. **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.

created in task 4

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

created in task 4

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

created in task 4

1. Create interface/abstract class **IEventServiceProvider** with following methods:

• **create\_event(event\_name: str, date:str, time:str, total\_seats: int, ticket\_price: float,**

**event\_type: str, venu: Venu):** Create a new event with the specified details and event type

(movie, sport or concert) and return event object.

• **getEventDetails():** return array of event details from the event class.

• **getAvailableNoOfTickets()**: return the total available tickets.

from abc import ABC, abstractmethod

class IEventServiceProvider(ABC):

@abstractmethod

def create\_event(self, event\_name: str, date: str, time: str, total\_seats: int, ticket\_price: float, event\_type: str, venue: Venue):

pass

@abstractmethod

def get\_event\_details(self):

pass

@abstractmethod

def get\_available\_no\_of\_tickets(self):

pass

1. Create interface/abstract class **IBookingSystemServiceProvider** with following methods:

• **calculate\_booking\_cost(num\_tickets):** Calculate and set the total cost of the booking.

• **book\_tickets(eventname:str, num\_tickets, arrayOfCustomer):** Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array

also should update the attributes of Booking class.

• **cancel\_booking(booking\_id):** Cancel the booking and update the available seats.

• **get\_booking\_details(booking\_id):**get the booking details.

from abc import ABC, abstractmethod

class IBookingSystemServiceProvider(IEventServiceProvider):

@abstractmethod

def calculate\_booking\_cost(self, num\_tickets: int):

pass

@abstractmethod

def book\_tickets(self, event\_name: str, num\_tickets: int, customers: list):

pass

@abstractmethod

def cancel\_booking(self, booking\_id: int):

pass

@abstractmethod

def get\_booking\_details(self, booking\_id: int):

pass

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

from service.IEventServiceProvider import IEventServiceProvider

class EventServiceProviderImpl(IEventServiceProvider):

def \_\_init\_\_(self):

self.events = []

def create\_event(self, event\_name, event\_date, event\_time,venue\_name, total\_seats, ticket\_price, event\_type ):

if event\_type == "Movie":

genre = input("Enter genre: ")

actor\_name = input("Enter actor name: ")

actress\_name = input("Enter actress name: ")

event = Movie(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, genre, actor\_name, actress\_name)

elif event\_type == "Concert":

artist = input("Enter artist name: ")

concert\_type = input("Enter concert type: ")

event = Concert(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, artist, concert\_type)

elif event\_type == "Sports":

sport\_name = input("Enter sport name: ")

teams\_name = input("Enter teams playing: ")

event = Sports(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, sport\_name, teams\_name)

else:

print("Invalid event type!")

return None

self.events.append(event)

return event

def get\_event\_details(self):

self.event.display\_event\_details()

def get\_available\_no\_of\_tickets(self):

self.event.get\_available\_seats()

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

attributes.

• **Attributes**

o array of events

from service.IBookingSystemServiceProvider import IBookingSystemServiceProvider

from bean.EventServiceProviderImpl import EventServiceProviderImpl

class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def calculate\_booking\_cost(self, num\_tickets: int):

return self.get\_ticket\_price() \* num\_tickets

def book\_tickets(self, event, num\_tickets):

if event.book\_tickets(num\_tickets):

print(f"{num\_tickets} tickets booked successfully!")

return event.calculate\_total\_revenue(num\_tickets)

else:

print("Failed to book tickets.")

return 0

def cancel\_tickets(self, event, num\_tickets):

if event.cancel\_booking(num\_tickets):

print(f"{num\_tickets} tickets canceled successfully!")

return True

else:

print("Failed to cancel tickets.")

return False

def get\_booking\_details(self, booking\_id: int):

self.display\_booking\_details()

9. Create **TicketBookingSystem** class and perform following operations:

• Create a simple user interface in a main method that allows users to interact with the ticket

booking system by entering commands such as "create\_event", "book\_tickets",

"cancel\_tickets", "get\_available\_seats,", "get\_event\_details," and "exit."

1. Place the interface/abstract class in service package and interface/abstract class implementation class, all concrete class in bean package and **TicketBookingSystem** class in app package.

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

created in task 5

**Task 9: Exception Handling**

throw the exception whenever needed and Handle in main method,

1. **EventNotFoundException** throw this exception when user try to book the tickets for Event not listed

in the menu.

2. **InvalidBookingIDException** throw this exception when user entered the invalid bookingId when he

tries to view the booking or cancel the booking.

3. **NullPointerException** handle in main method**.**

Throw these exceptions from the methods in **TicketBookingSystem** class. Make necessary changes to accommodate exception in the source code. Handle all these exceptions from the main program.

class EventNotFoundException(Exception):

def \_\_init\_\_(self, event\_name):

self.event\_name = event\_name

super().\_\_init\_\_(f"Event '{event\_name}' not found.")

class InvalidBookingIDException(Exception):

def \_\_init\_\_(self, booking\_id):

self.booking\_id = booking\_id

super().\_\_init\_\_(f"Invalid booking ID '{booking\_id}'.")

class NullPointerException(Exception):

def \_\_init\_\_(self, message="Null pointer exception occurred."):

super().\_\_init\_\_(message)

**Task 10: Collection**

1. From the previous task change the **Booking** class attribute customers to List of customers and **BookingSystem** class attribute events to List of events and perform the same operation.

from typing import List

class Booking:

def \_\_init\_\_(self, booking\_id, customers: List[Customer], event: Event, num\_tickets, total\_cost, booking\_date):

self.booking\_id = booking\_id

self.customers = customers

self.event = event

self.num\_tickets = num\_tickets

self.total\_cost = total\_cost

self.booking\_date = booking\_date

class BookingSystem:

def \_\_init\_\_(self):

self.events = []

2. From the previous task change all list type of attribute to type Set in **Booking** and **BookingSystem**

class and perform the same operation.

• Avoid adding duplicate Account object to the set.

• Create Comparator<Event> object to sort the event based on event name and location in

alphabetical order.

from typing import Set

class Booking:

def \_\_init\_\_(self, booking\_id, customers: Set[Customer], event: Event, num\_tickets, total\_cost, booking\_date):

self.booking\_id = booking\_id

self.customers = customers

self.event = event

self.num\_tickets = num\_tickets

self.total\_cost = total\_cost

self.booking\_date = booking\_date

class BookingSystem:

def \_\_init\_\_(self):

self.events = set()

**3.** From the previous task change all list type of attribute to type Map object in **Booking** and

**BookingSystem** class and perform the same operation.

from typing import Dict

class Booking:

def \_\_init\_\_(self, booking\_id, customers: Dict[int, Customer], event: Event, num\_tickets, total\_cost, booking\_date):

self.booking\_id = booking\_id

self.customers = customers

self.event = event

self.num\_tickets = num\_tickets

self.total\_cost = total\_cost

self.booking\_date = booking\_date

class BookingSystem:

def \_\_init\_\_(self):

self.events = {}

**Task 11: Database Connectivity.**

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

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

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

4. Create I**BookingSystemRepository** interface/abstract class which include following methods to interact with database.

• **create\_event(event\_name: str, date:str, time:str, total\_seats: int, ticket\_price: float,**

**event\_type: str, venu: Venu):** Create a new event with the specified details and event type

(movie, sport or concert) and return event object and should store in database.

• **getEventDetails():** return array of event details from the database.

• **getAvailableNoOfTickets()**: return the total available tickets from the database.

• **calculate\_booking\_cost(num\_tickets):** Calculate and set the total cost of the booking.

• **book\_tickets(eventname:str, num\_tickets, listOfCustomer):** Book a specified number of

tickets for an event. for each tickets customer object should be created and stored in array

also should update the attributes of Booking class and stored in database.

• **cancel\_booking(booking\_id):** Cancel the booking and update the available seats and stored

in database.

• **get\_booking\_details(booking\_id):** get the booking details from database.

from abc import ABC, abstractmethod

class IBookingSystemRepository(ABC):

@abstractmethod

def create\_event(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type):

pass

@abstractmethod

def getEventDetails(self) :

pass

@abstractmethod

def getAvailableNoOfTickets(self) :

pass

@abstractmethod

def calculate\_booking\_cost(self, num\_tickets, ticket\_price):

pass

@abstractmethod

def book\_tickets(self, event, num\_tickets, customer):

pass

@abstractmethod

def cancel\_booking(self, booking\_id: int) :

pass

@abstractmethod

def get\_booking\_details(self, booking\_id: int) :

pass

5. Create **BookingSystemRepositoryImpl** interface/abstract class which implements

I**BookingSystemRepository** interface/abstract class and provide implementation of all methods and perform the database operations.

from TicketBookingSystem.entity.movie import Movie

from TicketBookingSystem.entity.concert import Concert

from TicketBookingSystem.entity.sports import Sports

from TicketBookingSystem.entity.booking import Booking

from TicketBookingSystem.service.IBookingSystemRepository import IBookingSystemRepository

class BookingSystemRepositoryImpl(IBookingSystemRepository):

def \_\_init\_\_(self, db\_conn):

self.db\_conn = db\_conn

self.events = {}

self.bookings = {}

def create\_event(self, event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type):

if event\_type.lower() == "movie":

genre = input("Enter movie genre: ")

actor\_name = input("Enter actor name: ")

actress\_name = input("Enter actress name: ")

event = Movie(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, genre, actor\_name, actress\_name)

elif event\_type.lower() == "concert":

artist = input("Enter artist name: ")

concert\_type = input("Enter concert type: ")

event = Concert(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, artist, concert\_type)

elif event\_type.lower() == "sports":

sport\_name = input("Enter sport name: ")

teams\_name = input("Enter teams name: ")

event = Sports(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, sport\_name, teams\_name)

else:

print("Invalid event type.")

return None

try:

cursor = self.db\_conn.cursor()

cursor.execute(

"INSERT INTO Event (event\_id, event\_name, event\_date, event\_time, venue\_name, total\_seats, available\_seats, ticket\_price, event\_type) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)",

(event.event\_id, event\_date, event\_time, venue\_name, total\_seats, total\_seats, ticket\_price, event\_type))

self.db\_conn.commit()

self.events[event.event\_id] = event

return event

except mysql.connector.Error as e:

print(f"Error creating event: {e}")

def getEventDetails(self) :

try:

cursor = self.db\_conn.cursor()

cursor.execute("SELECT \* FROM Event")

events = cursor.fetchall()

cursor.close()

return events

except mysql.connector.Error as e:

raise exception (f"Error fetching event details: {e}")

def getAvailableNoOfTickets(self) :

try:

cursor = self.db\_conn.cursor()

cursor.execute("SELECT SUM(available\_seats) FROM Event")

total\_available\_tickets = cursor.fetchone()[0]

cursor.close()

print(f"total available tickets ={total\_available\_tickets}")

return total\_available\_tickets

except mysql.connector.Error as e:

raise exception(f"Error fetching available tickets: {e}")

def calculate\_booking\_cost(self, num\_tickets, ticket\_price):

return num\_tickets \* ticket\_price

def book\_tickets(self, event, num\_tickets, customer) :

try:

cursor = self.db\_conn.cursor()

cursor.execute("SELECT available\_seats FROM Event WHERE event\_name = %s", (event.event\_id))

available\_seats = cursor.fetchone()[0]

if available\_seats >= num\_tickets:

cursor.execute("UPDATE Event SET available\_seats = available\_seats - %s WHERE event\_id = %s",

(num\_tickets, event.event\_id))

self.db\_conn.commit()

booking = Booking(event, num\_tickets, customer)

self.bookings[booking.booking\_id] = booking

cursor.execute(

"INSERT INTO Booking (booking\_id, event\_id, customer\_id, num\_tickets, total\_cost, booking\_date) VALUES (%s, %s, %s, %s, %s, %s)",

(booking.booking\_id, event.event\_id, customer.customer\_id, num\_tickets, booking.total\_cost, booking.booking\_date))

self.db\_conn.commit()

cursor.close()

print(f"{num\_tickets} tickets booked successfully for {event.event\_name}, booking id = {booking.booking\_id}")

return booking

else:

print("Not enough available seats for booking")

except mysql.connector.Error as e :

raise exception ProcessLookupError(f"Error booking tickets: {e}")

def cancel\_booking(self, booking\_id):

try:

cursor = self.db\_conn.cursor()

cursor.execute("SELECT event\_id, num\_tickets FROM Booking WHERE booking\_id = %s", (booking\_id,))

booking\_details = cursor.fetchone()

if booking\_details:

event\_id, num\_tickets = booking\_details

cursor.execute("UPDATE Event SET available\_seats = available\_seats + %s WHERE event\_id = %s",

(num\_tickets, event\_id))

cursor.execute("DELETE FROM Booking WHERE booking\_id = %s", (booking\_id,))

self.db\_conn.commit()

cursor.close()

else:

print("Booking ID not found")

except:

print(f"Error cancelling booking: {None}")

def get\_booking\_details(self, booking\_id):

try:

cursor = self.db\_conn.cursor()

cursor.execute("SELECT \* FROM Booking WHERE booking\_id = %s", (booking\_id,))

booking\_details = cursor.fetchone()

cursor.close()

return booking\_details

except mysql.connector.Error as e :

raise exception(f"Error fetching booking details: {None}")

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

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

Connection reference

import mysql.connector

class DBUtil:

@staticmethod

def getDBConn():

try:

conn = (mysql.connector.connect(

host="localhost",

user="root",

password="root",

port='3306',

database="TBS"

))

return conn

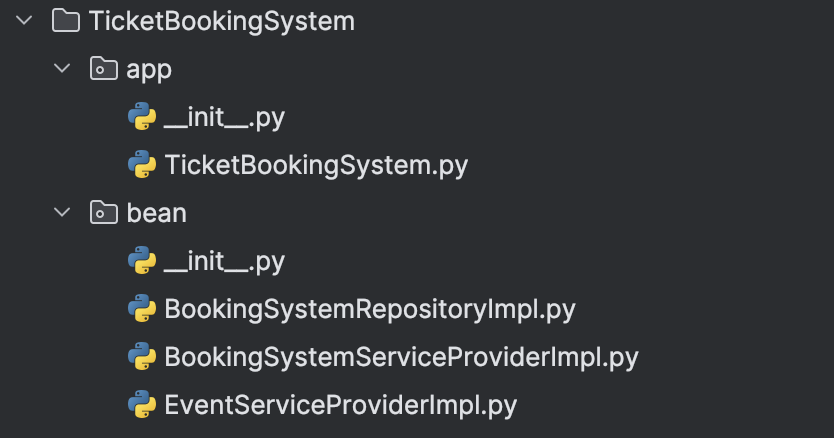
except mysql.connector.Error as e:

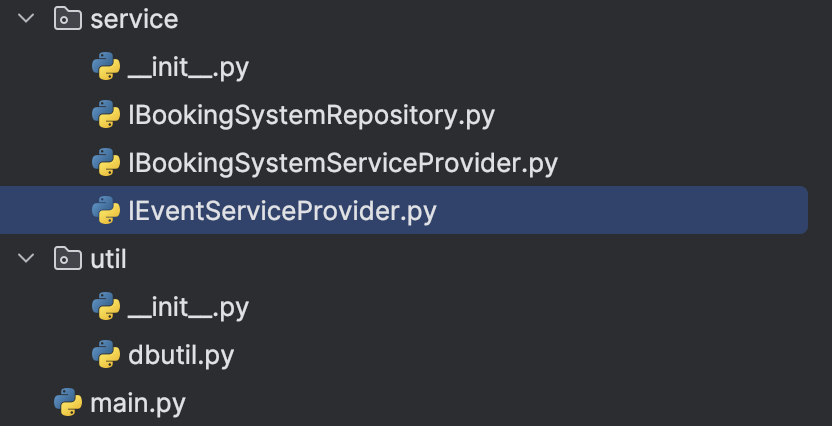
print(f"Error connecting to MySQL database: {e}")

return None

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

class in bean package and **TicketBookingSystemRepository** class in app package.





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

9. Create **TicketBookingSystem** class and perform following operations:

• Create a simple user interface in a main method that allows users to interact with the ticket

booking system by entering commands such as "create\_event", "book\_tickets",

"cancel\_tickets", "get\_available\_seats,", "get\_event\_details," and "exit."

from TicketBookingSystem.bean.BookingSystemRepositoryImpl import BookingSystemRepositoryImpl

from TicketBookingSystem.entity.customer import Customer

class TicketBookingSystem:

def \_\_init\_\_(self, db\_conn):

self.booking\_system\_repository = BookingSystemRepositoryImpl(db\_conn)

def display\_menu(self):

print("===== Ticket Booking System =====")

print("1. Create Event")

print("2. View Event Details")

print("3. Book Tickets")

print("4. Cancel Booking")

print("5. View Available Seats")

print("6. Exit")

def create\_event(self):

event\_name = input("Enter event name: ")

event\_date = input("Enter event date (YYYY-MM-DD): ")

event\_time = input("Enter event time (HH:MM:SS): ")

venue\_name = input("Enter venue name: ")

total\_seats = int(input("Enter total seats: "))

ticket\_price = float(input("Enter ticket price: "))

event\_type = input("Enter event type (Movie/Sports/Concert): ")

self.booking\_system\_repository.create\_event(event\_name, event\_date, event\_time, venue\_name, total\_seats, ticket\_price, event\_type)

def view\_event\_details(self):

event\_details = self.booking\_system\_repository.getEventDetails()

if event\_details:

for event in event\_details:

print(event)

else:

print("no events found")

def book\_tickets(self):

self.view\_event\_details()

event\_id = int(input("Enter event ID to book tickets: "))

try:

event = self.booking\_system\_repository.events[event\_id]

except:

print("event id not found")

return None

num\_tickets = int(input("Enter number of tickets to book: "))

# Creating customer

customer\_name = input("Enter customer name: ")

email = input("Enter email: ")

phone\_number = input("Enter phone number: ")

customer = Customer(customer\_name, email, phone\_number)

# Book tickets

self.booking\_system\_repository.book\_tickets(event, num\_tickets, customer)

def cancel\_booking(self):

booking\_id = int(input("Enter booking ID to cancel: "))

self.booking\_system\_repository.cancel\_booking(booking\_id)

def view\_available\_seats(self):

available\_seats = self.booking\_system\_repository.getAvailableNoOfTickets()

print(f"Total available seats: {available\_seats}")

def run(self):

while True:

self.display\_menu()

choice = input("Enter your choice: ")

if choice == "1":

self.create\_event()

elif choice == "2":

self.view\_event\_details()

elif choice == "3":

self.book\_tickets()

elif choice == "4":

self.cancel\_booking()

elif choice == "5":

self.view\_available\_seats()

elif choice == "6":

print("Exiting...")

break

else:

print("Invalid choice. Please enter a valid option.")