

## Ticket Booking System

### Control structure

#### Task 1: Conditional Statements

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

##### Tasks:

1. Write a program that takes the availableTicket and noOfBookingTicket as input.

Code

2. Use conditional statements (if-else) to determine if the ticket is available or not.
3. Display an appropriate message based on ticket availability.

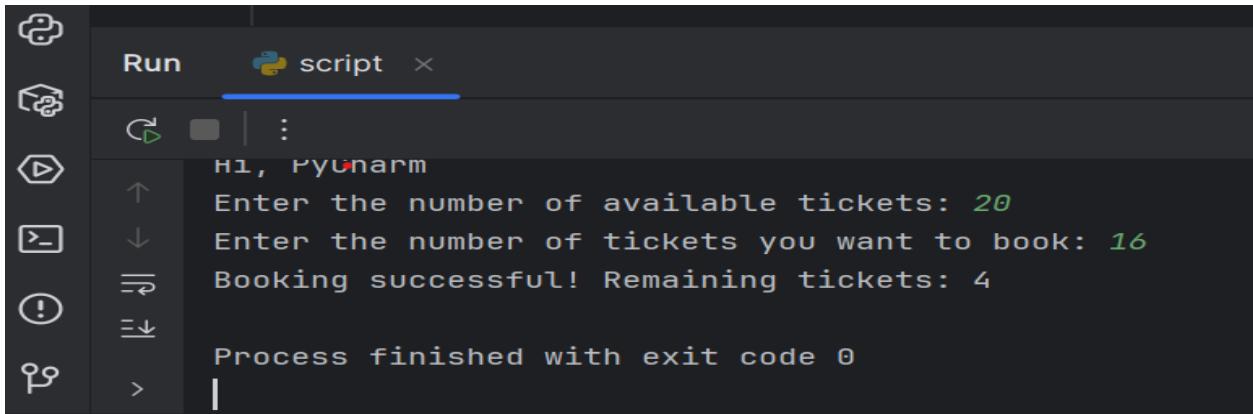
Code :

```
# Step 1: Take input
available_ticket = int(input("Enter the number of available tickets: "))
no_of_booking_ticket = int(input("Enter the number of tickets you want to book: "))

# Step 2: Use conditional statement to check availability
if available_ticket >= no_of_booking_ticket:
    remaining = available_ticket - no_of_booking_ticket
    print(f"Booking successful! Remaining tickets: {remaining}")
else:
    print("Sorry, not enough tickets available.")
```

```
# Step 1: Take input
available_ticket = int(input("Enter the number of available tickets: "))
no_of_booking_ticket = int(input("Enter the number of tickets you want to book: "))

# Step 2: Use conditional statement to check availability
if available_ticket >= no_of_booking_ticket:
    remaining = available_ticket - no_of_booking_ticket
    print(f"Booking successful! Remaining tickets: {remaining}")
else:
    print("Sorry, not enough tickets available.")
```



The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar has a 'Run' button, a 'script' tab, and a close button. The left sidebar contains various icons for file operations like new file, open, save, and help. The main area displays the following Python script and its output:

```
Hi, Pyunarm
Enter the number of available tickets: 20
Enter the number of tickets you want to book: 16
Booking successful! Remaining tickets: 4
Process finished with exit code 0
```

```
Enter the number of available tickets: 20
Enter the number of tickets you want to book: 22
Sorry, not enough tickets available.

Process finished with exit code 0
```

### Task 2: Nested Conditional Statements

Create a program that simulates a Ticket booking and calculating cost of tickets. Display tickets options such as "Silver", "Gold", "Dimond". Based on ticket category fix the base ticket price and get the user input for ticket type and no of tickets need and calculate the total cost of tickets booked.

Code :

```
# Step 1: Display ticket options
print("Ticket Categories:")
print("1. Silver - ₹150")
print("2. Gold - ₹250")
print("3. Diamond - ₹500")
ticket_type = input("Enter ticket category (Silver/Gold/Diamond): ").strip().capitalize()
no_of_tickets = int(input("Enter number of tickets to book: "))

# Step 3: Use nested conditionals to determine price and calculate total
if no_of_tickets > 0:
    if ticket_type == "Silver":
        price = 150
    elif ticket_type == "Gold":
        price = 250
    elif ticket_type == "Diamond":
```

```
    price = 500
else:
    price = 0
    print("Invalid ticket category selected.")

# Step 4: Calculate total cost only if category was valid
if price > 0:
    total_cost = price * no_of_tickets
    print(f"Total cost for {no_of_tickets} {ticket_type} ticket(s): ₹{total_cost}")
else:
    print("Invalid number of tickets.")
```

The screenshot shows a code editor window with a file named 'script.py'. The code is a script for calculating ticket costs based on category and quantity. It includes comments, print statements, and conditional logic. The code is color-coded for readability.

```
script.py
31     # Step 1: Display ticket options
32     print("Ticket Categories:")
33     print("1. Silver - ₹150")
34     print("2. Gold - ₹250")
35     print("3. Diamond - ₹500")
36     ticket_type = input("Enter ticket category (Silver/Gold/Diamond): ").strip().capitalize()
37     no_of_tickets = int(input("Enter number of tickets to book: "))
38
39     # Step 3: Use nested conditionals to determine price and calculate total
40     if no_of_tickets > 0:
41         if ticket_type == "Silver":
42             price = 150
43         elif ticket_type == "Gold":
44             price = 250
45         elif ticket_type == "Diamond":
46             price = 500
47         else:
48             price = 0
49             print("Invalid ticket category selected.")
50
51     # Step 4: Calculate total cost only if category was valid
52     if price > 0:
53         total_cost = price * no_of_tickets
54         print(f"Total cost for {no_of_tickets} {ticket_type} ticket(s): ₹{total_cost}")
55     else:
56         print("Invalid number of tickets.")
```

Output :

The screenshot shows a terminal window titled 'Run' with a Python script running. The output shows the program prompting for ticket categories and quantities, and then calculating and printing the total cost.

```
Run script
Booking successful! Remaining tickets: 7
Ticket Categories:
1. Silver - ₹150
2. Gold - ₹250
3. Diamond - ₹500
Enter ticket category (Silver/Gold/Diamond): Silver
Enter number of tickets to book: 4
Total cost for 4 Silver ticket(s): ₹600
Process finished with exit code 0
```

### Task 3: Looping

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

Code :

```
print("Welcome to the Ticket Booking System")
print("Type 'Exit' anytime to stop booking.\n")
while True:
    print("\nTicket Categories:")
    print("1. Silver - ₹150")
    print("2. Gold - ₹250")
    print("3. Diamond - ₹500")
    ticket_type = input("Enter ticket category (Silver/Gold/Diamond) or type 'Exit' to stop:")
    ticket_type.strip().capitalize()
    if ticket_type == "Exit":
        print("Thank you for using the Ticket Booking System!")
        break
    try:
        no_of_tickets = int(input("Enter number of tickets to book: "))
    except ValueError:
        print("Invalid input. Please enter a number for tickets.")
        continue
    if no_of_tickets <= 0:
        print("Number of tickets must be greater than 0.")
        continue
    if ticket_type == "Silver":
        price = 150
    elif ticket_type == "Gold":
        price = 250
    elif ticket_type == "Diamond":
        price = 500
    else:
        print("Invalid ticket category selected.")
        continue
    total_cost = price * no_of_tickets
    print(f"Booking successful! {no_of_tickets} {ticket_type} ticket(s) booked. Total cost: ₹{total_cost}")
```

code ss:

The screenshot shows the PyCharm interface with the 'script.py' file open. The code implements a ticket booking system. It prints a welcome message and a list of ticket categories (Silver, Gold, Diamond). It then enters a loop where it asks for the number of tickets and calculates the total cost based on the category selected. If the category is invalid, it prints an error message. If the number of tickets is less than or equal to zero, it prints an error message and continues. If the user types 'Exit', the program exits.

```
59     print("🔴 Welcome to the Ticket Booking System 🔴")
60     print("Type 'Exit' anytime to stop booking.\n")
61     while True:
62         print("\nTicket Categories:")
63         print("1. Silver - ₹150")
64         print("2. Gold   - ₹250")
65         print("3. Diamond - ₹500")
66         ticket_type = input("Enter ticket category (Silver/Gold/Diamond) or type 'Exit' to stop: ").strip().capitalize()
67         if ticket_type == "Exit":
68             print("Thank you for using the Ticket Booking System!")
69             break
70         try:
71             no_of_tickets = int(input("Enter number of tickets to book: "))
72         except ValueError:
73             print("Invalid input. Please enter a number for tickets.")
74             continue
75         if no_of_tickets <= 0:
76             print("Number of tickets must be greater than 0.")
77             continue
78         if ticket_type == "Silver":
79             price = 150
80         elif ticket_type == "Gold":
81             price = 250
82         elif ticket_type == "Diamond":
83             price = 500
84         else:
85             print("Invalid ticket category selected.")
86             continue
87         total_cost = price * no_of_tickets
88         print(f"✅ Booking successful! {no_of_tickets} {ticket_type} ticket(s) booked. Total cost: ₹{total_cost}!")
```

## Output :

The screenshot shows the PyCharm 'Run' tab with the output of the 'script.py' run. The output shows the program's interaction with the user, including the welcome message, ticket categories, user inputs for ticket type and quantity, and the resulting total cost. The process exits with code 0 at the end.

```
Enter ticket category (Silver/Gold/Diamond): gold
Enter number of tickets to book: 4
Total cost for 4 Gold ticket(s): ₹1000
🔴 Welcome to the Ticket Booking System 🔴
Type 'Exit' anytime to stop booking.

Ticket Categories:
1. Silver - ₹150
2. Gold   - ₹250
3. Diamond - ₹500
Enter ticket category (Silver/Gold/Diamond) or type 'Exit' to stop: gold
Enter number of tickets to book: 2
✅ Booking successful! 2 Gold ticket(s) booked. Total cost: ₹500

Ticket Categories:
1. Silver - ₹150
2. Gold   - ₹250
3. Diamond - ₹500
Enter ticket category (Silver/Gold/Diamond) or type 'Exit' to stop: exit
Thank you for using the Ticket Booking System!

Process finished with exit code 0
```

#### Task 4: Class & Object

Create a Following classes with the following attributes and methods:

##### 1. Event Class:

- **Attributes:**

- event\_name, ○ event\_date DATE, ○ event\_time TIME, ○ venue\_name, ○ total\_seats,
  - available\_seats, ○ ticket\_price DECIMAL,
  - event\_type ENUM('Movie', 'Sports', 'Concert')

- **Methods and Constructors:**

- Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
- **calculate\_total\_revenue():** Calculate and return the total revenue based on the number of tickets sold.
- **getBookedNoOfTickets():** return the total booked tickets
- **book\_tickets(num\_tickets):** Book a specified number of tickets for an event. Initially available seats are equal to the total seats when tickets are booked available seats number should be reduced.
- **cancel\_booking(num\_tickets):** Cancel the booking and update the available seats.
- **display\_event\_details():** Display event details, including event name, date time seat availability.

Code :

```
from datetime import date, time
```

```
class Event:
```

```
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type="Movie"):
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type
```

```
# ----- Getters -----
```

```
    def get_event_name(self): return self.event_name
    def get_event_date(self): return self.event_date
    def get_event_time(self): return self.event_time
    def get_venue_name(self): return self.venue_name
    def get_total_seats(self): return self.total_seats
    def get_available_seats(self): return self.available_seats
    def get_ticket_price(self): return self.ticket_price
    def get_event_type(self): return self.event_type
```

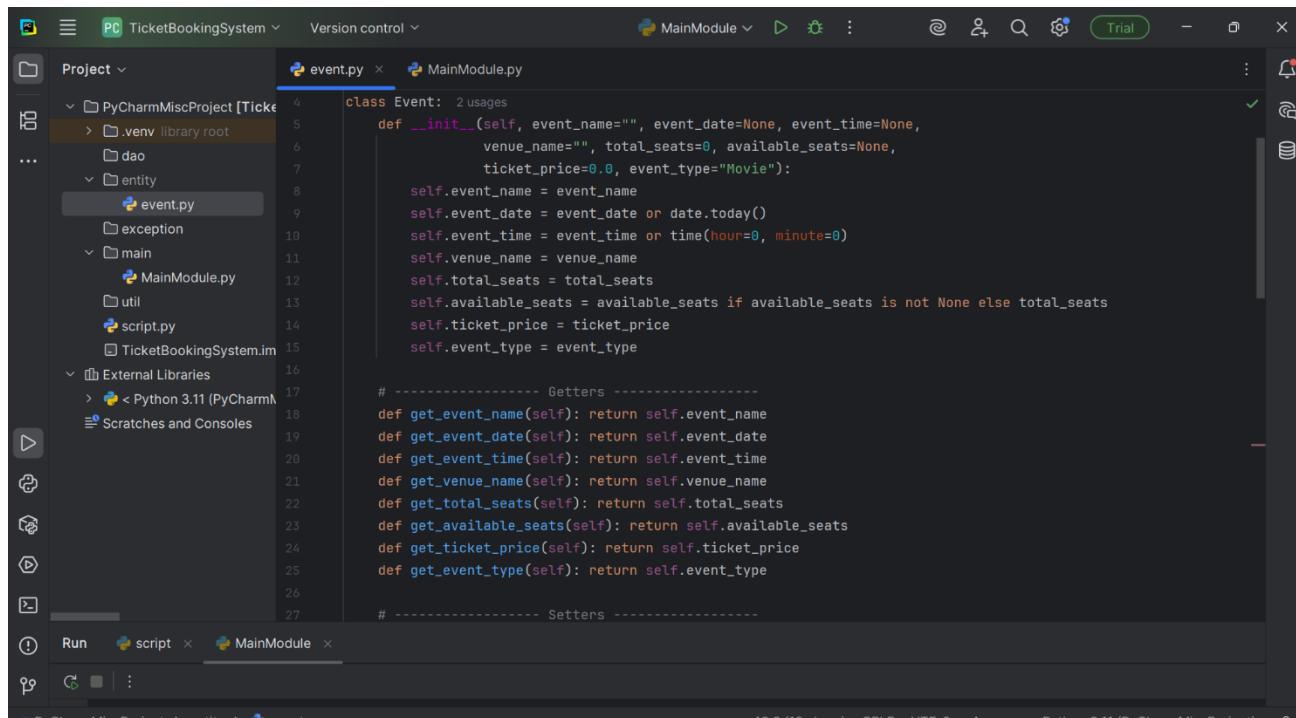
```
def set_event_name(self, name): self.event_name = name
def set_event_date(self, event_date): self.event_date = event_date
def set_event_time(self, event_time): self.event_time = event_time
def set_venue_name(self, venue): self.venue_name = venue
def set_total_seats(self, seats): self.total_seats = seats
def set_available_seats(self, seats): self.available_seats = seats
def set_ticket_price(self, price): self.ticket_price = price
def set_event_type(self, e_type): self.event_type = e_type
def calculate_total_revenue(self):
    booked = self.total_seats - self.available_seats
    return booked * self.ticket_price

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

def book_tickets(self, num_tickets):
    if num_tickets <= self.available_seats:
        self.available_seats -= num_tickets
        print(f"✅ {num_tickets} ticket(s) booked successfully.")
    else:
        print("❌ Not enough tickets available.")

def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets <= self.total_seats:
        self.available_seats += num_tickets
        print(f"✅ {num_tickets} ticket(s) cancelled successfully.")
    else:
        print("❌ Cannot cancel more than total booked tickets.")

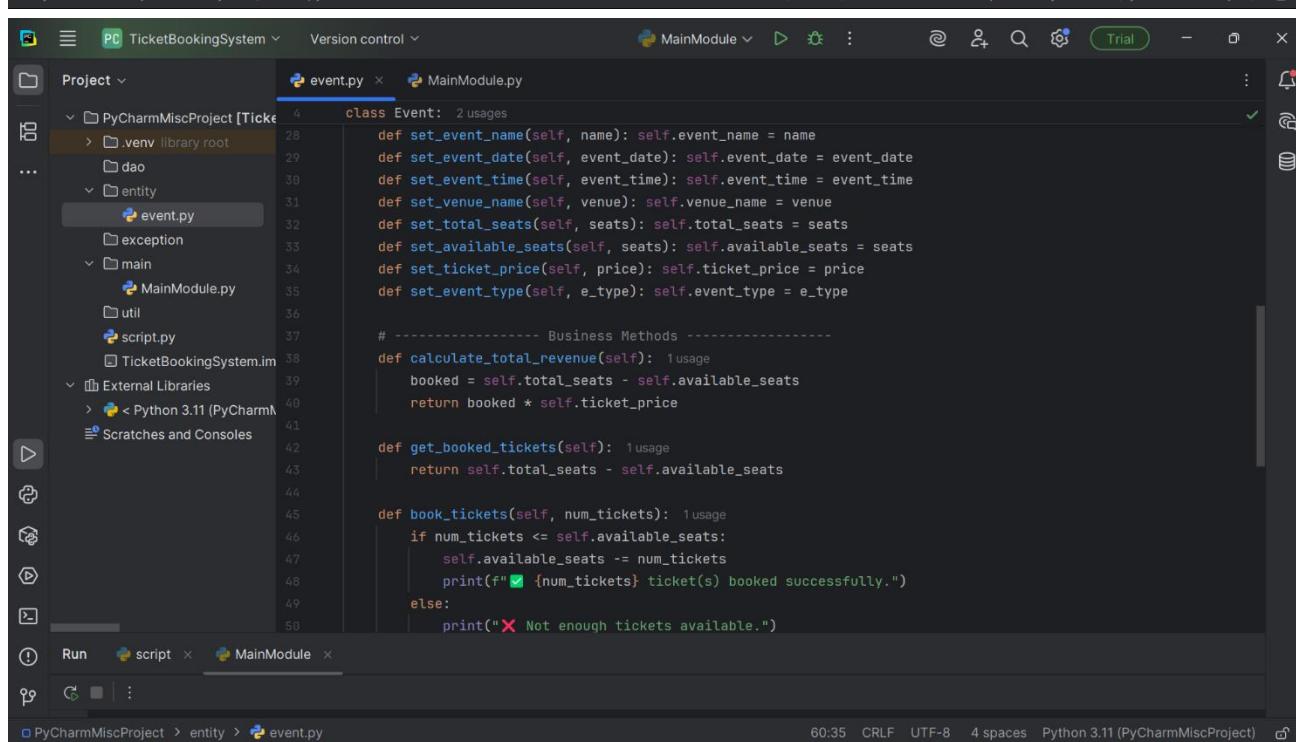
def display_event_details(self):
    print("\n💻 Event Details")
    print(f"Event Name : {self.event_name}")
    print(f"Event Date : {self.event_date}")
    print(f"Event Time : {self.event_time}")
    print(f"Venue Name : {self.venue_name}")
    print(f"Event Type : {self.event_type}")
    print(f"Total Seats : {self.total_seats}")
    print(f"Available Seats : {self.available_seats}")
    print(f"Ticket Price (₹) : {self.ticket_price:.2f}")
```



The screenshot shows the PyCharm IDE interface with the project 'TicketBookingSystem' open. The left sidebar displays the project structure with files like event.py, MainModule.py, and exception.py. The main editor window shows the code for the Event class:

```
4   class Event: 2 usages
5       def __init__(self, event_name="", event_date=None, event_time=None,
6                   venue_name="", total_seats=0, available_seats=None,
7                   ticket_price=0.0, event_type="Movie"):
8           self.event_name = event_name
9           self.event_date = event_date or date.today()
10          self.event_time = event_time or time(hour=0, minute=0)
11          self.venue_name = venue_name
12          self.total_seats = total_seats
13          self.available_seats = available_seats if available_seats is not None else total_seats
14          self.ticket_price = ticket_price
15          self.event_type = event_type
16
17      # ----- Getters -----
18      def get_event_name(self): return self.event_name
19      def get_event_date(self): return self.event_date
20      def get_event_time(self): return self.event_time
21      def get_venue_name(self): return self.venue_name
22      def get_total_seats(self): return self.total_seats
23      def get_available_seats(self): return self.available_seats
24      def get_ticket_price(self): return self.ticket_price
25      def get_event_type(self): return self.event_type
26
27      # ----- Setters -----
28      def set_event_name(self, name): self.event_name = name
29      def set_event_date(self, event_date): self.event_date = event_date
30      def set_event_time(self, event_time): self.event_time = event_time
31      def set_venue_name(self, venue): self.venue_name = venue
32      def set_total_seats(self, seats): self.total_seats = seats
33      def set_available_seats(self, seats): self.available_seats = seats
34      def set_ticket_price(self, price): self.ticket_price = price
35      def set_event_type(self, e_type): self.event_type = e_type
36
37      # ----- Business Methods -----
38      def calculate_total_revenue(self): 1 usage
39          booked = self.total_seats - self.available_seats
40          return booked * self.ticket_price
41
42      def get_booked_tickets(self): 1 usage
43          return self.total_seats - self.available_seats
44
45      def book_tickets(self, num_tickets): 1 usage
46          if num_tickets <= self.available_seats:
47              self.available_seats -= num_tickets
48              print(f"\u2713 {num_tickets} ticket(s) booked successfully.")
49          else:
50              print("X Not enough tickets available.")
```

The code defines the Event class with its constructor, getters, setters, and business methods like calculate\_total\_revenue and book\_tickets.



The second screenshot shows the same PyCharm interface with the event.py file open. The code has been updated to include additional logic for booking tickets:

```
4   class Event: 2 usages
5       def __init__(self, event_name="", event_date=None, event_time=None,
6                   venue_name="", total_seats=0, available_seats=None,
7                   ticket_price=0.0, event_type="Movie"):
8           self.event_name = event_name
9           self.event_date = event_date or date.today()
10          self.event_time = event_time or time(hour=0, minute=0)
11          self.venue_name = venue_name
12          self.total_seats = total_seats
13          self.available_seats = available_seats if available_seats is not None else total_seats
14          self.ticket_price = ticket_price
15          self.event_type = event_type
16
17      # ----- Getters -----
18      def get_event_name(self): return self.event_name
19      def get_event_date(self): return self.event_date
20      def get_event_time(self): return self.event_time
21      def get_venue_name(self): return self.venue_name
22      def get_total_seats(self): return self.total_seats
23      def get_available_seats(self): return self.available_seats
24      def get_ticket_price(self): return self.ticket_price
25      def get_event_type(self): return self.event_type
26
27      # ----- Setters -----
28      def set_event_name(self, name): self.event_name = name
29      def set_event_date(self, event_date): self.event_date = event_date
30      def set_event_time(self, event_time): self.event_time = event_time
31      def set_venue_name(self, venue): self.venue_name = venue
32      def set_total_seats(self, seats): self.total_seats = seats
33      def set_available_seats(self, seats): self.available_seats = seats
34      def set_ticket_price(self, price): self.ticket_price = price
35      def set_event_type(self, e_type): self.event_type = e_type
36
37      # ----- Business Methods -----
38      def calculate_total_revenue(self): 1 usage
39          booked = self.total_seats - self.available_seats
40          return booked * self.ticket_price
41
42      def get_booked_tickets(self): 1 usage
43          return self.total_seats - self.available_seats
44
45      def book_tickets(self, num_tickets): 1 usage
46          if num_tickets <= self.available_seats:
47              self.available_seats -= num_tickets
48              print(f"\u2713 {num_tickets} ticket(s) booked successfully.")
49          else:
50              print("X Not enough tickets available.")
```

The logic for booking tickets has been added to the book\_tickets method, which subtracts the number of booked tickets from the available seats and prints a success message or an error message if there are not enough available seats.

The screenshot shows the PyCharm IDE interface. The project structure on the left includes a .venv library root, dao, entity (with event.py selected), exception, main (containing MainModule.py), util, script.py, and TicketBookingSystem.in. External Libraries and Scratches and Consoles are also listed. The code editor on the right displays event.py with the following content:

```
class Event:
    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"\n{num_tickets} ticket(s) cancelled successfully.")
        else:
            print("X Cannot cancel more than total booked tickets.")

    def display_event_details(self):
        print("\nEvent Details")
        print(f"Event Name : {self.event_name}")
        print(f"Event Date : {self.event_date}")
        print(f"Event Time : {self.event_time}")
        print(f"Venue Name : {self.venue_name}")
        print(f"Event Type : {self.event_type}")
        print(f"Total Seats : {self.total_seats}")
        print(f"Available Seats : {self.available_seats}")
        print(f"Ticket Price (₹) : {self.ticket_price:.2f}")
```

The status bar at the bottom indicates the file is Python 3.11 (PyCharmProject). The output tab below the code editor shows the execution results.

output :

The output window displays the following information:

```
Event Details
Event Name      : IPL Finals
Event Date     : 2025-07-25
Event Time     : 20:00:00
Venue Name     : Wankhede Stadium
Event Type     : Sports
Total Seats    : 200
Available Seats: 200
Ticket Price (₹) : 1500.00
✓ 10 ticket(s) booked successfully.
☒ 3 ticket(s) cancelled successfully.
☒ Total Booked Tickets: 7
₹ Total Revenue       : ₹10500.0

Process finished with exit code 0
```

## 2. Venue Class

- Attributes:

- venue\_name, ○ address
- **Methods and Constructors:**
  - **display\_venue\_details():** Display venue details.
  - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

Code:

```
class Venue:  
    def __init__(self, venue_name="", address=""):  
        self.venue_name = venue_name  
        self.address = address  
  
    # ----- Getter Methods -----  
    def get_venue_name(self):  
        return self.venue_name  
  
    def get_address(self):  
        return self.address  
  
    # ----- Setter Methods -----  
    def set_venue_name(self, name):  
        self.venue_name = name  
  
    def set_address(self, address):  
        self.address = address  
  
    # ----- Display Method -----  
    def display_venue_details(self):  
        print("\n🌟 Venue Details")  
        print(f"Venue Name : {self.venue_name}")  
        print(f"Address : {self.address}")
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'PyCharmMiscProject [TicketBookingSystem]'. The 'entity' folder contains 'event.py' and 'venue.py', which is currently selected. The right panel shows the code for 'venue.py':

```
# entity/venue.py
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 get_address(self):
        return self.address
    def set_venue_name(self, name):
        self.venue_name = name
    def set_address(self, address):
        self.address = address
    def display_venue_details(self):
        print("\n\tVenue Details")
        print(f"Venue Name : {self.venue_name}")
        print(f"Address     : {self.address}")
```

Output :

The screenshot shows the 'Run' tab in PyCharm. It lists the run configuration 'script' and the current active configuration 'MainModule'. The output window shows the execution of 'MainModule.py' and its output:

```
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\main>MainModule.py
Venue Details
Venue Name : Wankhede Stadium
Address     : Mumbai, Maharashtra
Process finished with exit code 0
```

### 3. Customer Class

- **Attributes:**
  - customer\_name, ○ email, ○ phone\_number,
- **Methods and Constructors:**
  - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
  - **display\_customer\_details():** Display customer details.

Code :

```
# entity/customer.py
```

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

```
self.customer_name = customer_name
self.email = email
self.phone_number = phone_number

# ----- Getter Methods -----
def get_customer_name(self):
    return self.customer_name

def get_email(self):
    return self.email

def get_phone_number(self):
    return self.phone_number

# ----- Setter Methods -----
def set_customer_name(self, name):
    self.customer_name = name

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

def set_phone_number(self, phone_number):
    self.phone_number = phone_number

# ----- Display Method -----
def display_customer_details(self):
    print("\n👤 Customer Details")
    print(f"Name : {self.customer_name}")
    print(f"Email : {self.email}")
    print(f"Phone Number: {self.phone_number}")
```

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top indicates "TicketBookingSystem". Below it, the "customer.py" file is selected in the "entity" folder. The code editor displays the following Python class definition:

```
# entity/customer.py

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 get_email(self):
        return self.email
    def get_phone_number(self):
        return self.phone_number
    def set_customer_name(self, name):
        self.customer_name = name
    def set_email(self, email):
        self.email = email
    def set_phone_number(self, phone_number):
        self.phone_number = phone_number
    def display_customer_details(self):
        print("\nCustomer Details")
        print(f"Name : {self.customer_name}")
        print(f"Email : {self.email}")
        print(f"Phone Number: {self.phone_number}")

The status bar at the bottom shows "No problems in customer.py", "25:1 CRLF UTF-8 4 spaces", and "Python 3.11 (PyCharmMiscProject)".
```

output:

The screenshot shows the PyCharm "Run" tab. The "MainModule" configuration is selected. The output window displays the following text:

```
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\main\MainModule.py
Customer Details
Name : Rohit Sharma
Email : rohit@example.com
Phone Number: 9990001234
Process finished with exit code 0
```

#### 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 Constructors:**
  - **calculate\_booking\_cost(num\_tickets):** Calculate and set the total cost of the booking.
  - **book\_tickets(num\_tickets):** Book a specified number of tickets for an event.
  - **cancel\_booking(num\_tickets):** Cancel the booking and update the available seats.
  - **getAvailableNoOfTickets():** return the total available tickets
  - **getEventDetails():** return event details from the event class

code:

```
# entity/booking.py
```

```
class Booking:  
    def __init__(self, event):  
        self.event = event # reference to Event object  
        self.total_cost = 0.0  
        self.num_tickets = 0  
  
    def calculate_booking_cost(self, num_tickets):  
        self.total_cost = num_tickets * self.event.ticket_price  
        return self.total_cost  
  
    def book_tickets(self, num_tickets):  
        if num_tickets <= self.event.available_seats:  
            self.num_tickets += num_tickets  
            self.event.book_tickets(num_tickets)  
            self.calculate_booking_cost(num_tickets)  
            print(f"✅ Booking successful. Total Cost: ₹{self.total_cost:.2f}")  
        else:  
            print("❌ Not enough tickets available to book.")  
  
    def cancel_booking(self, num_tickets):  
        if num_tickets <= self.num_tickets:  
            self.num_tickets -= num_tickets  
            self.event.cancel_booking(num_tickets)  
            self.total_cost = self.num_tickets * self.event.ticket_price  
            print(f"🔴 {num_tickets} ticket(s) cancelled.")  
        else:  
            print("❌ You can't cancel more tickets than booked.")  
  
    def get_available_no_of_tickets(self):  
        return self.event.available_seats  
  
    def get_event_details(self):  
        self.event.display_event_details()
```

The screenshot shows the PyCharm IDE interface with the project 'TicketBookingSystem' open. The 'booking.py' file is the active editor. The code defines a class 'Booking' with methods for initializing an event, calculating booking cost, booking tickets, canceling bookings, and getting available seats and event details.

```
2     class Booking:
3         def __init__(self, event):
4             self.event = event
5             self.total_cost = 0.0
6             self.num_tickets = 0
7
8         def calculate_booking_cost(self, num_tickets): 1 usage
9             self.total_cost = num_tickets * self.event.ticket_price
10            return self.total_cost
11
12         def book_tickets(self, num_tickets): 1 usage (1 dynamic)
13             if num_tickets <= self.event.available_seats:
14                 self.num_tickets += num_tickets
15                 self.event.book_tickets(num_tickets)
16                 self.calculate_booking_cost(num_tickets)
17                 print(f"Booking successful. Total Cost: ₹{self.total_cost:.2f}")
18             else:
19                 print("Not enough tickets available to book.")
20
21         def cancel_booking(self, num_tickets): 1 usage (1 dynamic)
22             if num_tickets <= self.num_tickets:
23                 self.num_tickets -= num_tickets
24                 self.event.cancel_booking(num_tickets)
25                 self.total_cost = self.num_tickets * self.event.ticket_price
26                 print(f" {num_tickets} ticket(s) cancelled.")
27             else:
28                 print("You can't cancel more tickets than booked.")
29
30         def get_available_no_of_tickets(self):
31             return self.event.available_seats
32
33         def get_event_details(self):
34             self.event.display_event_details()
```

### Output :

The screenshot shows the PyCharm Run tab with the script 'MainModule' selected. The output window displays the execution of 'MainModule.py'. It shows event details (Event Name: India vs Australia, Event Date: 2025-11-10, Event Time: 18:00:00, Venue Name: Eden Gardens, Event Type: Sports, Total Seats: 50000, Available Seats: 50000, Ticket Price: ₹1200.00). It then shows the booking process: 5 ticket(s) booked successfully, resulting in a total cost of ₹6000.00. It also shows 2 ticket(s) cancelled successfully and 2 ticket(s) cancelled. Finally, it displays 49997 tickets still available.

```
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\main>MainModule.py
Event Details
Event Name      : India vs Australia
Event Date     : 2025-11-10
Event Time      : 18:00:00
Venue Name     : Eden Gardens
Event Type      : Sports
Total Seats    : 50000
Available Seats : 50000
Ticket Price (₹) : 1200.00
✓ 5 ticket(s) booked successfully.
₹ Booking successful. Total Cost: ₹6000.00
₹ 2 ticket(s) cancelled successfully.
₹ 2 ticket(s) cancelled.
₹ Tickets still available: 49997
Process finished with exit code 0
```

### Task 5: Inheritance and polymorphism

#### 1. Inheritance

- Create a subclass **Movie** that inherits from **Event**. Add the following attributes and methods:
  - Attributes:**
    - genre: Genre of the movie (e.g., Action, Comedy, Horror).
    - ActorName
    - 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.

Code :

```
# entity/movie.py
from entity.event import Event
class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name ""):
        super().__init__(event_name, event_date, event_time,
                         venue_name, total_seats, None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name
    def get_genre(self):
        return self.genre
    def get_actor_name(self):
        return self.actor_name
    def get_actress_name(self):
        return self.actress_name
    def set_genre(self, genre):
        self.genre = genre
    def set_actor_name(self, actor_name):
        self.actor_name = actor_name
    def set_actress_name(self, actress_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}")
```

The screenshot shows the PyCharm IDE interface. The project tree on the left shows a structure with 'PyCharmMiscProject' containing 'entity' and 'MainModule'. The 'entity' folder contains files like booking.py, customer.py, event.py, movie.py, and venue.py. The 'MainModule' folder contains MainModule.py and util/script.py. The 'TicketBookingSystem.iml' file is also visible. The central editor window displays the 'movie.py' file, which contains the following Python code:

```
class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 genre="", actor_name="", actress_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue_name, total_seats, available_seats, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name
    def get_genre(self):
        return self.genre
    def get_actor_name(self):
        return self.actor_name
    def get_actress_name(self):
        return self.actress_name
    def set_genre(self, genre):
        self.genre = genre
    def set_actor_name(self, actor_name):
        self.actor_name = actor_name
    def set_actress_name(self, actress_name):
        self.actress_name = actress_name
    def display_event_details(self):  # usage (1 dynamic)
        super().display_event_details()
        print("Movie Details")
        print(f"Genre : {self.genre}")
        print(f"Actor : {self.actor_name}")
        print(f"Actress : {self.actress_name}")
```

The status bar at the bottom indicates the file is 'movie.py', encoding is 'Python 3.11 (PyCharm)', and the time is 25:40.

### Output :

The screenshot shows the PyCharm terminal window titled 'script' with the command 'MainModule' run. The output displays the details of an event and a movie. The event details are:

Event Details	
Event Name	: Avengers: Secret Wars
Event Date	: 2025-12-20
Event Time	: 17:30:00
Venue Name	: PVR Cinemas, Delhi
Event Type	: Movie
Total Seats	: 150
Available Seats	: 150
Ticket Price (₹)	: 300.00

The movie details are:

Movie Details	
Genre	: Action
Actor	: Chris Evans
Actress	: Scarlett Johansson

The terminal concludes with 'Process finished with exit code 0'.

- Create another subclass **Concert** that inherits from **Event**.

- Add the following attributes and methods:
  - **Attributes:**
    1. artist: Name of the performing artist or band.
    2. type: (Theatrical, Classical, Rock, Recital)
  - **Methods:**
    1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
    2. **display\_concert\_details():** Display concert details, including the artist.

Code :

```
# entity/concert.py

from entity.event import Event

class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type ""):
        super().__init__(event_name, event_date, event_time,
                        venue_name, total_seats, None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type

    # ----- Getter Methods -----
    def get_artist(self):
        return self.artist

    def get_concert_type(self):
        return self.concert_type
    # ----- Setter Methods -----
    def set_artist(self, artist):
        self.artist = artist

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

    # ----- Display Method -----
    def display_concert_details(self):
        super().display_event_details()
        print("♪ Concert Details")
        print(f"Artist : {self.artist}")
        print(f"Concert Type : {self.concert_type}")
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'PyCharmMiscProject [TicketBookingSystem]'. The 'entity' folder contains several files: booking.py, concert.py, customer.py, event.py, movie.py, and venue.py. The 'MainModule' tab is selected in the top navigation bar. The code editor window shows the 'concert.py' file with the following content:

```
# entity/concert.py
from entity.event import Event
class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type=""):
        super().__init__(event_name, event_date, event_time,
                         venue_name, total_seats, available_seats=None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type
    def get_artist(self):
        return self.artist
    def get_concert_type(self):
        return self.concert_type
    def set_artist(self, artist):
        self.artist = artist
    def set_concert_type(self, concert_type):
        self.concert_type = concert_type
    def display_concert_details(self):
        super().display_event_details()
        print("♪ Concert Details")
        print(f"Artist : {self.artist}")
        print(f"Concert Type : {self.concert_type}")
```

The status bar at the bottom indicates the file is 'script' type, the current file is 'MainModule.py', and the Python version is 'Python 3.11 (PyCharm)'.

## Output :

The 'Run' tab shows the command line output of running 'MainModule.py'. The output displays the details of a concert event:

```
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\main>MainModule.py
Event Details
Event Name      : Coldplay World Tour
Event Date     : 2025-09-05
Event Time     : 20:00:00
Venue Name     : Bandra Kurla Complex, Mumbai
Event Type     : Concert
Total Seats    : 30000
Available Seats : 30000
Ticket Price (₹) : 3500.00
♪ Concert Details
Artist       : Coldplay
Concert Type : Rock

Process finished with exit code 0
```

- Create another subclass **Sports** that inherits from **Event**. Add the following attributes and methods:
  - **Attributes:**
    1. sportName: Name of the game.
    2. teamsName: (India vs Pakistan)
  - **Methods:**
    1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
    2. **display\_sport\_details():** Display concert details, including the artist.

Code:

```
# entity/sports.py

from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name ""):
        super().__init__(event_name, event_date, event_time,
                         venue_name, total_seats, None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def get_sport_name(self):
        return self.sport_name

    def get_teams_name(self):
        return self.teams_name

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

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

    def display_sport_details(self):
        super().display_event_details()
        print(" ↴ Sports Details")
        print(f"Sport Name : {self.sport_name}")
        print(f"Teams : {self.teams_name}")
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'PyCharmMiscProject [TicketBookingSystem]'. The 'entity' folder contains several files: booking.py, concert.py, customer.py, event.py, movie.py, sports.py (which is currently selected), and venue.py. The 'exception' and 'main' folders also contain some files. The bottom navigation bar shows 'Run', 'script', and 'MainModule'. The status bar at the bottom indicates the file path 'PyCharmMiscProject > entity > sports.py', encoding 'UTF-8', and Python version 'Python 3.11 (PyCharm)'.

```
# entity/sports.py
from entity.event import Event
class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name=""):
        super().__init__(event_name, event_date, event_time,
                        venue_name, total_seats, available_seats=None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name
    def get_sport_name(self):
        return self.sport_name
    def get_teams_name(self):
        return self.teams_name
    def set_sport_name(self, sport_name):
        self.sport_name = sport_name
    def set_teams_name(self, teams_name):
        self.teams_name = teams_name
    def display_sport_details(self):
        super().display_event_details()
        print("FK Sports Details")
        print(f"Sport Name : {self.sport_name}")
        print(f"Teams : {self.teams_name}")
```

Output :

The screenshot shows the 'Run' tab in PyCharm. The output window displays the following text:

```
Event Details
Event Name      : Asia Cup Final
Event Date     : 2025-08-25
Event Time      : 18:00:00
Venue Name      : Narendra Modi Stadium, Ahmedabad
Event Type      : Sports
Total Seats     : 90000
Available Seats : 90000
Ticket Price (₹) : 2200.00
FK Sports Details
Sport Name : Cricket
Teams      : India vs Pakistan

Process finished with exit code 0
```

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

- **create\_event(event\_name: str, date:str, time:str, total\_seats: int, ticket\_price: float, event\_type: str, venu\_name:str):**

Create a new event with the specified details and event type (movie, sport or concert) and return event object.

- **display\_event\_details(event: Event):** Accepts an event object and calls its **display\_event\_details()** method to display event details.
- **book\_tickets(event: Event, num\_tickets: int):**
  1. Accepts an event object and the number of tickets to be booked.
  2. Checks if there are enough available seats for the booking.
  3. If seats are available, updates the available seats and returns the total cost of the booking.
  4. If seats are not available, displays a message indicating that the event is sold out.
- **cancel\_tickets(event: Event, num\_tickets):** cancel a specified number of tickets for an event.
- **main():** simulates the ticket booking system
  1. User can book tickets and view the event details as per their choice in menu (movies, sports, concerts).
  2. Display event details using the **display\_event\_details()** method without knowing the specific event type (demonstrate polymorphism).
  3. Make bookings using the **book\_tickets()** and cancel tickets **cancel\_tickets()** method.

Code :

```
# main/ticket_booking_system.py
```

```
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime
```

```
class TicketBookingSystem:
```

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

        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            return Movie(event_name, event_date, event_time, venue_name,
                        total_seats, ticket_price, "Movie", genre, actor, actress)
```

```
elif event_type.lower() == "concert":  
    artist = input("Enter artist/band name: ")  
    concert_type = input("Enter concert type (Rock, Classical, etc.): ")  
    return Concert(event_name, event_date, event_time, venue_name,  
                  total_seats, ticket_price, "Concert", artist, concert_type)  
  
elif event_type.lower() == "sports":  
    sport_name = input("Enter sport name: ")  
    teams_name = input("Enter teams (e.g., India vs Pakistan): ")  
    return Sports(event_name, event_date, event_time, venue_name,  
                  total_seats, ticket_price, "Sports", sport_name, teams_name)  
  
else:  
    print("X Invalid event type!")  
    return None  
  
def display_event_details(self, event):  
    event.display_event_details() # Polymorphism  
  
def book_tickets(self, event, num_tickets: int):  
    if num_tickets <= event.available_seats:  
        event.book_tickets(num_tickets)  
        cost = num_tickets * event.ticket_price  
        print(f"Ticket Total cost: ₹{cost}")  
        return cost  
    else:  
        print("X Event is sold out or not enough tickets.")  
        return 0  
  
def cancel_tickets(self, event, num_tickets: int):  
    event.cancel_booking(num_tickets)  
  
def main(self):  
    system = TicketBookingSystem()  
    current_event = None  
  
    while True:  
        print("\n❖ Ticket Booking Menu")  
        print("1. Create New Event")  
        print("2. View Event Details")
```

```
print("3. Book Tickets")
print("4. Cancel Tickets")
print("5. Exit")

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

if choice == "1":
    name = input("Enter event name: ")
    date_str = input("Enter event date (YYYY-MM-DD): ")
    time_str = input("Enter event time (HH:MM): ")
    venue = input("Enter venue name: ")
    seats = int(input("Enter total seats: "))
    price = float(input("Enter ticket price: "))
    type_str = input("Enter event type (Movie/Sports/Concert): ")
    current_event = system.create_event(name, date_str, time_str, seats, price, type_str, venue)

elif choice == "2":
    if current_event:
        system.display_event_details(current_event)
    else:
        print("⚠️ No event created yet.")

elif choice == "3":
    if current_event:
        num = int(input("Enter number of tickets to book: "))
        system.book_tickets(current_event, num)
    else:
        print("⚠️ No event available for booking.")

elif choice == "4":
    if current_event:
        num = int(input("Enter number of tickets to cancel: "))
        system.cancel_tickets(current_event, num)
    else:
        print("⚠️ No event available for cancellation.")

elif choice == "5":
    print("👋 Thank you for using the Ticket Booking System.")
    break

else:
```

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

The screenshot shows the PyCharm IDE interface with the project 'TicketBookingSystem' open. The code editor displays `MainModule.py` with the following content:

```
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime

class TicketBookingSystem:
    def create_event(self, event_name: str, date_str: str, time_str: str, total_seats: int, ticket_price: float, event_type: str, venue_name: str):
        event_date = datetime.strptime(date_str, "%Y-%m-%d").date()
        event_time = datetime.strptime(time_str, "%H:%M").time()
        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            return Movie(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type, genre, actor, actress)
        elif event_type.lower() == "concert":
            artist = input("Enter artist/band name: ")
            concert_type = input("Enter concert type (Rock, Classical, etc.): ")
            return Concert(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type, artist, concert_type)
        elif event_type.lower() == "sports":
            sport_name = input("Enter sport name: ")
            teams_name = input("Enter teams (e.g., India vs Pakistan): ")
            return Sports(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type, sport_name, teams_name)
        else:
            print("X Invalid choice. Please try again.")
```

The status bar at the bottom indicates the file is `ticket_booking_system.py`, the encoding is `UTF-8`, and the Python version is `Python 3.11 (PyCharm)`.

The screenshot shows the PyCharm IDE interface with the project 'TicketBookingSystem' open. The code editor displays `MainModule.py` with the following content:

```
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime

class TicketBookingSystem:
    def create_event(self, event_name: str, date_str: str, time_str: str, total_seats: int, ticket_price: float, event_type: str, venue_name: str):
        else:
            print("X Invalid event type!")
            return None
    def display_event_details(self, event):
        event.display_event_details() # Polymorphism
    def book_tickets(self, event, num_tickets: int):
        if num_tickets <= event.available_seats:
            event.book_tickets(num_tickets)
            cost = num_tickets * event.ticket_price
            print(f"Total cost: {cost}")
            return cost
        else:
            print("X Event is sold out or not enough tickets.")
            return 0
    def cancel_tickets(self, event, num_tickets: int):
        event.cancel_booking(num_tickets)
    def main(self):
        system = TicketBookingSystem()
        current_event = None
        while True:
            print("\n★ Ticket Booking Menu")
            print("1. Create New Event")
            print("2. View Event Details")
            print("3. Book Tickets")
            print("4. Cancel Tickets")
```

The status bar at the bottom indicates the file is `ticket_booking_system.py`, the encoding is `UTF-8`, and the Python version is `Python 3.11 (PyCharm)`.

The image shows two screenshots of the PyCharm IDE interface, one above the other, displaying Python code for a ticket booking system.

**Top Screenshot:** The code is incomplete, showing the start of a main menu loop. It includes options for creating events, booking tickets, canceling tickets, exiting, and viewing event details. The code uses input() for user choice and print() for output.

```
6     class TicketBookingSystem: 3 usages
44         def main(self): 1 usage
45             print("1. Create Event")
46             print("2. Book Tickets")
47             print("3. Cancel Tickets")
48             print("4. Exit")
49             choice = input("Enter your choice (1-5): ")
50
51             if choice == "1":
52                 name = input("Enter event name: ")
53                 date_str = input("Enter event date (YYYY-MM-DD): ")
54                 time_str = input("Enter event time (HH:MM): ")
55                 venue = input("Enter venue name: ")
56                 seats = int(input("Enter total seats: "))
57                 price = float(input("Enter ticket price: "))
58                 type_str = input("Enter event type (Movie/Sports/Concert): ")
59                 current_event = system.create_event(name, date_str, time_str, seats, price, type_str, venue)
60
61             elif choice == "2":
62                 if current_event:
63                     system.display_event_details(current_event)
64                 else:
65                     print("⚠️ No event created yet.")
66
67             elif choice == "3":
68                 if current_event:
69                     num = int(input("Enter number of tickets to book: "))
70                     system.book_tickets(current_event, num)
71                 else:
72                     print("⚠️ No event available for booking.")
73
74             elif choice == "4":
75                 break
76
77             else:
78                 print("⚠️ Invalid choice. Please try again.")
```

**Bottom Screenshot:** The code is complete, adding functionality to handle invalid choices and exit the program. It also includes a break statement to exit the loop when the user chooses to exit.

```
6     class TicketBookingSystem: 3 usages
44         def main(self): 1 usage
45             system.display_event_details(current_event)
46
47             else:
48                 print("⚠️ No event created yet.")
49
50             elif choice == "3":
51                 if current_event:
52                     num = int(input("Enter number of tickets to book: "))
53                     system.book_tickets(current_event, num)
54                 else:
55                     print("⚠️ No event available for booking.")
56
57             elif choice == "4":
58                 if current_event:
59                     num = int(input("Enter number of tickets to cancel: "))
60                     system.cancel_tickets(current_event, num)
61                 else:
62                     print("⚠️ No event available for cancellation.")
63
64             elif choice == "5":
65                 print("👋 Thank you for using the Ticket Booking System.")
66                 break
67
68             else:
69                 print("⚠️ Invalid choice. Please try again.")
```

Output:

```
Run MainModule x
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\main\MainModule.py

Ticket Booking Menu
1. Create New Event
2. View Event Details
3. Book Tickets
4. Cancel Tickets
5. Exit

Enter your choice (1-5): 1
Enter event name: Avengers Endgame
Enter event date (YYYY-MM-DD): 2025-12-25
Enter event time (HH:MM): 18:30
Enter venue name: PVR Mumbai
Enter total seats: 120
Enter ticket price: 300
Enter event type (Movie/Sports/Concert): Movie
Enter genre: Action
Enter actor name: Robert Downey Jr
Enter actress name: Scarlett Johansson

Ticket Booking Menu
1. Create New Event
2. View Event Details
3. Book Tickets
4. Cancel Tickets

PyCharmMiscProject > main > MainModule.py
6:1 CRLF UTF-8 4 spaces Python 3.11 (PyCharmMiscProject) ⌂
```

```
Run MainModule x
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\main\MainModule.py

Ticket Booking Menu
1. Create New Event
2. View Event Details
3. Book Tickets
4. Cancel Tickets
5. Exit

Enter your choice (1-5): 2

Event Details
Event Name      : Avengers Endgame
Event Date       : 2025-12-25
Event Time       : 18:30:00
Venue Name       : PVR Mumbai
Event Type       : Movie
Total Seats     : 120
Available Seats : 120
Ticket Price (₹) : 300.00

Movie Details
Genre      : Action
Actor     : Robert Downey Jr
Actress   : Scarlett Johansson
```

```
... Run MainModule ×  
↻ | :  
↑ ↗ Ticket Booking Menu  
↓ 1. Create New Event  
⤵ 2. View Event Details  
⤷ 3. Book Tickets  
⤸ 4. Cancel Tickets  
⤹ 5. Exit  
Delete Enter your choice (1-5): 3  
Enter number of tickets to book: 5  
✓ 5 ticket(s) booked successfully.  
₹ Total cost: ₹1500.0  
  
↻ Ticket Booking Menu  
1. Create New Event  
2. View Event Details  
3. Book Tickets  
4. Cancel Tickets  
5. Exit  
Enter your choice (1-5): 4  
Enter number of tickets to cancel: 1  
✗ 1 ticket(s) cancelled successfully.
```

```
Run MainModule ×  
↻ | :  
↑ 5. Exit  
↓ Enter your choice (1-5): 4  
⤵ Enter number of tickets to cancel: 1  
✗ 1 ticket(s) cancelled successfully.  
  
⤷ ↗ Ticket Booking Menu  
⤸ 1. Create New Event  
⤹ 2. View Event Details  
⤸ 3. Book Tickets  
⤸ 4. Cancel Tickets  
⤹ 5. Exit  
Enter your choice (1-5): 5  
👋 Thank you for using the Ticket Booking System.
```

#### 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*:

Code :

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

class Event(ABC):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type ""):
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    @abstractmethod
    def display_event_details(self):
        pass

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, num_tickets):
        pass

    @abstractmethod
    def calculate_total_revenue(self):
        pass

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

```
from abc import ABC, abstractmethod
from datetime import date, time
class Event(ABC): 6 usages
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type ""):
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type
    @abstractmethod 5 usages (2 dynamic)
    def display_event_details(self):
        pass
    @abstractmethod 2 usages (2 dynamic)
    def book_tickets(self, num_tickets):
        pass
    @abstractmethod 2 usages (2 dynamic)
    def cancel_booking(self, num_tickets):
        pass
    @abstractmethod
    def calculate_total_revenue(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.

Code :

```
from entity.event import Event
```

```
class Movie(Event):
```

```
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name ""):
        super().__init__(event_name, event_date, event_time, venue_name,
                        total_seats, None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name
```

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

```
    print("\n🎬 Movie Event Details")
    print(f"Name : {self.event_name}")
    print(f"Date : {self.event_date}")
    print(f"Time : {self.event_time}")
    print(f"Venue : {self.venue_name}")
    print(f"Genre : {self.genre}")
```

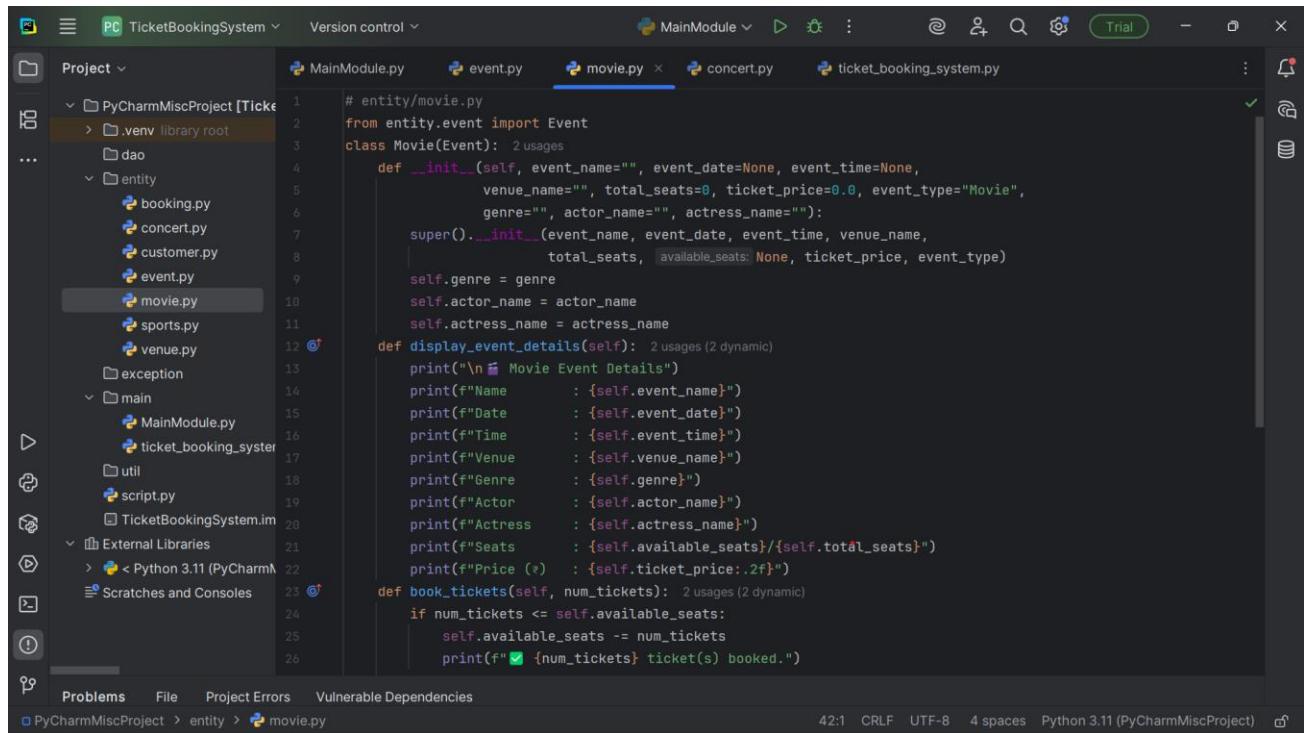
```
        print(f"Actor : {self.actor_name}")
        print(f"Actress : {self.actress_name}")
        print(f"Seats : {self.available_seats}/{self.total_seats}")
        print(f"Price (₹) : {self.ticket_price:.2f}")

    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✅ {num_tickets} ticket(s) booked.")
        else:
            print("❌ Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"🚫 {num_tickets} ticket(s) cancelled.")
        else:
            print("❌ Cannot cancel more than booked.")

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

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



```
# entity/movie.py
from entity.event import Event
class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name=""):
        super().__init__(event_name, event_date, event_time, venue_name,
                         total_seats, available_seats=None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name
    def display_event_details(self):
        print("\nMovie Event Details")
        print(f"Name : {self.event_name}")
        print(f"Date : {self.event_date}")
        print(f"Time : {self.event_time}")
        print(f"Venue : {self.venue_name}")
        print(f"Genre : {self.genre}")
        print(f"Actor : {self.actor_name}")
        print(f"Actress : {self.actress_name}")
        print(f"Seats : {self.available_seats}/{self.total_seats}")
        print(f"Price (₹) : {self.ticket_price:.2f}")
    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✅ {num_tickets} ticket(s) booked.")
```

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** PyCharmMiscProject [TicketBookingSystem]
- File:** movie.py
- Code Content:** The code defines a class `Movie` that inherits from `Event`. It contains methods for booking and canceling tickets, calculating total revenue, and getting booked tickets.

```
class Movie(Event):
    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"\u2713 {num_tickets} ticket(s) booked.")
        else:
            print("X Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"\u2713 {num_tickets} ticket(s) cancelled.")
        else:
            print("X Cannot cancel more than booked.")

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

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

- Toolbars and Status:** Shows various icons for file operations, a progress bar, and status information like "42:1 CRLF UTF-8 4 spaces Python 3.11 (PyCharmMiscProject)".

- Concert.

Code :

```
from entity.event import Event
```

```
class Concert(Event):
```

```
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type ""):
        super().__init__(event_name, event_date, event_time, venue_name,
                        total_seats, None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type
```

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

```
        print("\n♪ Concert Event Details")
        print(f"Name : {self.event_name}")
        print(f"Date : {self.event_date}")
        print(f"Time : {self.event_time}")
        print(f"Venue : {self.venue_name}")
        print(f"Artist/Band : {self.artist}")
        print(f"Concert Type : {self.concert_type}")
        print(f"Seats : {self.available_seats}/{self.total_seats}")
        print(f"Price (₹) : {self.ticket_price:.2f}")
```

```
def book_tickets(self, num_tickets):
    if num_tickets <= self.available_seats:
        self.available_seats -= num_tickets
        print(f"✓ {num_tickets} ticket(s) booked.")
    else:
        print("✗ Not enough available seats.")

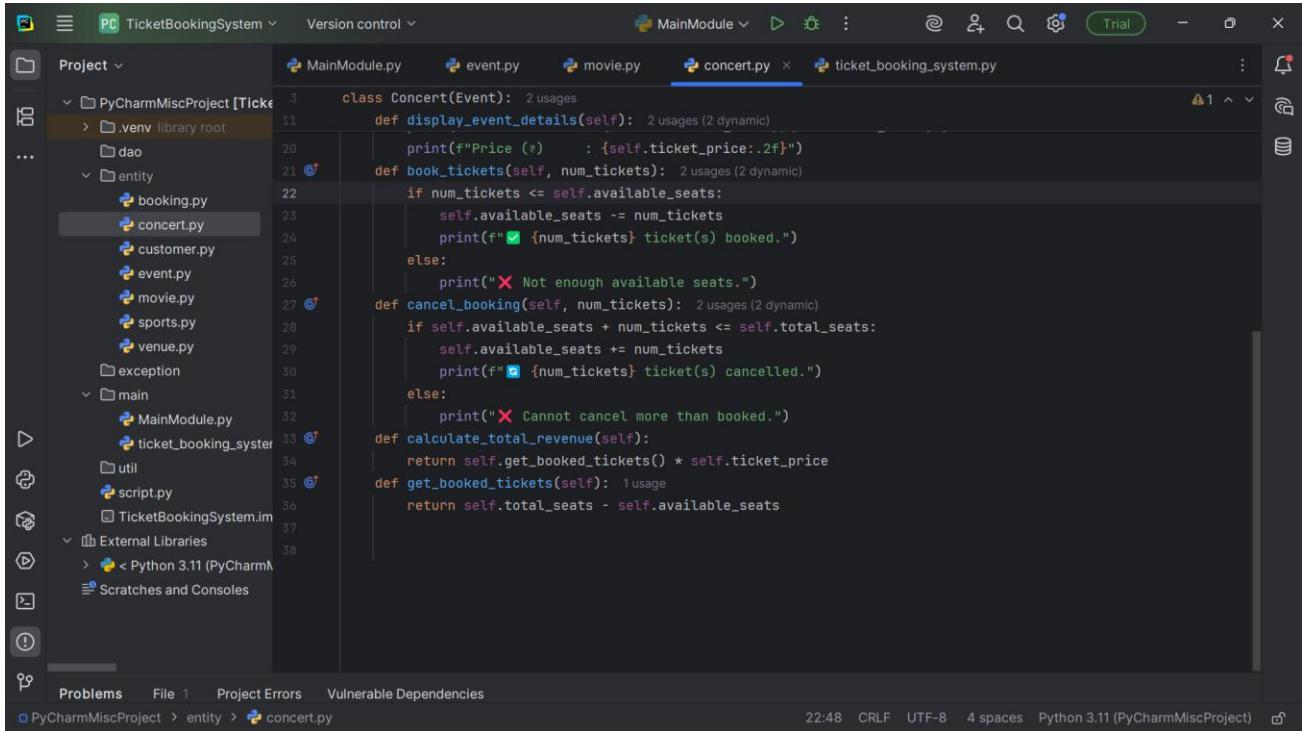
def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets <= self.total_seats:
        self.available_seats += num_tickets
        print(f"✗ {num_tickets} ticket(s) cancelled.")
    else:
        print("✗ Cannot cancel more than booked.")

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

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

The screenshot shows the PyCharm IDE interface. The project structure on the left includes a 'PyCharmMiscProject' folder containing 'entity' and 'main' packages, and an 'External Libraries' section. The 'concert.py' file is open in the main editor window, showing Python code for a Concert class. The code defines the Concert class, its \_\_init\_\_ method, display\_event\_details method, and book\_tickets method. The code uses f-strings for printing details and handles ticket booking logic. The status bar at the bottom indicates the file is 34.61 lines long, uses CRLF line endings, has 4 spaces indentation, and is written in Python 3.11 (PyCharmMiscProject).

```
# entity/concert.py
from entity.event import Event
class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type=""):
        super().__init__(event_name, event_date, event_time, venue_name,
                         total_seats, available_seats=None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type
    def display_event_details(self):
        print("\n-Concert Event Details")
        print(f"Name : {self.event_name}")
        print(f"Date : {self.event_date}")
        print(f"Time : {self.event_time}")
        print(f"Venue : {self.venue_name}")
        print(f"Artist/Band : {self.artist}")
        print(f"Concert Type : {self.concert_type}")
        print(f"Seats : {self.available_seats}/{self.total_seats}")
        print(f"Price (*) : {self.ticket_price:.2f}")
    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✓ {num_tickets} ticket(s) booked.")
        else:
            print("✗ Not enough available seats.")
```



The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "TicketBookingSystem". It contains a "PyCharmMiscProject" folder, a ".venv" library root, a "dao" directory, and a "entity" directory which contains "booking.py", "concert.py", "customer.py", "event.py", "movie.py", "sports.py", and "venue.py". There are also "exception", "main", "util", and "TicketBookingSystem.iml" files.
- Code Editor:** The file "concert.py" is open. The code defines a `Concert` class that inherits from `Event`. It includes methods for displaying event details, booking tickets (with seat validation), canceling bookings (with seat validation), and calculating total revenue.
- Status Bar:** Shows the current time as 22:48, encoding as CRLF, character set as UTF-8, 4 spaces indentation, and Python 3.11 (PyCharm) as the active interpreter.

- Sport.

Code :

```
# entity/sports.py
```

```
from entity.event import Event
```

```
class Sports(Event):  
    def __init__(self, event_name="", event_date=None, event_time=None,  
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Sports",  
                 sport_name="", teams_name=""):  
        super().__init__(event_name, event_date, event_time, venue_name,  
                        total_seats, None, ticket_price, event_type)  
        self.sport_name = sport_name  
        self.teams_name = teams_name  
  
    def display_event_details(self):  
        print("\n🌟 Sports Event Details")  
        print(f"Event Name : {self.event_name}")  
        print(f"Date : {self.event_date}")  
        print(f"Time : {self.event_time}")  
        print(f"Venue : {self.venue_name}")  
        print(f"Sport : {self.sport_name}")  
        print(f"Teams : {self.teams_name}")
```

```
print(f"Seats      : {self.available_seats}/{self.total_seats}")  
print(f"Price (₹)   : {self.ticket_price:.2f}")  
  
def book_tickets(self, num_tickets):  
    if num_tickets <= self.available_seats:  
        self.available_seats -= num_tickets  
        print(f"✓ {num_tickets} ticket(s) booked.")  
    else:  
        print("✗ Not enough available seats.")  
  
def cancel_booking(self, num_tickets):  
    if self.available_seats + num_tickets <= self.total_seats:  
        self.available_seats += num_tickets  
        print(f"✗ {num_tickets} ticket(s) cancelled.")  
    else:  
        print("✗ Cannot cancel more than booked.")  
  
def calculate_total_revenue(self):  
    return self.get_booked_tickets() * self.ticket_price  
  
def get_booked_tickets(self):  
    return self.total_seats - self.available_seats
```

The screenshot shows the PyCharm IDE interface with the project 'TicketBookingSystem' open. The 'sports.py' file is the active editor. The code defines a class 'Sports' that inherits from 'Event'. It includes methods for displaying event details and booking tickets, with a check for available seats.

```
# entity/sports.py
from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name=""):
        super().__init__(event_name, event_date, event_time, venue_name,
                         total_seats, available_seats=None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        print("\n*** Sports Event Details")
        print(f"Event Name : {self.event_name}")
        print(f"Date : {self.event_date}")
        print(f"Time : {self.event_time}")
        print(f"Venue : {self.venue_name}")
        print(f"Sport : {self.sport_name}")
        print(f"Teams : {self.teams_name}")
        print(f"Seats : {self.available_seats}/{self.total_seats}")
        print(f"Price : {self.ticket_price:.2f}")

    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✓ {num_tickets} ticket(s) booked.")
        else:
            print("✗ Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"✗ {num_tickets} ticket(s) cancelled.")
        else:
            print("✗ Cannot cancel more than booked.")

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

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

The screenshot shows the PyCharm IDE interface with the project 'TicketBookingSystem' open. The 'sports.py' file is the active editor. The code has been modified to include logic for calculating total revenue and getting booked tickets.

```
class Sports(Event):
    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✓ {num_tickets} ticket(s) booked.")
        else:
            print("✗ Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"✗ {num_tickets} ticket(s) cancelled.")
        else:
            print("✗ Cannot cancel more than booked.")

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

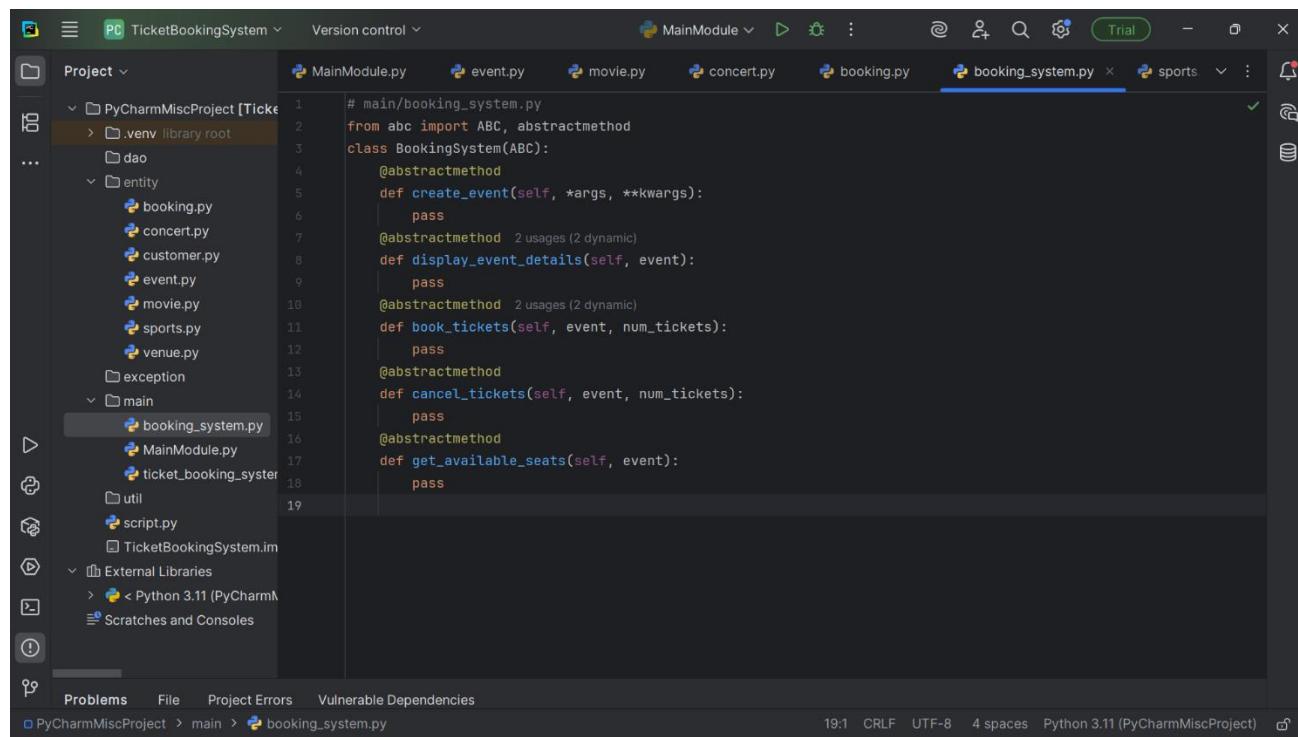
    def get_booked_tickets(self):
        return self.total_seats - self.available_seats
```

### 3. BookingSystem Abstraction:

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

Code :

```
# main/booking_system.py
from abc import ABC, abstractmethod
class BookingSystem(ABC):
    @abstractmethod
    def create_event(self, *args, **kwargs):
        pass
    @abstractmethod
    def display_event_details(self, event):
        pass
    @abstractmethod
    def book_tickets(self, event, num_tickets):
        pass
    @abstractmethod
    def cancel_tickets(self, event, num_tickets):
        pass
    @abstractmethod
    def get_available_seats(self, event):
        pass
```



4. Concrete **TicketBookingSystem** Class:

- Create a concrete class **TicketBookingSystem** that inherits from **BookingSystem**:
- **TicketBookingSystem**: Implement the abstract methods to create events, book tickets, and retrieve available seats. Maintain an array of events in this class.
- Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create\_event", "book\_tickets", "cancel\_tickets", "get\_available\_seats," and "exit."

Code :

```
# main/ticket_booking_system.py
```

```
from main.booking_system import BookingSystem
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime

class TicketBookingSystem(BookingSystem):

    def __init__(self):
        self.events = [] # Store all created events

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

        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            event = Movie(event_name, event_date, event_time, venue_name,
                          total_seats, ticket_price, "Movie", genre, actor, actress)

        elif event_type.lower() == "concert":
            artist = input("Enter artist/band name: ")
            concert_type = input("Enter concert type (Rock, Classical, etc.): ")
            event = Concert(event_name, event_date, event_time, venue_name,
                            total_seats, ticket_price, "Concert", artist, concert_type)

        elif event_type.lower() == "sports":
            sport_name = input("Enter sport name: ")
```

```
teams_name = input("Enter teams (e.g., India vs Pakistan): ")
event = Sports(event_name, event_date, event_time, venue_name,
               total_seats, ticket_price, "Sports", sport_name, teams_name)

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

self.events.append(event)
print("✓ Event created successfully!")
return event

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

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

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

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

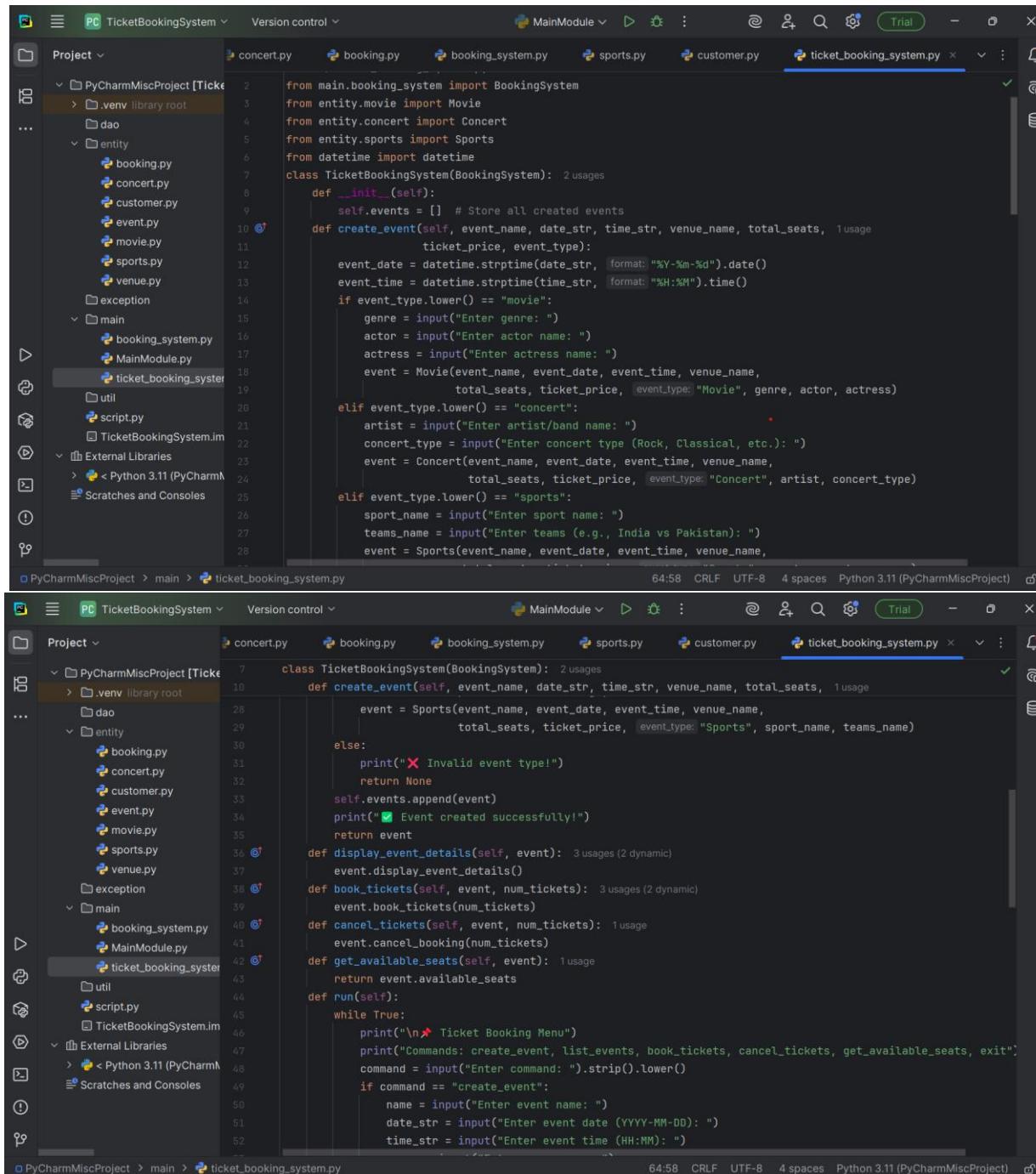
def run(self):
    while True:
        print("\n❖ Ticket Booking Menu")
        print("Commands: create_event, list_events, book_tickets, cancel_tickets,
get_available_seats, exit")
        command = input("Enter command: ").strip().lower()

        if command == "create_event":
            name = input("Enter event name: ")
            date_str = input("Enter event date (YYYY-MM-DD): ")
            time_str = input("Enter event time (HH:MM): ")
            venue = input("Enter venue name: ")
            seats = int(input("Enter total seats: "))
            price = float(input("Enter ticket price: "))
            e_type = input("Enter event type (Movie/Concert/Sports): ")

            self.create_event(name, date_str, time_str, venue, seats, price, e_type)
```

```
elif command == "list_events":  
    if not self.events:  
        print("⚠️ No events available.")  
    else:  
        for idx, event in enumerate(self.events):  
            print(f"\nEvent #{idx + 1}")  
            self.display_event_details(event)  
  
elif command == "book_tickets":  
    if not self.events:  
        print("⚠️ No events to book.")  
        continue  
    event_id = int(input("Enter event number to book tickets: ")) - 1  
    if 0 <= event_id < len(self.events):  
        count = int(input("Enter number of tickets: "))  
        self.book_tickets(self.events[event_id], count)  
    else:  
        print("❌ Invalid event number.")  
  
elif command == "cancel_tickets":  
    if not self.events:  
        print("⚠️ No events to cancel.")  
        continue  
    event_id = int(input("Enter event number to cancel tickets: ")) - 1  
    if 0 <= event_id < len(self.events):  
        count = int(input("Enter number of tickets to cancel: "))  
        self.cancel_tickets(self.events[event_id], count)  
    else:  
        print("❌ Invalid event number.")  
  
elif command == "get_available_seats":  
    if not self.events:  
        print("⚠️ No events.")  
        continue  
    event_id = int(input("Enter event number: ")) - 1  
    if 0 <= event_id < len(self.events):  
        seats = self.get_available_seats(self.events[event_id])  
        print(f"💡 Available Seats: {seats}")  
    else:  
        print("❌ Invalid event number.")
```

```
elif command == "exit":  
    print("👋 Exiting Ticket Booking System.")  
    break  
  
else:  
    print("❌ Unknown command.")
```



The image shows two screenshots of the PyCharm IDE interface, both displaying the file `ticket_booking_system.py`. The top screenshot shows the original code with a bug where the `event_type.lower()` check for "sports" is missing a colon. The bottom screenshot shows the corrected code where the colon has been added to the `if event_type.lower() == "sports":` line.

```
from main.booking_system import BookingSystem  
from entity.movie import Movie  
from entity.concert import Concert  
from entity.sports import Sports  
from datetime import datetime  
  
class TicketBookingSystem(BookingSystem): 2 usages  
    def __init__(self):  
        self.events = [] # Store all created events  
    def create_event(self, event_name, date_str, time_str, venue_name, total_seats, 1 usage  
                    ticket_price, event_type):  
        event_date = datetime.strptime(date_str, format: "%Y-%m-%d").date()  
        event_time = datetime.strptime(time_str, format: "%H:%M").time()  
        if event_type.lower() == "movie":  
            genre = input("Enter genre: ")  
            actor = input("Enter actor name: ")  
            actress = input("Enter actress name: ")  
            event = Movie(event_name, event_date, event_time, venue_name,  
                          total_seats, ticket_price, event_type: "Movie", genre, actor, actress)  
        elif event_type.lower() == "concert":  
            artist = input("Enter artist/band name: ")  
            concert_type = input("Enter concert type (Rock, Classical, etc.): ")  
            event = Concert(event_name, event_date, event_time, venue_name,  
                           total_seats, ticket_price, event_type: "Concert", artist, concert_type)  
        elif event_type.lower() == "sports":  
            sport_name = input("Enter sport name: ")  
            teams_name = input("Enter teams (e.g., India vs Pakistan): ")  
            event = Sports(event_name, event_date, event_time, venue_name,  
                           total_seats, ticket_price, event_type: "Sports", sport_name, teams_name)  
    else:  
        print("❌ Invalid event type!")  
        return None  
    self.events.append(event)  
    print("✅ Event created successfully!")  
    return event  
    def display_event_details(self, event): 3 usages (2 dynamic)  
        event.display_event_details()  
    def book_tickets(self, event, num_tickets): 3 usages (2 dynamic)  
        event.book_tickets(num_tickets)  
    def cancel_tickets(self, event, num_tickets): 1 usage  
        event.cancel_booking(num_tickets)  
    def get_available_seats(self, event): 1 usage  
        return event.available_seats  
    def run(self):  
        while True:  
            print("\n📌 Ticket Booking Menu")  
            print("Commands: create_event, list_events, book_tickets, cancel_tickets, get_available_seats, exit")  
            command = input("Enter command: ").strip().lower()  
            if command == "create_event":  
                name = input("Enter event name: ")  
                date_str = input("Enter event date (YYYY-MM-DD): ")  
                time_str = input("Enter event time (HH:MM): ")
```

```
class TicketBookingSystem(BookingSystem):    2 usages
    def run(self):
        venue = input("Enter venue name: ")
        seats = int(input("Enter total seats: "))
        price = float(input("Enter ticket price: "))
        e_type = input("Enter event type (Movie/Concert/Sports): ")
        self.create_event(name, date_str, time_str, venue, seats, price, e_type)
    elif command == "list_events":
        if not self.events:
            print("⚠️ No events available.")
        else:
            for idx, event in enumerate(self.events):
                print(f"\nEvent #{idx + 1}")
                self.display_event_details(event)
    elif command == "book_tickets":
        if not self.events:
            print("⚠️ No events to book.")
            continue
        event_id = int(input("Enter event number to book tickets: ")) - 1
        if 0 <= event_id < len(self.events):
            count = int(input("Enter number of tickets: "))
            self.book_tickets(self.events[event_id], count)
        else:
            print("❌ Invalid event number.")

    elif command == "cancel_tickets":
        if not self.events:
            print("⚠️ No events to cancel.")
            continue
        event_id = int(input("Enter event number to cancel tickets: ")) - 1
        if 0 <= event_id < len(self.events):
            count = int(input("Enter number of tickets to cancel: "))
            self.cancel_tickets(self.events[event_id], count)
        else:
            print("❌ Invalid event number.")

    elif command == "get_available_seats":
        if not self.events:
            print("⚠️ No events.")
            continue
        event_id = int(input("Enter event number: ")) - 1
        if 0 <= event_id < len(self.events):
            seats = self.get_available_seats(self.events[event_id])
            print(f"\nAvailable Seats: {seats}")
        else:
            print("❌ Invalid event number.")

    elif command == "exit":
        print("👋 Exiting Ticket Booking System.")
        break
    else:
        print("❌ Unknown command.")
```

**Output:**

```
Run MainModule x
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\main\MainModule.py
→ Ticket Booking Menu
Commands: create_event, list_events, book_tickets, cancel_tickets, get_available_seats, exit
Enter command: create_event
Enter event name: Rock Night
Enter event date (YYYY-MM-DD): 2025-07-12
Enter event time (HH:MM): 18:30
Enter venue name: Hard Rock Cafe
Enter total seats: 100
Enter ticket price: 1200
Enter event type (Movie/Concert/Sports): Concert
Enter artist/band name: Linkin Park
Enter concert type (Rock, Classical, etc.): Rock
✓ Event created successfully!

→ Ticket Booking Menu
Commands: create_event, list_events, book_tickets, cancel_tickets, get_available_seats, exit
Enter command: |
```

### Task 7: Has A Relation / Association

Create a Following classes with the following attributes and methods:

#### 1. Venue Class

- Attributes:
- venue\_name, ○ address
  - Methods and Constructors:
- **display\_venue\_details():** Display venue details.
- Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

Code :

```
# entity/venue.py
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 get_address(self):  
        return self.address  
    def set_venue_name(self, venue_name):  
        self.venue_name = venue_name  
    def set_address(self, address):  
        self.address = address  
    def display_venue_details(self):
```

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

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "TicketBookingSystem". It contains several packages and modules:
  - PyCharmMiscProject [TicketBookingSystem]:** Contains a ".venv" folder, "dao", "entity" (which includes "booking.py", "concert.py", "customer.py", "event.py", "movie.py", "sports.py", and "venue.py"), "exception", "main" (which includes "booking\_system.py", "MainModule.py", and "ticket\_booking\_system"), "util", "script.py", and "TicketBookingSystem.iml".
  - External Libraries:** Python 3.11 (PyCharm) is listed.
  - Scratches and Consoles:** An empty scratch space.
- Code Editor:** The file "venue.py" is open. The code defines a "Venue" class with an \_\_init\_\_ method, get\_venue\_name and get\_address methods, set\_venue\_name and set\_address methods, and a display\_venue\_details method that prints the venue name and address.
- Bottom Status Bar:** Shows the file path "PyCharmMiscProject > entity > venue.py", the line count "18:1", encoding "CRLF", character set "UTF-8", spaces "4 spaces", and the Python version "Python 3.11 (PyCharmMiscProject)".

## 2. Event Class:

- **Attributes:**
  - event\_name, ○ event\_date DATE, ○ event\_time TIME,
  - venue (reference of class **Venu**), ○ total\_seats, ○ available\_seats, ○ ticket\_price DECIMAL,
  - event\_type ENUM('Movie', 'Sports', 'Concert')
- **Methods and Constructors:**
  - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
  - **calculate\_total\_revenue():** Calculate and return the total revenue based on the number of tickets sold.
  - **getBookedNoOfTickets():** return the total booked tickets
  - **book\_tickets(num\_tickets):** Book a specified number of tickets for an event. Initially available seats are equal to total seats when tickets are booked available seats number should be reduced.
  - **cancel\_booking(num\_tickets):** Cancel the booking and update the available seats.
  - **display\_event\_details():** Display event details, including event name, date time seat availability.

Code :

```
# entity/event.py
```

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

```
from datetime import date, time
from entity.venue import Venue # Import the Venue class

class Event(ABC):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type ""):
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue = venue or Venue() # Venue object
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    # ----- Abstract Methods -----
    @abstractmethod
    def display_event_details(self):
        pass

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, num_tickets):
        pass

    @abstractmethod
    def calculate_total_revenue(self):
        pass

    @abstractmethod
    def get_booked_tickets(self):
        pass

    # ----- Getters -----
    def get_event_name(self):
        return self.event_name
```

```
def get_event_date(self):
    return self.event_date

def get_event_time(self):
    return self.event_time

def get_venue(self):
    return self.venue

def get_total_seats(self):
    return self.total_seats

def get_available_seats(self):
    return self.available_seats

def get_ticket_price(self):
    return self.ticket_price

def get_event_type(self):
    return self.event_type

# ----- Setters -----
def set_event_name(self, name):
    self.event_name = name

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

def set_event_time(self, event_time):
    self.event_time = event_time

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

def set_total_seats(self, seats):
    self.total_seats = seats
def set_available_seats(self, seats):
    self.available_seats = seats
def set_ticket_price(self, price):
```

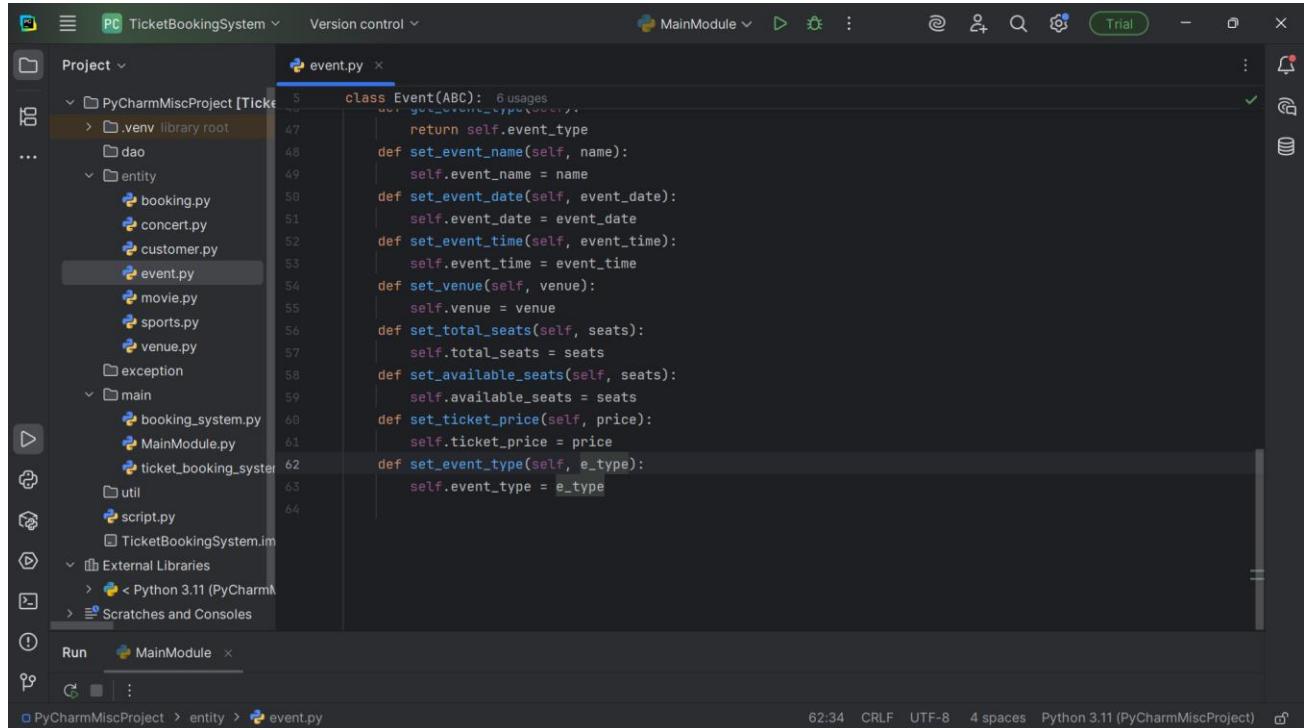
```
    self.ticket_price = price  
def set_event_type(self, e_type):  
    self.event_type = e_type
```

The screenshot shows the PyCharm interface with the project structure on the left and the code editor on the right. The code editor displays the `event.py` file, which contains the implementation of the `Event` class. The code includes methods for setting and getting event details like name, date, time, venue, total seats, available seats, ticket price, and event type. It also includes abstract methods for displaying event details, booking tickets, and canceling bookings.

```
# entity/event.py  
from abc import ABC, abstractmethod  
from datetime import date, time  
from entity.venue import Venue # Import the Venue class  
class Event(ABC): 6 usages  
    def __init__(self, event_name="", event_date=None, event_time=None,  
                 venue=None, total_seats=0, available_seats=None,  
                 ticket_price=0.0, event_type=""):  
        self.event_name = event_name  
        self.event_date = event_date or date.today()  
        self.event_time = event_time or time(hour=0, minute=0)  
        self.venue = venue or Venue() # Venue object  
        self.total_seats = total_seats  
        self.available_seats = available_seats if available_seats is not None else total_seats  
        self.ticket_price = ticket_price  
        self.event_type = event_type  
    @abstractmethod 2 usages (2 dynamic)  
    def display_event_details(self):  
        pass  
    @abstractmethod 2 usages (2 dynamic)  
    def book_tickets(self, num_tickets):  
        pass  
    @abstractmethod 2 usages (2 dynamic)  
    def cancel_booking(self, num_tickets):
```

This screenshot shows the same PyCharm interface after additional methods have been added to the `Event` class. The new methods include `calculate_total_revenue`, `get_booked_tickets`, `get_event_name`, `get_event_date`, `get_event_time`, `get_venue`, `get_total_seats`, `get_available_seats`, `get_ticket_price`, and `set_event_name`. The code editor shows the updated class definition with these new methods.

```
class Event(ABC): 6 usages  
    def calculate_total_revenue(self):  
        pass  
    @abstractmethod  
    def get_booked_tickets(self):  
        pass  
    def get_event_name(self):  
        return self.event_name  
    def get_event_date(self):  
        return self.event_date  
    def get_event_time(self):  
        return self.event_time  
    def get_venue(self):  
        return self.venue  
    def get_total_seats(self):  
        return self.total_seats  
    def get_available_seats(self):  
        return self.available_seats  
    def get_ticket_price(self):  
        return self.ticket_price  
    def get_event_type(self):  
        return self.event_type  
    def set_event_name(self, name):  
        self.event_name = name
```



```
class Event(ABC):
    def __init__(self, event_type="Event", event_name=None, event_date=None, event_time=None, venue=None, total_seats=0, ticket_price=0.0):
        self.event_type = event_type
        self.event_name = event_name
        self.event_date = event_date
        self.event_time = event_time
        self.venue = venue
        self.total_seats = total_seats
        self.available_seats = total_seats
        self.ticket_price = ticket_price

    @abstractmethod
    def set_event_name(self, name):
        pass

    @abstractmethod
    def set_event_date(self, event_date):
        pass

    @abstractmethod
    def set_event_time(self, event_time):
        pass

    @abstractmethod
    def set_venue(self, venue):
        pass

    @abstractmethod
    def set_total_seats(self, seats):
        pass

    @abstractmethod
    def set_available_seats(self, seats):
        pass

    @abstractmethod
    def set_ticket_price(self, price):
        pass

    @abstractmethod
    def set_event_type(self, e_type):
        pass
```

### 3. Event sub classes:

- Create three sub classes that inherit from **Event** abstract class and override abstract methods in concrete class should declare the variables as mentioned in above
- *Task 2:*
  - Movie.

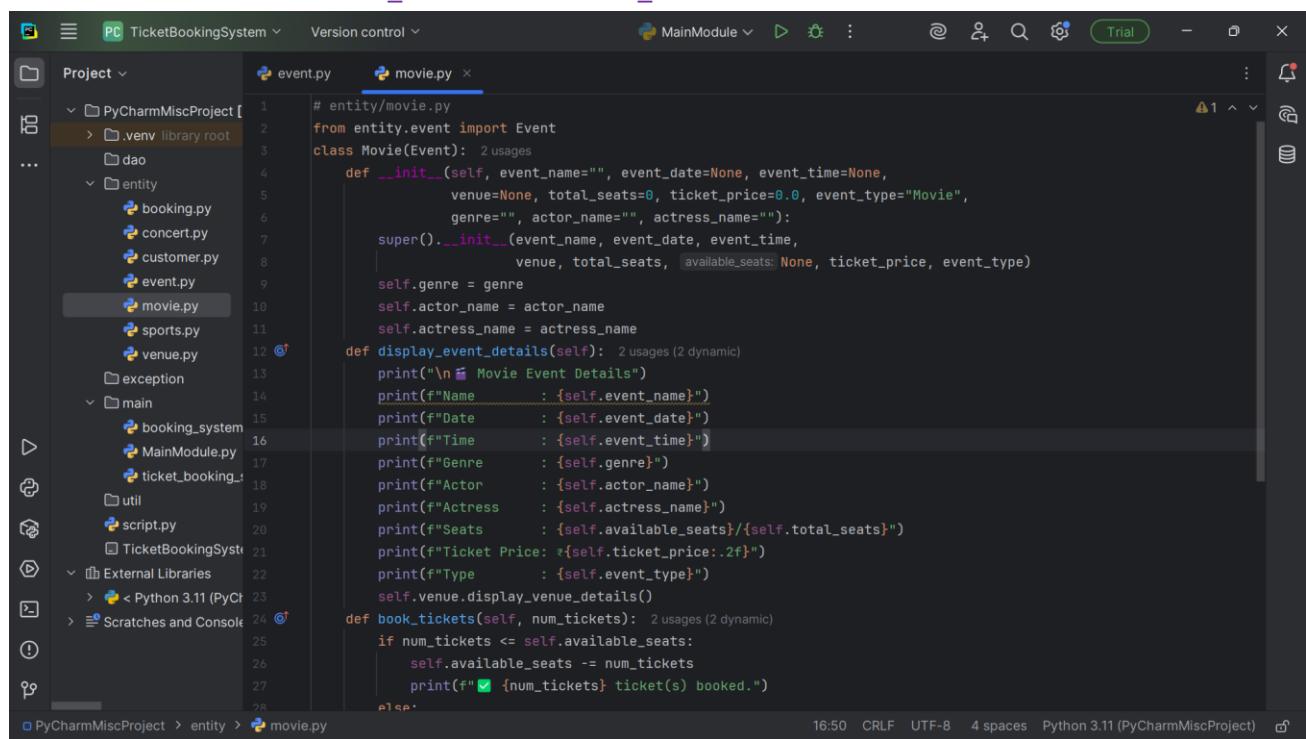
Code :

```
# entity/movie.py
from entity.event import Event

class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def display_event_details(self):
        print("\n🎬 Movie Event Details")
        print(f"Name : {self.event_name}")
        print(f"Date : {self.event_date}")
        print(f"Time : {self.event_time}")
        print(f"Genre : {self.genre}")
        print(f"Actor : {self.actor_name}")
```

```
print(f"Actress : {self.actress_name}")
print(f"Seats : {self.available_seats}/{self.total_seats}")
print(f"Ticket Price: ₹{self.ticket_price:.2f}")
print(f"Type : {self.event_type}")
self.venue.display_venue_details()
def book_tickets(self, num_tickets):
    if num_tickets <= self.available_seats:
        self.available_seats -= num_tickets
        print(f"✓ {num_tickets} ticket(s) booked.")
    else:
        print("✗ Not enough available seats.")
def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets <= self.total_seats:
        self.available_seats += num_tickets
        print(f"✗ {num_tickets} ticket(s) cancelled.")
    else:
        print("✗ Cannot cancel more than booked.")
def calculate_total_revenue(self):
    return self.get_booked_tickets() * self.ticket_price
def get_booked_tickets(self):
    return self.total_seats - self.available_seats
```



```
# entity/movie.py
from entity.event import Event
class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, available_seats=None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name
    def display_event_details(self):
        print("\nMovie Event Details")
        print(f"Name : {self.event_name}")
        print(f"Date : {self.event_date}")
        print(f"Time : {self.event_time}")
        print(f"Genre : {self.genre}")
        print(f"Actor : {self.actor_name}")
        print(f"Actress : {self.actress_name}")
        print(f"Seats : {self.available_seats}/{self.total_seats}")
        print(f"Ticket Price: ₹{self.ticket_price:.2f}")
        print(f"Type : {self.event_type}")
        self.venue.display_venue_details()
    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✓ {num_tickets} ticket(s) booked.")
        else:
```

```
class Movie(Event):
    def display_event_details(self):
        print("Seats : {self.available_seats}/{self.total_seats}")
        print(f"Ticket Price: {self.ticket_price:.2f}")
        print(f"Type : {self.event_type}")
        self.venue.display_venue_details()

    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✓ {num_tickets} ticket(s) booked.")
        else:
            print("X Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"✗ {num_tickets} ticket(s) cancelled.")
        else:
            print("✗ Cannot cancel more than booked.")

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

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

- o Concert.

Code:

```
# entity/concert.py
```

```
from entity.event import Event
```

```
class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type ""):
        super().__init__(event_name, event_date, event_time,
                        venue, total_seats, None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type
```

```
def display_event_details(self):
    print("\n🎵 Concert Event Details")
    print(f"Name : {self.event_name}")
    print(f"Date : {self.event_date}")
    print(f"Time : {self.event_time}")
    print(f"Artist : {self.artist}")
    print(f"Concert Type : {self.concert_type}")
```

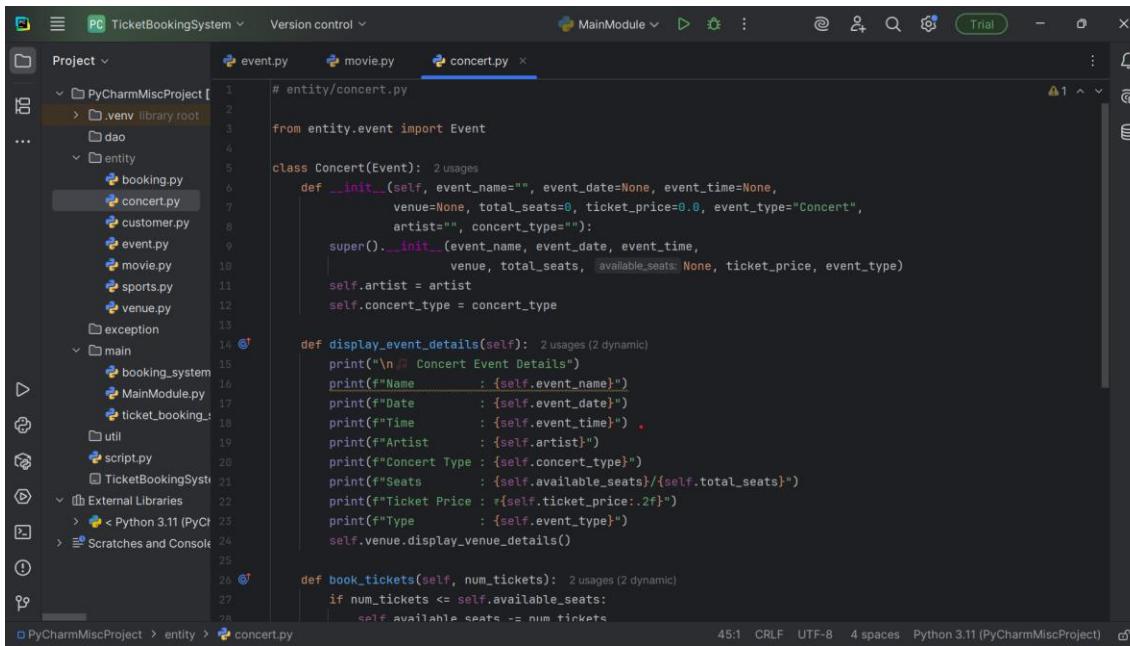
```
print(f"Seats      : {self.available_seats}/{self.total_seats}")
print(f"Ticket Price : ₹{self.ticket_price:.2f}")
print(f"Type       : {self.event_type}")
self.venue.display_venue_details()

def book_tickets(self, num_tickets):
    if num_tickets <= self.available_seats:
        self.available_seats -= num_tickets
        print(f"✓ {num_tickets} ticket(s) booked.")
    else:
        print("✗ Not enough available seats.")

def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets <= self.total_seats:
        self.available_seats += num_tickets
        print(f"✗ {num_tickets} ticket(s) cancelled.")
    else:
        print("✗ Cannot cancel more than booked.")

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

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



```
# entity/concert.py
from entity.event import Event

class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, available_seats=None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type

    def display_event_details(self):
        print("\nConcert Event Details")
        print(f"Name      : {self.event_name}")
        print(f"Date      : {self.event_date}")
        print(f"Time      : {self.event_time} .")
        print(f"Artist    : {self.artist}")
        print(f"Concert Type : {self.concert_type}")
        print(f"Seats     : {self.available_seats}/{self.total_seats}")
        print(f"Ticket Price : ₹{self.ticket_price:.2f}")
        print(f"Type      : {self.event_type}")
        self.venue.display_venue_details()

    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
```

```
class Concert(Event):
    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"\n{num_tickets} ticket(s) booked.")
        else:
            print("Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"\n{num_tickets} ticket(s) cancelled.")
        else:
            print("Cannot cancel more than booked.")

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

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

- o Sport.

Code :

```
# entity/sports.py

from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name ""):
        super().__init__(event_name, event_date, event_time,
                        venue, total_seats, None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        print("\n\n Sports Event Details")
        print(f"Name : {self.event_name}")
        print(f"Date : {self.event_date}")
        print(f"Time : {self.event_time}")
        print(f"Sport : {self.sport_name}")
        print(f"Teams : {self.teams_name}")
        print(f"Seats : {self.available_seats}/{self.total_seats}")
        print(f"Ticket Price: ₹{self.ticket_price:.2f}")
```

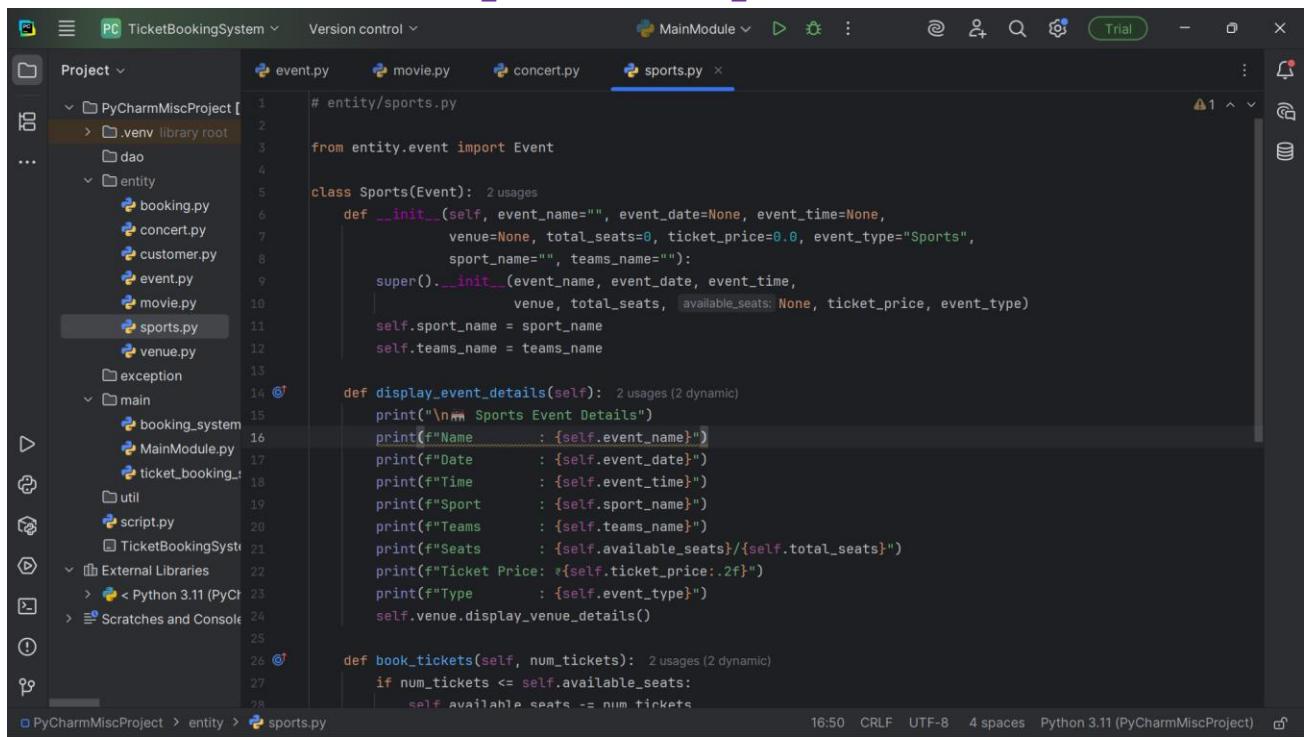
```
print(f"Type      : {self.event_type}")
self.venue.display_venue_details()

def book_tickets(self, num_tickets):
    if num_tickets <= self.available_seats:
        self.available_seats -= num_tickets
        print(f"✅ {num_tickets} ticket(s) booked.")
    else:
        print("❌ Not enough available seats.")

def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets <= self.total_seats:
        self.available_seats += num_tickets
        print(f"🚫 {num_tickets} ticket(s) cancelled.")
    else:
        print("❌ Cannot cancel more than booked.")

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

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



```
# entity/sports.py

from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, available_seats=None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        print("\nSports Event Details")
        print(f"Name      : {self.event_name}")
        print(f"Date      : {self.event_date}")
        print(f"Time      : {self.event_time}")
        print(f"Sport     : {self.sport_name}")
        print(f"Teams    : {self.teams_name}")
        print(f"Seats     : {self.available_seats}/{self.total_seats}")
        print(f"Ticket Price: {self.ticket_price:.2f}")
        print(f"Type      : {self.event_type}")
        self.venue.display_venue_details()

    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✅ {num_tickets} ticket(s) booked.")
        else:
            print("❌ Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"🚫 {num_tickets} ticket(s) cancelled.")
        else:
            print("❌ Cannot cancel more than booked.")

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

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

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** PyCharmMiscProject
- File:** sports.py
- Code Content:**

```
class Sports(Event):
    def book_tickets(self, num_tickets):
        if num_tickets < self.available_seats:
            self.available_seats -= num_tickets
            print(f"✓ {num_tickets} ticket(s) booked.")
        else:
            print("✗ Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"✗ {num_tickets} ticket(s) cancelled.")
        else:
            print("✗ Cannot cancel more than booked.")

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

    def get_booked_tickets(self):
        return self.total_seats - self.available_seats
```
- Toolbars and Status Bar:** The status bar at the bottom shows "16:50 CRLF UTF-8 4 spaces Python 3.11 (PyCharmMiscProject)".

#### 4. Customer Class

- **Attributes:**
  - customer\_name, ○ email, ○ phone\_number,
- **Methods and Constructors:**
  - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
  - **display\_customer\_details():** Display customer details.

##### Code :

```
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 get_email(self):
        return self.email
    def get_phone_number(self):
        return self.phone_number
    def set_customer_name(self, name):
        self.customer_name = name
    def set_email(self, email):
```

```
        self.email = email
def set_phone_number(self, phone_number):
    self.phone_number = phone_number
def display_customer_details(self):
    print("\n👤 Customer Details")
    print(f"Name      : {self.customer_name}")
    print(f"Email     : {self.email}")
    print(f"Phone Number : {self.phone_number}")
```

The screenshot shows the PyCharm IDE interface. The top bar displays "TicketBookingSystem" and "Version control". The main area shows the "customer.py" file under the "entity" package. The code implements a Customer class with methods for setting and getting name, email, and phone number, and a method to display customer details. The left sidebar shows the project structure with packages like "PyCharmMiscProject", "entity", "main", and "util", and files like "booking.py", "concert.py", "event.py", etc. The bottom status bar indicates the file is 25:1, uses CRLF line endings, is in UTF-8 encoding, has 4 spaces, and is using Python 3.11.

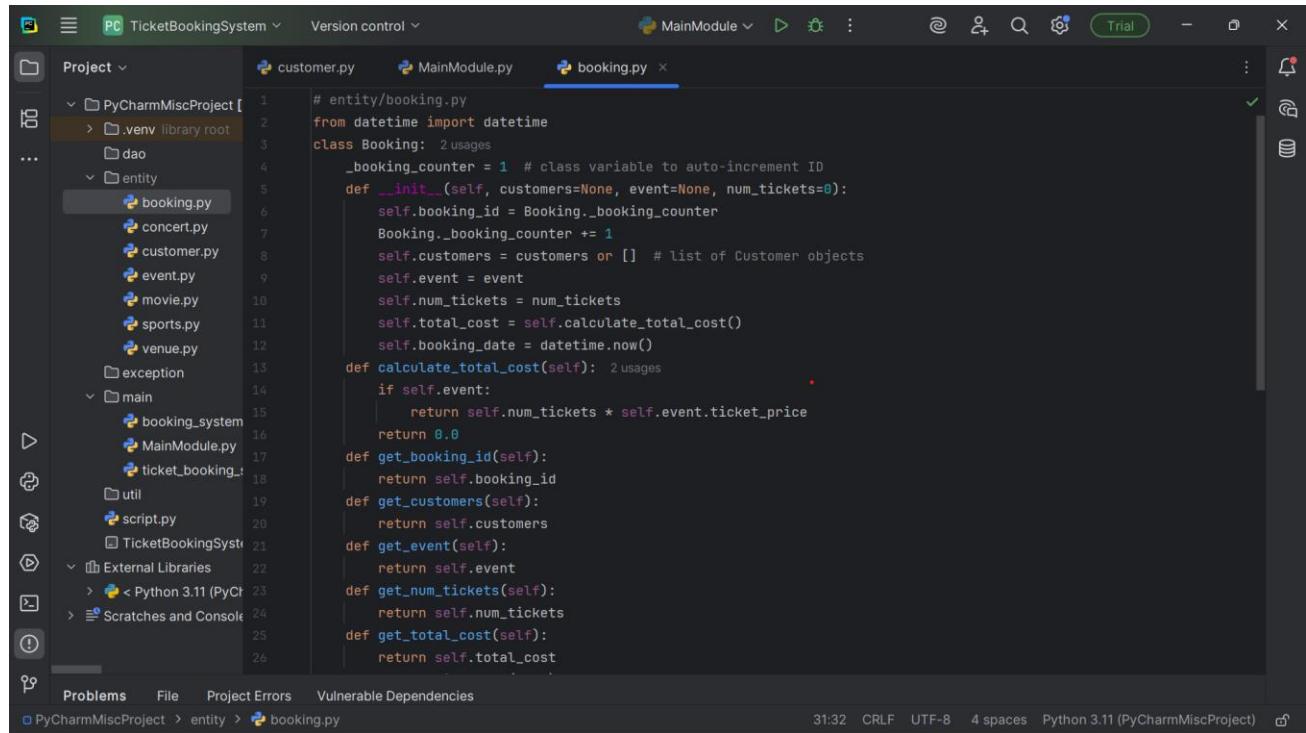
5. Create a class **Booking** with the following attributes:

- bookingId (should be incremented for each booking)
- array of customer (reference to the customer who made the booking)
- event (reference to the event booked)
- num\_tickets(no of tickets and array of customer must equal)
- total\_cost
- booking\_date (timestamp of when the booking was made)
- **Methods and Constructors:**
- Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
- **display\_booking\_details()**: Display customer details.

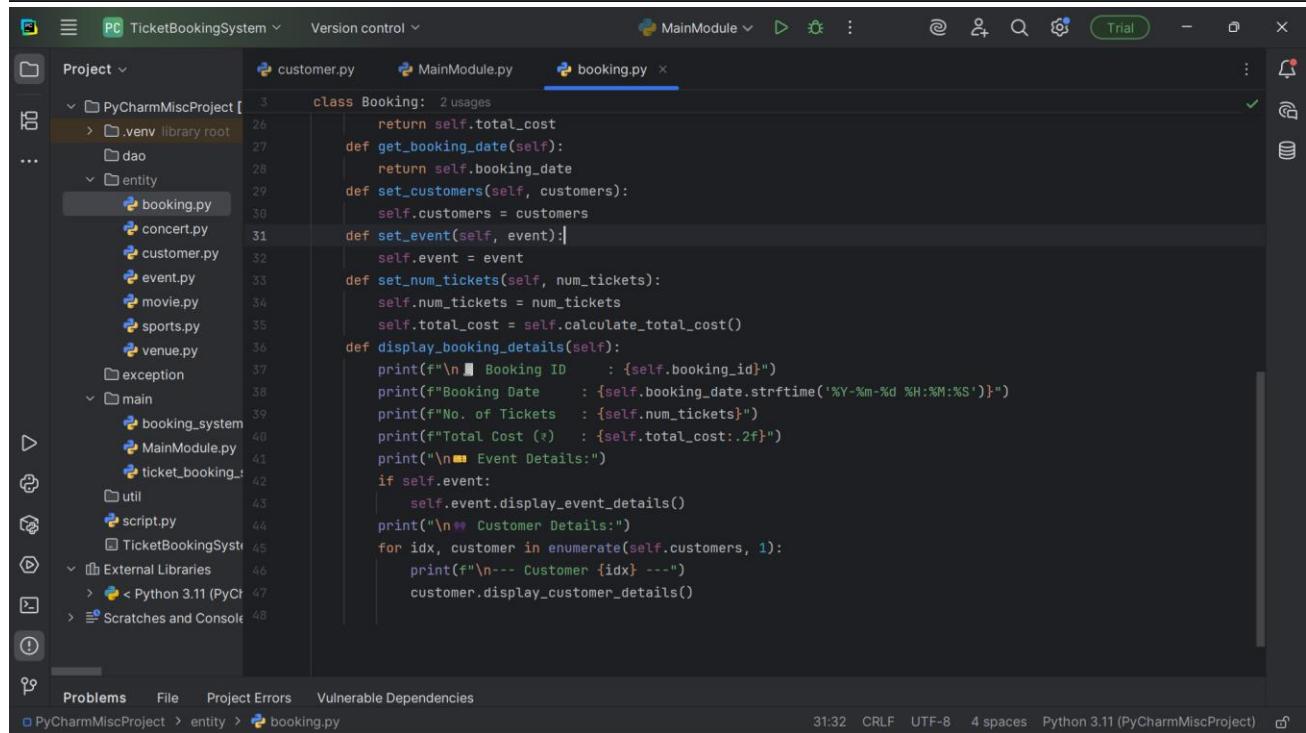
Code :

```
from datetime import datetime
class Booking:
    _booking_counter = 1 # class variable to auto-increment ID
    def __init__(self, customers=None, event=None, num_tickets=0):
        self.booking_id = Booking._booking_counter
        Booking._booking_counter += 1
        self.customers = customers or [] # list of Customer objects
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
        self.booking_date = datetime.now()
    def calculate_total_cost(self):
        if self.event:
            return self.num_tickets * self.event.ticket_price
        return 0.0
    def get_booking_id(self):
        return self.booking_id
    def get_customers(self):
        return self.customers
    def get_event(self):
        return self.event
    def get_num_tickets(self):
        return self.num_tickets
    def get_total_cost(self):
        return self.total_cost
    def get_booking_date(self):
        return self.booking_date
    def set_customers(self, customers):
        self.customers = customers
    def set_event(self, event):
        self.event = event
    def set_num_tickets(self, num_tickets):
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
    def display_booking_details(self):
        print(f"\nBooking ID : {self.booking_id}")
        print(f"Booking Date : {self.booking_date.strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"No. of Tickets : {self.num_tickets}")
        print(f"Total Cost (₹) : {self.total_cost:.2f}")
        print("\nEvent Details:")
        if self.event:
            self.event.display_event_details()
```

```
print("\n👤 Customer Details:")
for idx, customer in enumerate(self.customers, 1):
    print(f"\n--- Customer {idx} ---")
    customer.display_customer_details()
```



The screenshot shows the PyCharm IDE interface with the 'booking.py' file open in the main editor window. The code is identical to the one above. A code completion dropdown is visible over the line 'if self.event:', showing suggestions like 'display\_event\_details()', 'display\_event\_details()', and 'display\_event\_details()'. The project structure on the left shows files like 'customer.py', 'MainModule.py', and 'script.py'.



The screenshot shows the PyCharm IDE interface with the 'booking.py' file open in the main editor window. The code has been modified to include the 'display\_event\_details()' method call. A code completion dropdown is visible over the line 'if self.event:', showing suggestions like 'display\_event\_details()', 'display\_event\_details()', and 'display\_event\_details()'. The project structure on the left shows files like 'customer.py', 'MainModule.py', and 'script.py'.

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

- **Attributes** ○ array of events
- **Methods and Constructors:**
- **create\_event(event\_name: str, date:str, time:str, total\_seats: int, ticket\_price: float, event\_type: str, venu:Venu):**  
Create a new event with the specified details and event type (movie, sport or concert) and return event object.
- **calculate\_booking\_cost(num\_tickets):** Calculate and set the total cost of the booking.
- **book\_tickets(eventname:str, num\_tickets, arrayOfCustomer):** Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of **Booking** class.
- **cancel\_booking(booking\_id):** Cancel the booking and update the available seats.
- **getAvailableNoOfTickets():** return the total available tickets ○ **getEventDetails():** return event details from the event class
- Create a simple user interface in a **main method** that allows users to interact with the ticket booking system by entering commands such as "create\_event",  
"book\_tickets", "cancel\_tickets", "get\_available\_seats", "get\_event\_details," and "exit."

**Code :**

```
# main/ticket_booking_system.py
from entity.venue import Venue
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from entity.customer import Customer
from entity.booking import Booking
from datetime import datetime

class TicketBookingSystem:
    def __init__(self):
        self.events = [] # list of all events
        self.bookings = [] # list of all bookings

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

        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            event = Movie(event_name, event_date, event_time, venue_obj,
```

```
total_seats, ticket_price, "Movie", genre, actor, actress)

elif event_type.lower() == "concert":
    artist = input("Enter artist/band name: ")
    concert_type = input("Enter concert type: ")
    event = Concert(event_name, event_date, event_time, venue_obj,
                    total_seats, ticket_price, "Concert", artist, concert_type)

elif event_type.lower() == "sports":
    sport_name = input("Enter sport name: ")
    teams_name = input("Enter teams name (e.g., India vs Pakistan): ")
    event = Sports(event_name, event_date, event_time, venue_obj,
                   total_seats, ticket_price, "Sports", sport_name, teams_name)

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

self.events.append(event)
print("✓ Event created successfully!")
return event

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

def book_tickets(self, event_name, num_tickets):
    event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
    if not event:
        print("X Event not found.")
        return

    if event.available_seats < num_tickets:
        print("X Not enough tickets available.")
        return

    customers = []
    for i in range(num_tickets):
        print(f"\nEnter details for Customer {i + 1}:")
        name = input("Name: ")
        email = input("Email: ")
        phone = input("Phone: ")
        customers.append(Customer(name, email, phone))

    event.book_tickets(num_tickets)
```

```
booking = Booking(customers, event, num_tickets)
self.bookings.append(booking)
print("✅ Booking completed!")
booking.display_booking_details()

def cancel_booking(self, booking_id):
    booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
    if not booking:
        print("❌ Booking ID not found.")
        return
    booking.event.cancel_booking(booking.num_tickets)
    self.bookings.remove(booking)
    print("✅ Booking cancelled successfully.")

def get_available_no_of_tickets(self, event_name):
    event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
    if event:
        print(f"🕒 Available Seats: {event.available_seats}")
    else:
        print("❌ Event not found.")

def get_event_details(self, event_name):
    event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
    if event:
        event.display_event_details()
    else:
        print("❌ Event not found.")

def run(self):
    while True:
        print("\n❖ Ticket Booking Menu")
        print("Commands: create_event, book_tickets, cancel_tickets, get_available_seats, get_event_details, exit")
        command = input("Enter command: ").strip().lower()

        if command == "create_event":
            name = input("Enter event name: ")
            date_str = input("Enter date (YYYY-MM-DD): ")
            time_str = input("Enter time (HH:MM): ")
            venue_name = input("Enter venue name: ")
            address = input("Enter venue address: ")
            venue = Venue(venue_name, address)
            seats = int(input("Enter total seats: "))
```

```
price = float(input("Enter ticket price: "))
event_type = input("Enter event type (Movie/Concert/Sports): ")

self.create_event(name, date_str, time_str, seats, price, event_type, venue)

elif command == "book_tickets":
    name = input("Enter event name: ")
    count = int(input("Enter number of tickets: "))
    self.book_tickets(name, count)

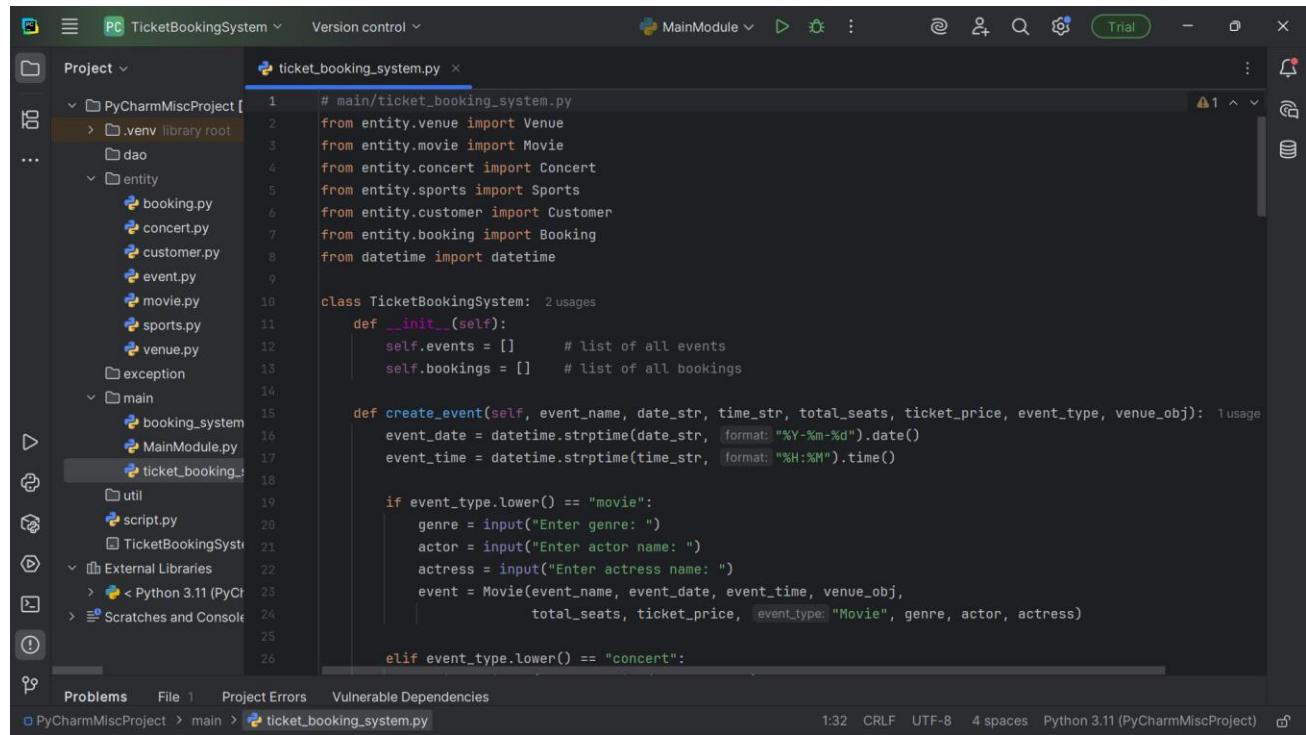
elif command == "cancel_tickets":
    bid = int(input("Enter booking ID to cancel: "))
    self.cancel_booking(bid)

elif command == "get_available_seats":
    name = input("Enter event name: ")
    self.get_available_no_of_tickets(name)

elif command == "get_event_details":
    name = input("Enter event name: ")
    self.get_event_details(name)

elif command == "exit":
    print("👋 Exiting system. Goodbye!")
    break

else:
    print("✖ Invalid command.")
```



The screenshot shows the PyCharm IDE interface with the project 'TicketBookingSystem' open. The code editor displays 'ticket\_booking\_system.py' with the following content:

```
# main/ticket_booking_system.py
from entity.venue import Venue
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from entity.customer import Customer
from entity.booking import Booking
from datetime import datetime

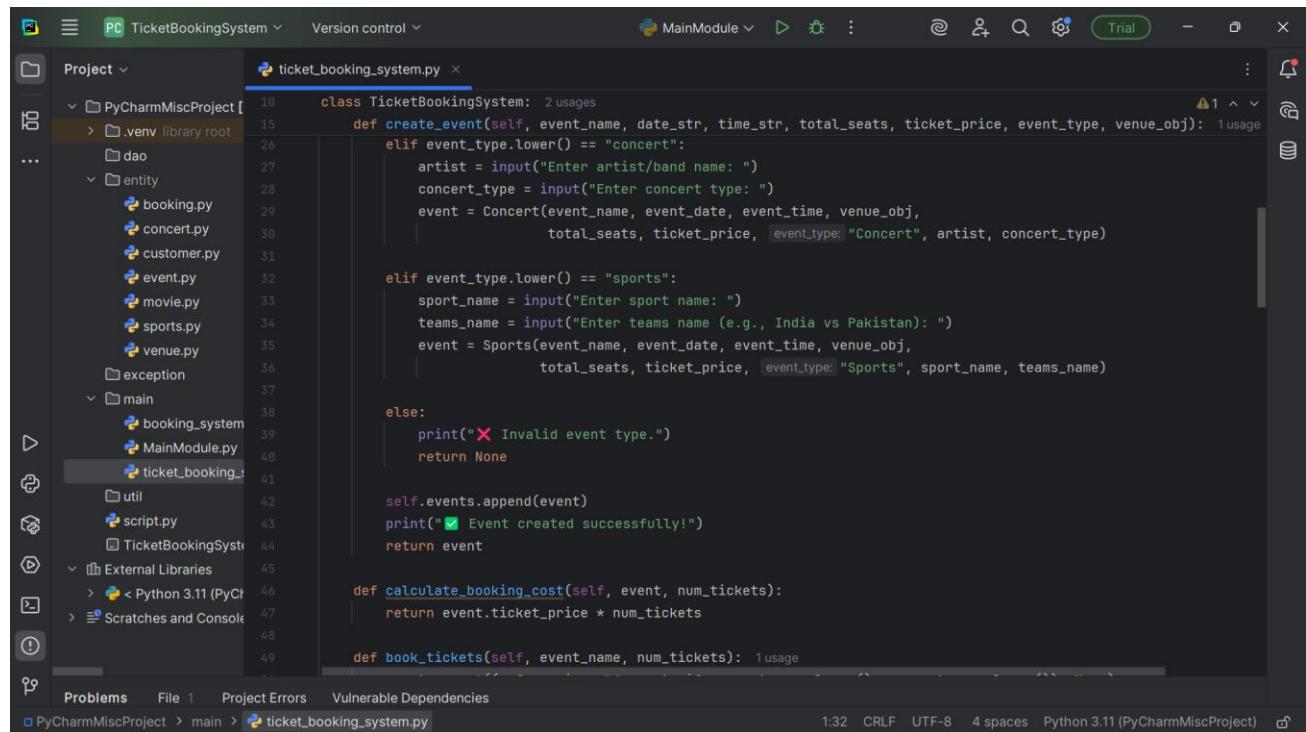
class TicketBookingSystem:
    def __init__(self):
        self.events = [] # list of all events
        self.bookings = [] # list of all bookings

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

        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            event = Movie(event_name, event_date, event_time, venue_obj,
                          total_seats, ticket_price, event_type, genre, actor, actress)

        elif event_type.lower() == "concert":
```

The code defines a `TicketBookingSystem` class with methods for creating events (specifically movies and concerts) and calculating booking costs.



The screenshot shows the PyCharm IDE interface with the project 'TicketBookingSystem' open. The code editor displays 'ticket\_booking\_system.py' with the following content:

```
class TicketBookingSystem:
    def create_event(self, event_name, date_str, time_str, total_seats, ticket_price, event_type, venue_obj):
        elif event_type.lower() == "concert":
            artist = input("Enter artist/band name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue_obj,
                            total_seats, ticket_price, event_type, artist, concert_type)

        elif event_type.lower() == "sports":
            sport_name = input("Enter sport name: ")
            teams_name = input("Enter teams name (e.g., India vs Pakistan): ")
            event = Sports(event_name, event_date, event_time, venue_obj,
                           total_seats, ticket_price, event_type, sport_name, teams_name)

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

        self.events.append(event)
        print("✓ Event created successfully!")
        return event

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

    def book_tickets(self, event_name, num_tickets):
```

The code now includes logic for creating sports events and handling invalid event types. It also includes methods for calculating booking costs and booking tickets.

```
class TicketBookingSystem:
    def book_tickets(self, event_name, num_tickets):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if not event:
            print("X Event not found.")
            return

        if event.available_seats < num_tickets:
            print("X Not enough tickets available.")
            return

        customers = []
        for i in range(num_tickets):
            print(f"\nEnter details for Customer {i + 1}:")
            name = input("Name: ")
            email = input("Email: ")
            phone = input("Phone: ")
            customers.append(Customer(name, email, phone))

        event.book_tickets(num_tickets)
        booking = Booking(customers, event, num_tickets)
        self.bookings.append(booking)
        print("✓ Booking completed!")
        booking.display_booking_details()

    def cancel_booking(self, booking_id):
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if not booking:
            print("X Booking ID not found.")
            return

        booking.event.cancel_booking(booking.num_tickets)
        self.bookings.remove(booking)
        print("✓ Booking cancelled successfully.")

    def get_available_no_of_tickets(self, event_name):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if event:
            print(f"■ Available Seats: {event.available_seats}")
        else:
            print("X Event not found.")

    def get_event_details(self, event_name):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if event:
            event.display_event_details()
        else:
            print("X Event not found.")

    def run(self):
        while True:
```

The image shows two screenshots of the PyCharm IDE interface, each displaying the file `ticket_booking_system.py`. The top screenshot shows the initial state of the code, and the bottom screenshot shows the code after modifications.

**Initial State (Top Screenshot):**

```
10     class TicketBookingSystem: 2 usages
 96         def run(self): 1 usage
 97             while True:
 98                 print("\n\t\tTicket Booking Menu")
 99                 print("Commands: create_event, book_tickets, cancel_tickets, get_available_seats, get_event_details, exit")
100                 command = input("Enter command: ").strip().lower()
101
102                 if command == "create_event":
103                     name = input("Enter event name: ")
104                     date_str = input("Enter date (YYYY-MM-DD): ")
105                     time_str = input("Enter time (HH:MM): ")
106                     venue_name = input("Enter venue name: ")
107                     address = input("Enter venue address: ")
108                     venue = Venue(venue_name, address)
109                     seats = int(input("Enter total seats: "))
110                     price = float(input("Enter ticket price: "))
111                     event_type = input("Enter event type (Movie/Concert/Sports): ")
112
113                     self.create_event(name, date_str, time_str, seats, price, event_type, venue)
114
115                 elif command == "book_tickets":
116                     name = input("Enter event name: ")
117                     count = int(input("Enter number of tickets: "))
118                     self.book_tickets(name, count)
119
120                 elif command == "cancel_tickets":
```

**Modified State (Bottom Screenshot):**

```
10     class TicketBookingSystem: 2 usages
 96         def run(self): 1 usage
 97             ...
 98
 99             elif command == "cancel_tickets":
100                 bid = int(input("Enter booking ID to cancel: "))
101                 self.cancel_booking(bid)
102
103             elif command == "get_available_seats":
104                 name = input("Enter event name: ")
105                 self.get_available_no_of_tickets(name)
106
107             elif command == "get_event_details":
108                 name = input("Enter event name: ")
109                 self.get_event_details(name)
110
111             elif command == "exit":
112                 print("Exiting system. Goodbye!")
113                 break
114
115             else:
116                 print("X Invalid command.")
```

Output :

```
Run MainModule x
Ticket Booking Menu
Commands: create_event, book_tickets, cancel_tickets, get_available_seats, get_event_details, exit
Enter command: create_event
Enter event name: Avengers Night
Enter date (YYYY-MM-DD): 2025-12-20
Enter time (HH:MM): 18:30
Enter venue name: PVR Mall
Enter venue address: Mumbai, Andheri
Enter total seats: 120
Enter ticket price: 350
Enter event type (Movie/Concert/Sports): Movie
Enter genre: Action
Enter actor name: Robert Downey Jr
Enter actress name: Scarlett Johansson
Event created successfully!
Ticket Booking Menu
Commands: create_event, book_tickets, cancel_tickets, get_available_seats, get_event_details, exit
Enter command:
```

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

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

Code :

```
# entity/venue.py
```

```
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 get_address(self):
        return self.address
    def set_venue_name(self, name):
        self.venue_name = name
    def set_address(self, address):
        self.address = address
    def display_venue_details(self):
        print("\n门票 Venue Details")
        print(f"Venue Name : {self.venue_name}")
        print(f"Address : {self.address}")
```

```
# entity/venue.py

class Venue: 6 usages
    def __init__(self, venue_name="", address=""):
        self.venue_name = venue_name
        self.address = address
    def get_venue_name(self):
        return self.venue_name
    def get_address(self):
        return self.address
    def set_venue_name(self, name):
        self.venue_name = name
    def set_address(self, address):
        self.address = address
    def display_venue_details(self): 3 usages (3 dynamic)
        print("\nVenue Details")
        print(f"Venue Name : {self.venue_name}")
        print(f"Address : {self.address}")

20:1 CRLF UTF-8 4 spaces Python 3.11 (PyCharmMiscProject)
```

## 2. Event Class:

- **Attributes:**
  - event\_name, ○ event\_date DATE, ○ event\_time TIME,
  - venue (reference of class Venu), ○ total\_seats, ○ available\_seats, ○ ticket\_price DECIMAL,
  - event\_type ENUM('Movie', 'Sports', 'Concert')
- **Methods and Constructors:**
  - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

Code :

```
# entity/event.py
```

```
from datetime import date, time
from entity.venue import Venue

class Event:
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type ""):
        self.event_name = event_name
        self.event_date = event_date or date.today()
```

```
    self.event_time = event_time or time(hour=0, minute=0)
    self.venue = venue or Venue()
    self.total_seats = total_seats
    self.available_seats = available_seats if available_seats is not None else
total_seats
    self.ticket_price = ticket_price
    self.event_type = event_type

# ----- Getters -----
def get_event_name(self):
    return self.event_name

def get_event_date(self):
    return self.event_date

def get_event_time(self):
    return self.event_time

def get_venue(self):
    return self.venue

def get_total_seats(self):
    return self.total_seats

def get_available_seats(self):
    return self.available_seats

def get_ticket_price(self):
    return self.ticket_price

def get_event_type(self):
    return self.event_type

# ----- Setters -----
def set_event_name(self, name):
    self.event_name = name

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

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

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

def set_total_seats(self, seats):
    self.total_seats = seats

def set_available_seats(self, seats):
    self.available_seats = seats

def set_ticket_price(self, price):
    self.ticket_price = price

def set_event_type(self, e_type):
    self.event_type = e_type

# ----- Display -----
def display_event_details(self):
    print("\n🕒 Event Details")
    print(f"Name : {self.event_name}")
    print(f"Date : {self.event_date}")
    print(f"Time : {self.event_time}")
    print(f"Type : {self.event_type}")
    print(f"Seats : {self.available_seats}/{self.total_seats}")
    print(f"Ticket Price: ₹{self.ticket_price:.2f}")
    self.venue.display_venue_details()
```

The image displays two screenshots of a Python project in PyCharm, illustrating the evolution of a class definition.

**Top Screenshot (Initial Implementation):**

```
# entity/event.py
from datetime import date, time
from entity.venue import Venue

class Event: 8 usages
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type ""):
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue = venue or Venue()
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    def get_event_name(self):
        return self.event_name

    def get_event_date(self):
        return self.event_date

    def get_event_time(self):
        return self.event_time

    def get_venue(self):
        return self.venue

    def get_total_seats(self):
        return self.total_seats

    def get_available_seats(self):
        return self.available_seats

    def get_ticket_price(self):
        return self.ticket_price
```

**Bottom Screenshot (Refactored Implementation):**

```
class Event: 8 usages
    def get_ticket_price(self):
        return self.ticket_price

    def get_event_type(self):
        return self.event_type

    def set_event_name(self, name):
        self.event_name = name

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

    def set_event_time(self, event_time):
        self.event_time = event_time

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

    def set_total_seats(self, seats):
        self.total_seats = seats

    def set_available_seats(self, seats):
        self.available_seats = seats

    def set_ticket_price(self, price):
        self.ticket_price = price

    def set_event_type(self, e_type):
        self.event_type = e_type

    def display_event_details(self): 1 usage (1 dynamic)
        print("\n\tEvent Details")
        print(f"\tName : {self.event_name}")
        print(f"\tDate : {self.event_date}")
        print(f"\tTime : {self.event_time}")
        print(f"\tType : {self.event_type}")
        print(f"\tSeats : {self.available_seats}/{self.total_seats}")
```

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** PyCharmMiscProject
- File:** event.py (selected in the tab bar)
- Code Content:** The code defines a class `Event` with various methods like `set_event_time`, `set_venue`, etc., and a `display_event_details` method that prints event details.
- Toolbars and Status Bar:** The top bar includes tabs for `i_Event_service.py`, `venue.py`, and `event.py`. The status bar at the bottom shows the time (56:43), encoding (CRLF), file encoding (UTF-8), spaces (4 spaces), and Python version (Python 3.11 (PyCharmMiscProject)).

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

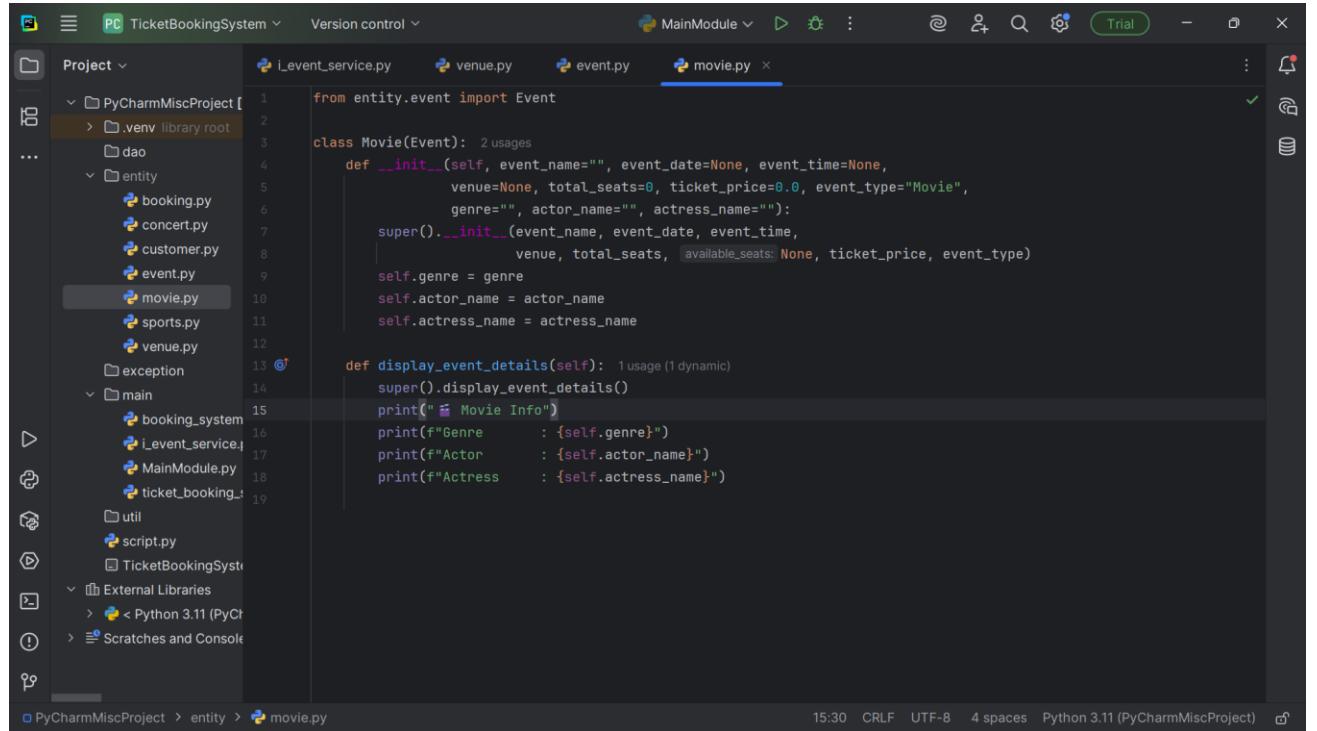
Movie :

Code :

```
from entity.event import Event
```

```
class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name ""):
        super().__init__(event_name, event_date, event_time,
                        venue, total_seats, None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

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



```
from entity.event import Event

class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name ""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

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

Sports :

Code :

```
from entity.event import Event
```

```
class Sports(Event):
```

```
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name "", teams_name ""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name
```

```
    def display_event_details(self):
        super().display_event_details()
        print("Sports Info")
        print(f"Sport : {self.sport_name}")
        print(f"Teams : {self.teams_name}")
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'PyCharmMiscProject'. The current file is 'sports.py' located in the 'entity' directory. The code defines a class 'Sports' that inherits from 'Event'. It includes an \_\_init\_\_ method and a display\_event\_details method. The right side of the screen shows the code editor with syntax highlighting. The status bar at the bottom indicates the file is saved, the time is 16:50, and the Python version is 3.11.

```
from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name ""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        super().display_event_details()
        print("Sports Info")
        print(f"Sport : {self.sport_name}")
        print(f"Teams : {self.teams_name}")
```

Concert :

Code :

```
from entity.event import Event
```

```
class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type

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

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'PyCharmMiscProject'. The current file is 'sports.py' located in the 'entity' directory. The code in 'sports.py' defines a class 'Sports' that inherits from 'Event'. It includes an \_\_init\_\_ method and a display\_event\_details method that prints sport and team information. The status bar at the bottom shows the file is saved at 16:50 with CRLF encoding, 4 spaces indentation, and Python 3.11 (PyCharmMiscProject).

```
from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name ""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, available_seats=None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

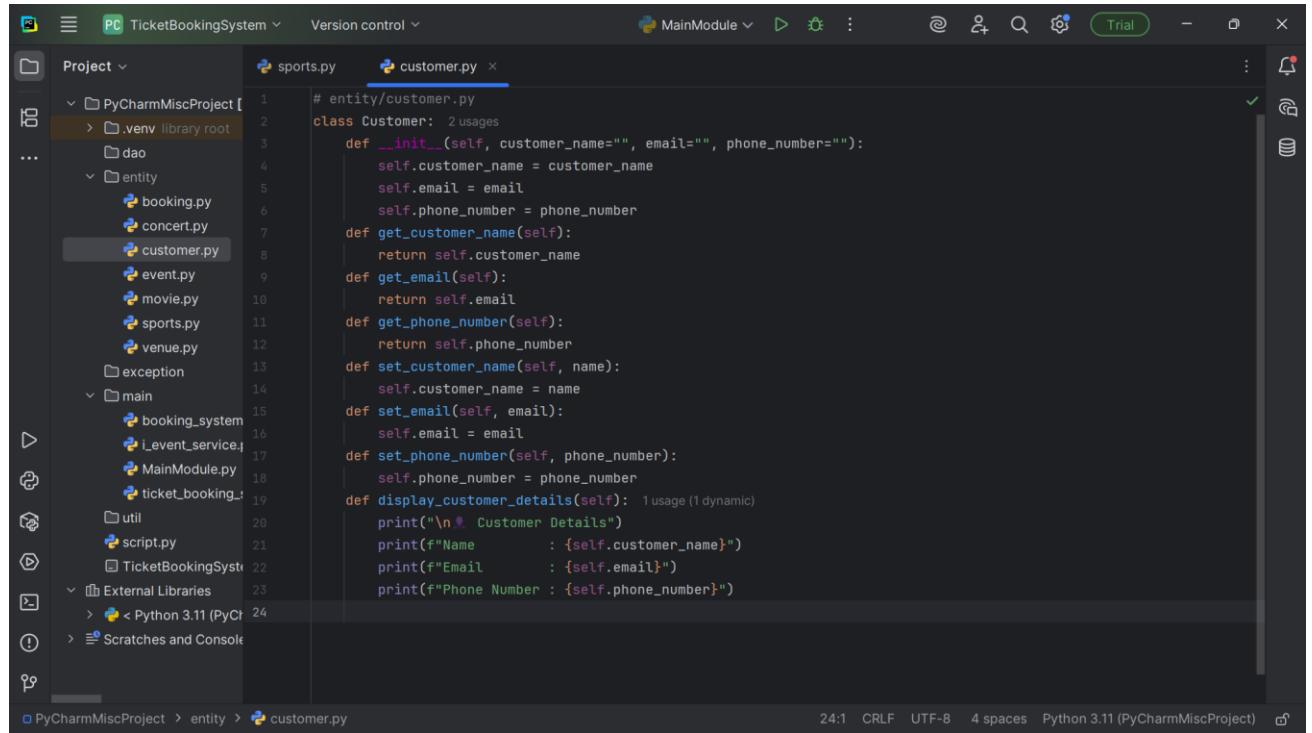
    def display_event_details(self):
        super().display_event_details()
        print("Sports Info")
        print(f"Sport : {self.sport_name}")
        print(f"Teams : {self.teams_name}")
```

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

Customer :

Code :

```
# entity/customer.py
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 get_email(self):
        return self.email
    def get_phone_number(self):
        return self.phone_number
    def set_customer_name(self, name):
        self.customer_name = name
    def set_email(self, email):
        self.email = email
    def set_phone_number(self, phone_number):
        self.phone_number = phone_number
    def display_customer_details(self):
        print("\n👤 Customer Details")
        print(f"Name : {self.customer_name}")
```



The screenshot shows the PyCharm IDE interface with the following details:

- Project:** PyCharmMiscProject
- File:** customer.py
- Code Content:** The code defines a Customer class with methods for initializing customer details, getting and setting customer name and email, and displaying customer details.

```
print(f"Email : {self.email}")
print(f"Phone Number : {self.phone_number}")

# entity/customer.py
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 get_email(self):
        return self.email
    def get_phone_number(self):
        return self.phone_number
    def set_customer_name(self, name):
        self.customer_name = name
    def set_email(self, email):
        self.email = email
    def set_phone_number(self, phone_number):
        self.phone_number = phone_number
    def display_customer_details(self):
        print("\nCustomer Details")
        print(f"Name : {self.customer_name}")
        print(f"Email : {self.email}")
        print(f"Phone Number : {self.phone_number}")
```

- Toolbars and Status Bar:** The top bar shows "TicketBookingSystem" and "Version control". The status bar at the bottom right shows "24:1 CRLF UTF-8 4 spaces Python 3.11 (PyCharmMiscProject)".

Booking :

Code :

```
# entity/booking.py
from datetime import datetime
from entity.customer import Customer
from entity.event import Event
class Booking:
    _booking_counter = 1 # Static variable
    def __init__(self, customers=None, event=None, num_tickets=0):
        self.booking_id = Booking._booking_counter
        Booking._booking_counter += 1
        self.customers = customers or []
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
        self.booking_date = datetime.now()
    def calculate_total_cost(self):
        if self.event:
            return self.num_tickets * self.event.ticket_price
        return 0.0
    def get_booking_id(self):
        return self.booking_id
```

```
def get_customers(self):
    return self.customers
def get_event(self):
    return self.event
def get_num_tickets(self):
    return self.num_tickets
def get_total_cost(self):
    return self.total_cost
def get_booking_date(self):
    return self.booking_date
def set_customers(self, customers):
    self.customers = customers
def set_event(self, event):
    self.event = event
def set_num_tickets(self, num_tickets):
    self.num_tickets = num_tickets
    self.total_cost = self.calculate_total_cost()
def display_booking_details(self):
    print(f"\n📝 Booking ID : {self.booking_id}")
    print(f"Booking Date : {self.booking_date.strftime('%Y-%m-%d %H:%M:%S')}")
    print(f"No. of Tickets : {self.num_tickets}")
    print(f"Total Cost (₹) : {self.total_cost:.2f}")
    print("\n👤 Event Info:")
    if self.event:
        self.event.display_event_details()
    print("\n👤 Customer Info:")
    for idx, customer in enumerate(self.customers, 1):
        print(f"\n--- Customer {idx} ---")
        customer.display_customer_details()
```

```
# entity/booking.py
from datetime import datetime
from entity.customer import Customer
from entity.event import Event

class Booking: 4 usages
    _booking_counter = 1 # Static variable
    def __init__(self, customers=None, event=None, num_tickets=0):
        self.booking_id = Booking._booking_counter
        Booking._booking_counter += 1
        self.customers = customers or []
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
        self.booking_date = datetime.now()
    def calculate_total_cost(self): 2 usages
        if self.event:
            return self.num_tickets * self.event.ticket_price
        return 0.0
    def get_booking_id(self):
        return self.booking_id
    def get_customers(self):
        return self.customers
    def get_event(self):
        return self.event
    def get_num_tickets(self):
        return self.num_tickets
    def get_total_cost(self):
        return self.total_cost

class Booking: 4 usages
    def __init__(self, customers=None, event=None, num_tickets=0):
        self._booking_id = self._generate_booking_id()
        self.customers = customers or []
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
    def _generate_booking_id():
        return f"BOOKING-{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
    def get_booking_id(self):
        return self._booking_id
    def get_total_cost(self):
        return self.total_cost
    def get_booking_date(self):
        return self.booking_date
    def set_customers(self, customers):
        self.customers = customers
    def set_event(self, event):
        self.event = event
    def set_num_tickets(self, num_tickets):
        self.num_tickets = num_tickets
    def display_booking_details(self): 1 usage
        print(f"\n■ Booking ID : {self._booking_id}")
        print(f"Booking Date : {self.booking_date.strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"No. of Tickets : {self.num_tickets}")
        print(f"Total Cost (₹) : {self.total_cost:.2f}")
        print("\n■ Event Info:")
        if self.event:
            self.event.display_event_details()
        print("\n■ Customer Info:")
        for idx, customer in enumerate(self.customers, 1):
            print(f"\n--- Customer {idx} ---")
            customer.display_customer_details()
```

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

- **create\_event(event\_name: str, date:str, time:str, total\_seats: int, ticket\_price: float, event\_type: str, venu: Venu):** Create a new event with the specified details and event type (movie, sport or concert) and return event object.
- **getEventDetails():** return array of event details from the event class.

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

Code :

```
# main/i_event_service.py
```

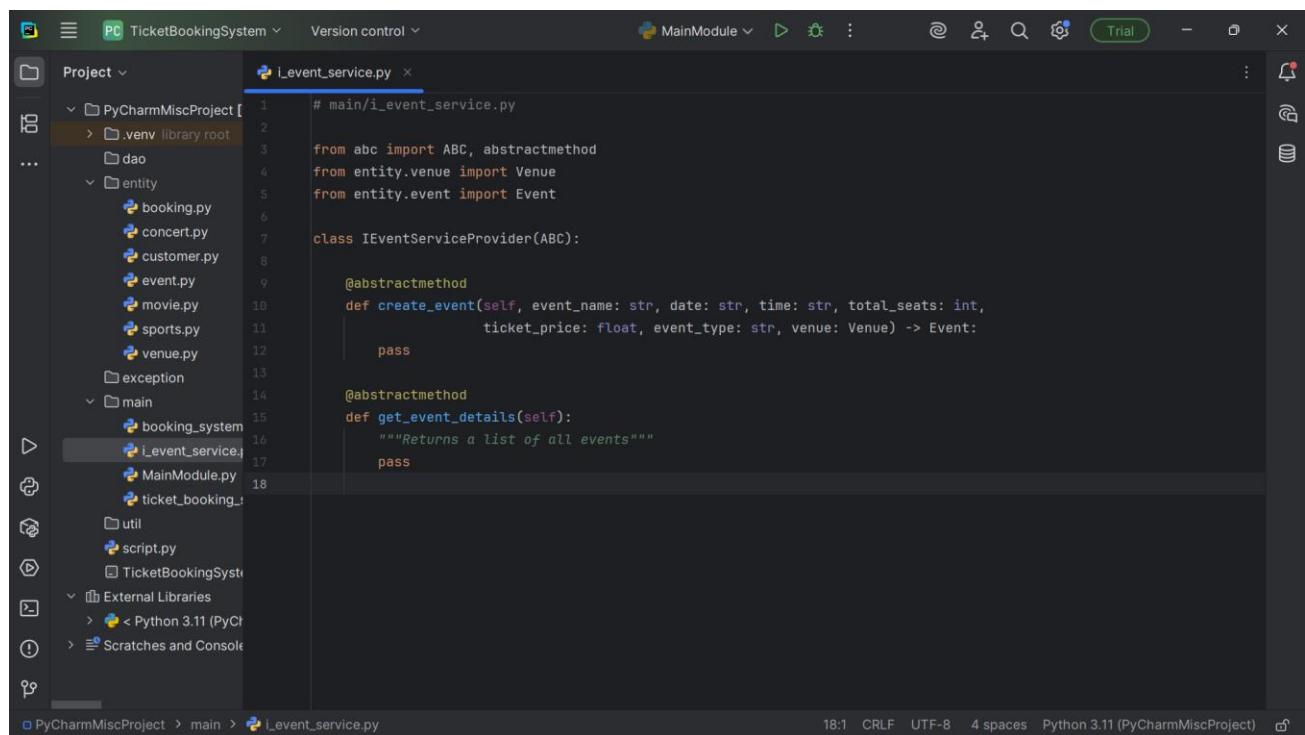
```
from abc import ABC, abstractmethod
from entity.venue import Venue
from entity.event import Event
```

```
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) -> Event:
        pass
```

```
    @abstractmethod
    def getEventDetails(self) -> list:
        pass
```

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



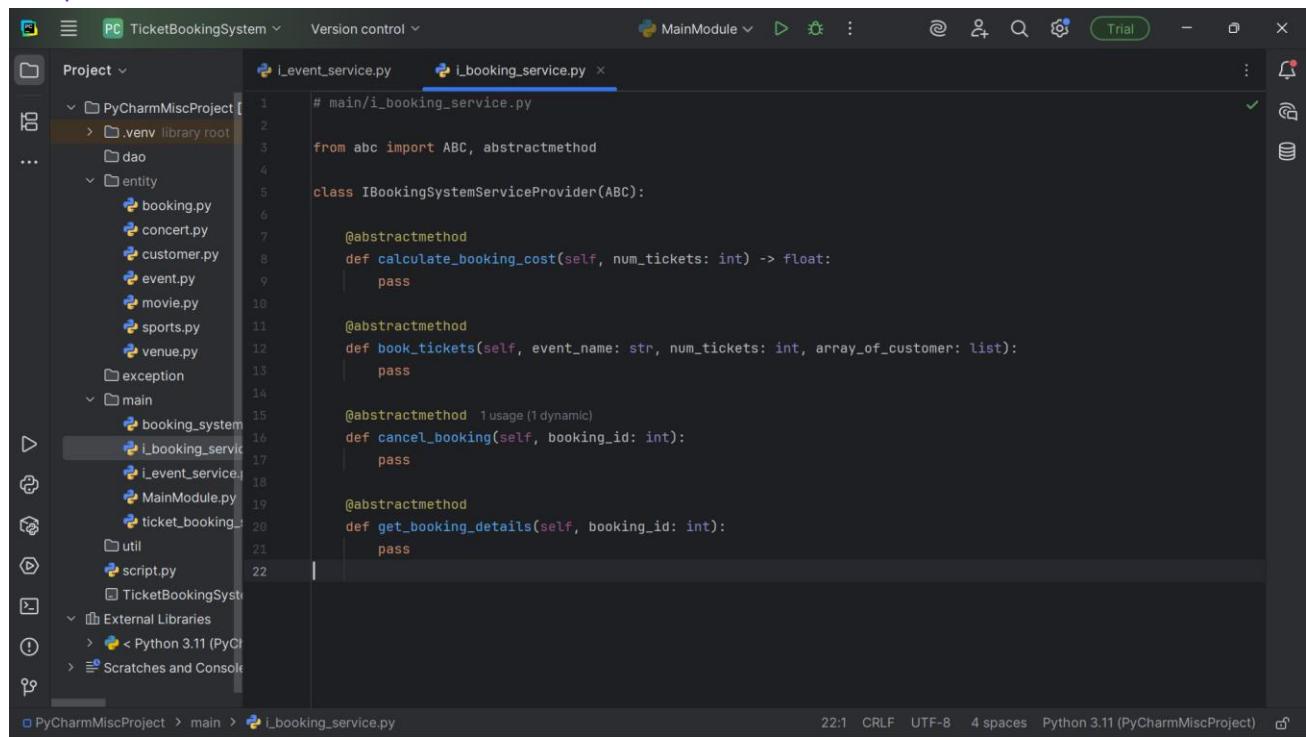
6. Create interface/abstract class **IBookingSystemServiceProvider** with following methods:

- **calculate\_booking\_cost(num\_tickets)**: Calculate and set the total cost of the booking.
- **book\_tickets(eventname:str, num\_tickets, arrayOfCustomer)**: Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class.
- **cancel\_booking(booking\_id)**: Cancel the booking and update the available seats.
- **get\_booking\_details(booking\_id)**: get the booking details.

Code :

```
# main/i_booking_service.py
```

```
from abc import ABC, abstractmethod
class IBookingSystemServiceProvider(ABC):
    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int) -> float:
        pass
    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, array_of_customer: list):
        pass
    @abstractmethod
    def cancel_booking(self, booking_id: int):
        pass
    @abstractmethod
    def get_booking_details(self, booking_id: int):
        pass
```



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

Code :

```
# main/event_service_provider_impl.py
```

```
from main.i_event_service import IEventServiceProvider
from entity.venue import Venue
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime

class EventServiceProviderImpl(IEventServiceProvider):
    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_date = datetime.strptime(date, "%Y-%m-%d").date()
        event_time = datetime.strptime(time, "%H:%M").time()

        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            event = Movie(event_name, event_date, event_time, venue,
                          total_seats, ticket_price, "Movie",
                          genre, actor, actress)

        elif event_type.lower() == "concert":
            artist = input("Enter artist/band name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue,
                            total_seats, ticket_price, "Concert",
                            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,
```

```
        total_seats, ticket_price, "Sports",
        sport_name, teams_name)

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

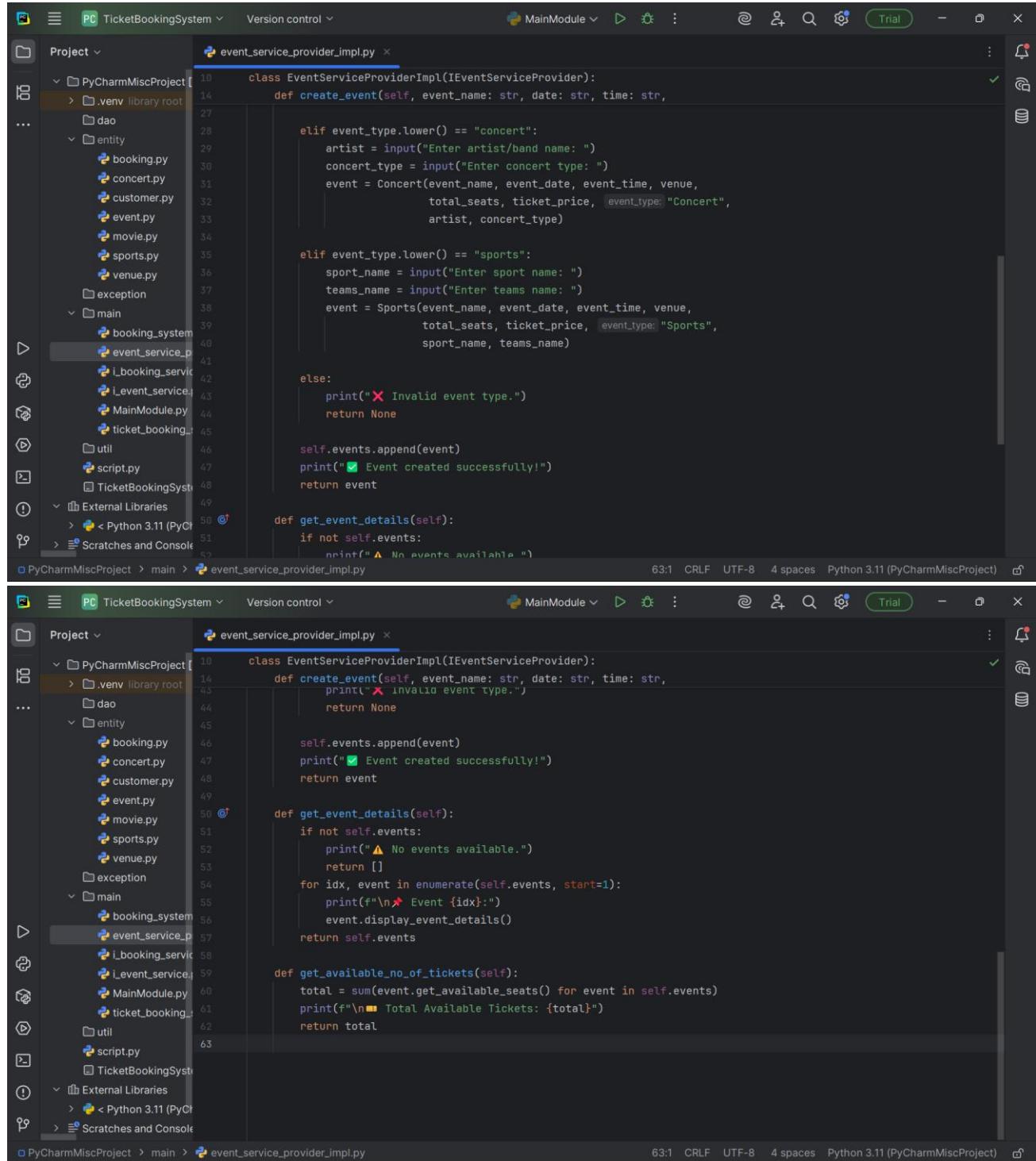
    self.events.append(event)
    print("✓ Event created successfully!")
    return event

def get_event_details(self):
    if not self.events:
        print("⚠ No events available.")
        return []
    for idx, event in enumerate(self.events, start=1):
        print(f"\n❖ Event {idx}:")
        event.display_event_details()
    return self.events

def get_available_no_of_tickets(self):
    total = sum(event.get_available_seats() for event in self.events)
    print(f"\n☒ Total Available Tickets: {total}")
    return total
```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "PyCharlMiscProject". It contains several packages and modules: .venv, library root, dao, entity (booking.py, concert.py, customer.py, event.py, movie.py, sports.py, venue.py), exception, main (booking\_system, event\_service\_provider.py, i\_booking\_service.py, L\_Event\_Service.py, MainModule.py, ticket\_booking\_system.py), util, script.py, and TicketBookingSystem.
- Code Editor:** The file "event\_service\_provider\_impl.py" is open. The code defines a class `EventServiceProviderImpl` that implements the `IEventServiceProvider` interface. It includes a `create_event` method that prompts the user for event details and creates instances of `Movie`, `Concert`, or `Sports` objects accordingly.
- Status Bar:** The status bar at the bottom indicates the file is 631 lines long, uses CRLF line endings, is in UTF-8 encoding, has 4 spaces indentation, and is using Python 3.11 (PyCharmMiscProject).



```
event_service_provider_implementation.py (Original)
class EventServiceProviderImpl(IEventServiceProvider):
    def create_event(self, event_name: str, date: str, time: str,
                     total_seats, ticket_price, event_type: str,
                     artist, concert_type):
        if event_type.lower() == "concert":
            artist = input("Enter artist/band name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue,
                            total_seats, ticket_price, event_type, "Concert",
                            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,
                           total_seats, ticket_price, event_type, "Sports",
                           sport_name, teams_name)

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

        self.events.append(event)
        print("✓ Event created successfully!")
        return event

    def get_event_details(self):
        if not self.events:
            print("▲ No events available.")

event_service_provider_implementation.py (Modified)
class EventServiceProviderImpl(IEventServiceProvider):
    def create_event(self, event_name: str, date: str, time: str,
                     total_seats, ticket_price, event_type: str,
                     artist, concert_type):
        if event_type.lower() == "concert":
            artist = input("Enter artist/band name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue,
                            total_seats, ticket_price, event_type, "Concert",
                            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,
                           total_seats, ticket_price, event_type, "Sports",
                           sport_name, teams_name)

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

        self.events.append(event)
        print("✓ Event created successfully!")
        return event

    def get_event_details(self):
        if not self.events:
            print("▲ No events available.")
            return []

        for idx, event in enumerate(self.events, start=1):
            print(f"\n▶ Event {idx}:")
            event.display_event_details()
        return self.events

    def get_available_no_of_tickets(self):
        total = sum(event.get_available_seats() for event in self.events)
        print(f"\n■ Total Available Tickets: {total}")
        return total
```

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

- **Attributes** ○ array of events

**Code :**

```
# main/booking_system_service_provider_impl.py
```

```
from main.i_booking_service import IBookingSystemServiceProvider
from main.event_service_provider_impl import EventServiceProviderImpl
from entity.booking import Booking
from entity.customer import Customer

class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__() # Initializes self.events from parent
        self.bookings = [] # Booking list

    def calculate_booking_cost(self, num_tickets: int) -> float:
        # Assumes one event context; usually handled during booking
        print(f"🔢 Cost for {num_tickets} tickets is calculated based on event during booking.")
        return 0.0

    def book_tickets(self, event_name: str, num_tickets: int, customers: list):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if not event:
            print("❌ Event not found.")
            return

        if event.available_seats < num_tickets:
            print("❌ Not enough tickets available.")
            return

        if len(customers) != num_tickets:
            print("❌ Number of customers must match number of tickets.")
            return

        event.book_tickets(num_tickets)
        booking = Booking(customers, event, num_tickets)
        self.bookings.append(booking)
        print("✅ Booking successful!")
        booking.display_booking_details()

    def cancel_booking(self, booking_id: int):
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if not booking:
            print("❌ Booking not found.")
            return
```

```
booking.event.cancel_booking(booking.num_tickets)
self.bookings.remove(booking)
print("✅ Booking cancelled successfully.")

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

The screenshot displays two code editors in PyCharm, showing the implementation and interface for a booking system provider.

**Left Editor (Implementation):**

```
# main/booking_system_service_provider_impl.py
from main.i_booking_service import IBookingSystemServiceProvider
from main.event_service_provider_impl import EventServiceProviderImpl
from entity.booking import Booking
class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__()
        self.bookings = [] # Booking list
    def calculate_booking_cost(self, num_tickets: int) -> float:
        # Assumes one event context; usually handled during booking
        print(f"💡 Cost for {num_tickets} tickets is calculated based on event during booking.")
        return 0.0
    def book_tickets(self, event_name: str, num_tickets: int, customers: list):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if not event:
            print("❌ Event not found.")
            return
        if event.available_seats < num_tickets:
            print("❌ Not enough tickets available.")
            return
        if len(customers) != num_tickets:
            print("❌ Number of customers must match number of tickets.")
            return
        event.book_tickets(num_tickets)
        booking = Booking(customers, event, num_tickets)
        self.bookings.append(booking)
        print("✅ Booking successful!")
        booking.display_booking_details()
    def cancel_booking(self, booking_id: int): 2 usages (2 dynamic)
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if not booking:
            print("❌ Booking not found.")
            return
        booking.event.cancel_booking(booking.num_tickets)
        self.bookings.remove(booking)
        print("✅ Booking cancelled successfully.")

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

**Right Editor (Interface):**

```
booking_system_service_provider_impl.py
MainModule v Trial
Project
PyCharmMiscProject [TicketBooki
... library root
entity
    booking.py
    concert.py
    customer.py
    event.py
    movie.py
    sports.py
    venue.py
exception
main
    booking_system.py
    booking_system_service_pro
    event_service_provider_impl
    i_booking_service.py
    l_event_service.py
    MainModule.py
    ticket_booking_system.py
util
script.py
TicketBookingSystem.Impl
External Libraries
Python 3.11 (PyCharmMiscProj
PyCharmMiscProject > main > booking_system_service_provider_impl.py
9:1 CRLF UTF-8 4 spaces Python 3.11 (PyCharmMiscProject)
```

The interface file contains the same code as the implementation file, indicating that the implementation follows the interface definition.

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."

Code :

```
from main.booking_system_service_provider_impl import BookingSystemServiceProviderImpl
from entity.venue import Venue
from entity.customer import Customer

class TicketBookingSystem:
    def __init__(self):
        self.system = BookingSystemServiceProviderImpl()

    def run(self):
        print("\n\t\t Welcome to the Ticket Booking System \t\t")
        while True:
            print("\n\t\t 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. get_booking_details")
            print("7. exit")

        command = input("\nEnter command: ").strip().lower()

        if command == "create_event":
            name = input("Enter event name: ")
            date = input("Enter event date (YYYY-MM-DD): ")
            time = input("Enter event time (HH:MM): ")
            venue_name = input("Enter venue name: ")
            address = input("Enter venue address: ")
            venue = Venue(venue_name, address)
            total_seats = int(input("Enter total seats: "))
            price = float(input("Enter ticket price: "))
            event_type = input("Enter event type (Movie/Concert/Sports): ")
            self.system.create_event(name, date, time, total_seats, price, event_type, venue)
```

```
elif command == "book_tickets":  
    event_name = input("Enter event name to book: ")  
    num = int(input("Enter number of tickets: "))  
    customers = []  
    for i in range(num):  
        print(f"\nCustomer {i+1}:")  
        name = input(" Name: ")  
        email = input(" Email: ")  
        phone = input(" Phone: ")  
        customers.append(Customer(name, email, phone))  
    self.system.book_tickets(event_name, num, customers)  
  
elif command == "cancel_tickets":  
    booking_id = int(input("Enter booking ID to cancel: "))  
    self.system.cancel_booking(booking_id)  
  
elif command == "get_available_seats":  
    self.system.get_available_no_of_tickets()  
  
elif command == "get_event_details":  
    self.system.get_event_details()  
  
elif command == "get_booking_details":  
    booking_id = int(input("Enter booking ID: "))  
    self.system.get_booking_details(booking_id)  
  
elif command == "exit":  
    print("👋 Exiting Ticket Booking System. Goodbye!")  
    break  
  
else:  
    print("❌ Invalid command. Try again.")
```

The screenshot shows two instances of the PyCharm IDE side-by-side, both displaying the same Python script for a ticket booking system. The project structure on the left of both editors is identical, showing a main directory 'TicketBookingSystem' containing a '.venv' folder, a 'dao' folder, an 'entity' folder with files like 'booking.py', 'concert.py', etc., and a 'main' folder with files like 'booking\_system.py', 'MainModule.py', and 'ticket\_booking\_system.py'. The 'ticket\_booking\_system.py' file is the active file in both editors.

**Code Content (ticket\_booking\_system.py):**

```
from main.booking_system_service_provider_implementation import BookingSystemServiceProviderImpl
from entity.venue import Venue
from entity.customer import Customer

class TicketBookingSystem:
    def __init__(self):
        self.system = BookingSystemServiceProviderImpl()

    def run(self):
        print("\nWelcome to the Ticket Booking System")
        while True:
            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. get_booking_details")
            print("7. exit")

            command = input("\nEnter command: ").strip().lower()

            if command == "create_event":
                name = input("Enter event name: ")
                date = input("Enter event date (YYYY-MM-DD): ")
                time = input("Enter event time (HH:MM): ")
                venue_name = input("Enter venue name: ")
                address = input("Enter venue address: ")

                self.system.create_event(name, date, time, total_seats, price, event_type, venue)

            elif command == "book_tickets":
                event_name = input("Enter event name to book: ")
                num = int(input("Enter number of tickets: "))
                customers = []
                for i in range(num):
                    print(f"\nCustomer {i+1}:")
                    name = input(" Name: ")
                    email = input(" Email: ")
                    phone = input(" Phone: ")
                    customers.append(Customer(name, email, phone))

                self.system.book_tickets(event_name, num, customers)

            elif command == "cancel_tickets":
                booking_id = int(input("Enter booking ID to cancel: "))
                self.system.cancel_booking(booking_id)

            elif command == "get_available_seats":
                seats = self.system.get_available_no_of_tickets()
                print(seats)
```

The screenshot shows the PyCharm IDE interface with the 'TicketBookingSystem' project open. The left sidebar displays the project structure, including sub-directories like 'PyCharmMiscProject [TicketBooki]', '.venv', 'entity', 'exception', 'main', 'util', and 'External Libraries'. The main editor window contains the 'ticket\_booking\_system.py' file, which defines a class 'TicketBookingSystem' with a 'run' method. This method handles various commands such as 'cancel\_tickets', 'get\_available\_seats', 'get\_event\_details', 'get\_booking\_details', and 'exit'. The code uses Python's built-in input function to interact with the user. The status bar at the bottom indicates the file is 67 lines long, uses CRLF line endings, and is encoded in UTF-8.

```
class TicketBookingSystem:
    def run(self):
        elif command == "cancel_tickets":
            booking_id = int(input("Enter booking ID to cancel: "))
            self.system.cancel_booking(booking_id)

        elif command == "get_available_seats":
            self.system.get_available_no_of_tickets()

        elif command == "get_event_details":
            self.system.get_event_details()

        elif command == "get_booking_details":
            booking_id = int(input("Enter booking ID: "))
            self.system.get_booking_details(booking_id)

        elif command == "exit":
            print("Exiting Ticket Booking System. Goodbye!")
            break

        else:
            print("X Invalid command. Try again.")
```

## Output :

The screenshot shows the 'Run' tab in PyCharm with the 'MainModule' configuration selected. The output window displays the execution of 'MainModule.py'. It starts with a welcome message and a list of available commands. The user then enters the 'create\_event' command, followed by details for a new event: name ('Avengers Assemble'), date ('2025-06-21'), time ('18:00'), venue name ('nandana halls'), venue address ('ameerpet'), total seats ('20'), ticket price ('20000'), event type ('movie'), genre ('action'), actor name ('dhamini'), and actress name ('dhamini'). The output is displayed in a monospaced font, showing the program's interaction with the user.

```
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. get_booking_details
7. exit

Enter command: create_event
Enter event name: Avengers Assemble
Enter event date (YYYY-MM-DD): 2025-06-21
Enter event time (HH:MM): 18:00
Enter venue name: nandana halls
Enter venue address: ameerpet
Enter total seats: 20
Enter ticket price: 20000
Enter event type (Movie/Concert/Sports): movie
Enter genre: action
Enter actor name: dhamini
Enter actress name: dhamini
```

The image shows two terminal windows in PyCharm, both running the same Python script. The top terminal window shows the interaction for booking tickets, and the bottom window shows the interaction for getting available seats.

**Top Terminal Window (Running MainModule.py):**

```
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. get_booking_details
7. exit

Enter command: book_tickets
Enter event name to book: Avengers Assemble
Enter number of tickets: 2

Customer 1:
Name: dhamini
Email: dhfksjgblk@gmail.com
Phone: 7569379348775

Customer 2:
Name: yamini
Email: dhfhskhgrjr@gmail.com
Phone: 8548793876758
```

**Bottom Terminal Window (Running MainModule.py):**

```
Available Commands:
1. create_event
2. book_tickets
3. cancel_tickets
4. get_available_seats
5. get_event_details
6. get_booking_details
7. exit

Enter command: get_available_seats

Total Available Tickets: 200

Available Commands:
1. create_event
2. book_tickets
3. cancel_tickets
4. get_available_seats
5. get_event_details
6. get_booking_details
7. exit

Enter command: get_event_details

Event 1.
```

The screenshot shows a PyCharm interface with a terminal window open. The terminal displays the following output:

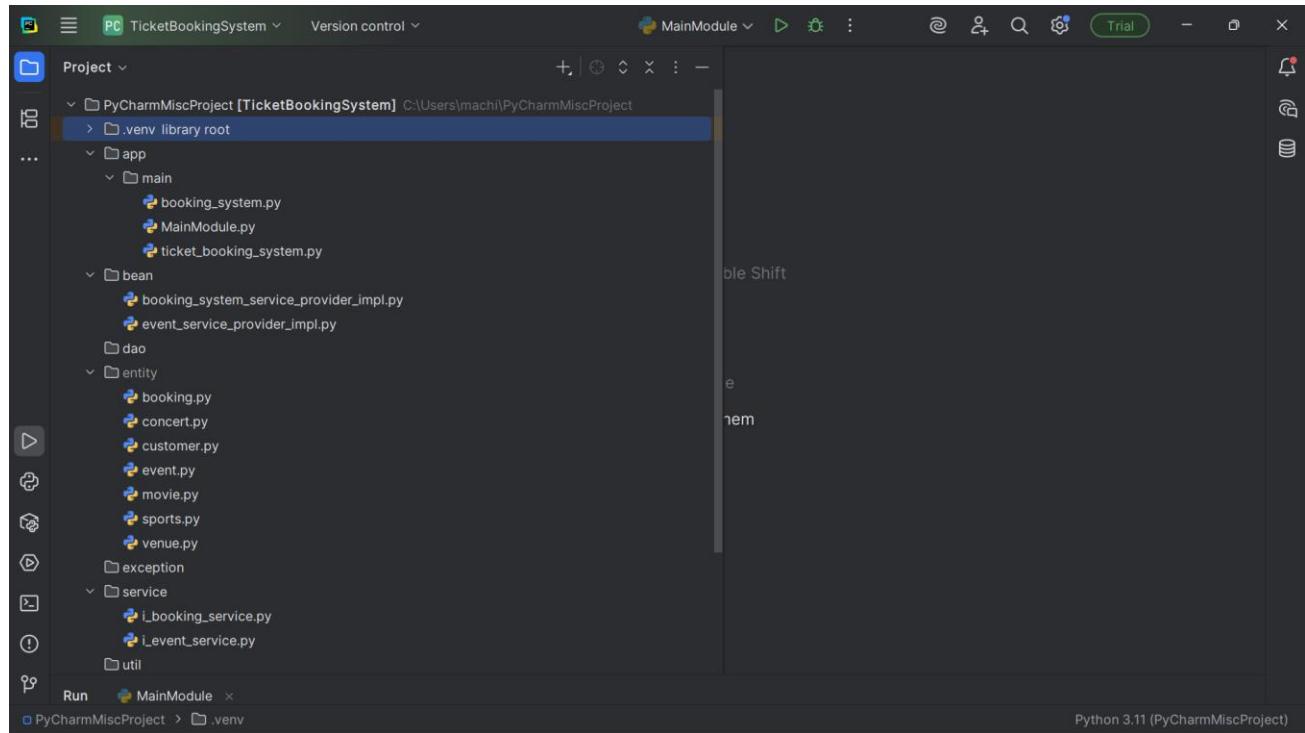
```
Enter command: get_event_details
→ Event 1:
Concert Event Details
Name      : avengers
Date      : 2025-06-12
Time      : 13:00:00
Artist    : dhamini
Concert Type: peptalk
Seats     : 200/200
Ticket Price : ₹20000.00
Type      : Concert

Venue Details
Venue Name : dhams
Address    : anantapuram

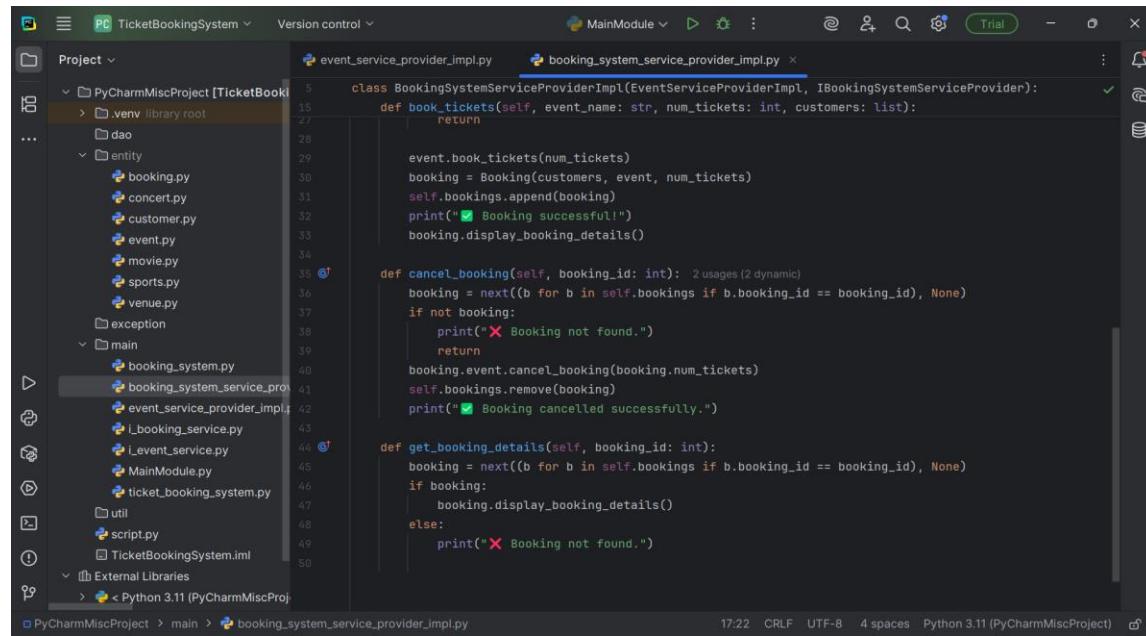
Available Commands:
1. create_event
2. book_tickets
3. cancel_tickets
4. get_available_seats
5. get_event_details
```

The terminal window has tabs for `MainModule.py` and `event.py`. The status bar at the bottom indicates Python 3.11 (PyCharmMiscProject).

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



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



```
class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
    def book_tickets(self, event_name: str, num_tickets: int, customers: list):
        return

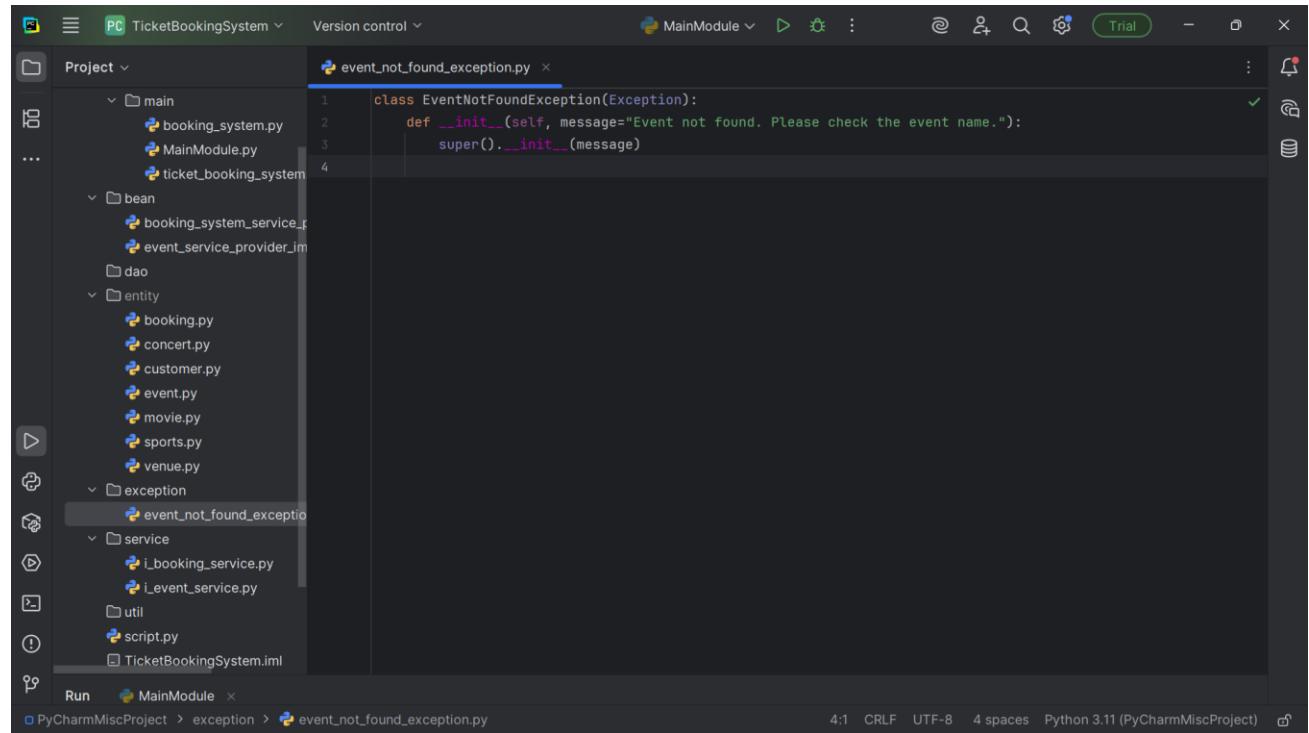
    def event_book_tickets(self, num_tickets):
        booking = Booking(customers, event_name, num_tickets)
        self.bookings.append(booking)
        print("Booking successful!")
        booking.display_booking_details()

    def cancel_booking(self, booking_id: int):
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if not booking:
            print("Booking not found.")
            return
        booking.event.cancel_booking(booking.num_tickets)
        self.bookings.remove(booking)
        print("Booking cancelled successfully.")

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

**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.



```
class EventNotFoundException(Exception):
    def __init__(self, message="Event not found. Please check the event name."):
        super().__init__(message)
```

2. **InvalidBookingIDException** throw this exception when user entered the invalid bookingId when he tries to view the booking or cancel the booking.

The screenshot shows the PyCharm IDE interface with the project 'TicketBookingSystem' open. The left sidebar displays the project structure with packages like 'main', 'bean', 'dao', and 'entity'. Under 'entity', there are files for 'booking.py', 'concert.py', 'customer.py', 'event.py', 'movie.py', 'sports.py', and 'venue.py'. A 'exception' folder contains two files: 'event\_not\_found\_exception.py' and 'invalid\_booking\_id\_exception.py'. The right pane shows the code for 'invalid\_booking\_id\_exception.py':

```
1 class InvalidBookingIDException(Exception):
2     def __init__(self, message="Booking ID is invalid or not found."):
3         super().__init__(message)
```

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.

Code :

```
from bean.booking_system_service_provider_impl import BookingSystemServiceProviderImpl
from entity.venue import Venue
from entity.customer import Customer
```

```
# Custom exceptions
from exception.event_not_found_exception import EventNotFoundException
from exception.invalid_booking_id_exception import InvalidBookingIDException

class TicketBookingSystem:
    def __init__(self):
        self.system = BookingSystemServiceProviderImpl()

    def book_tickets(self):
        event_name = input("Enter event name to book: ")
        event = next((e for e in self.system.events if e.event_name.lower() == event_name.lower()), None)
        if not event:
```

```
raise EventNotFoundException()

num = int(input("Enter number of tickets: "))
customers = []
for i in range(num):
    print(f"\nCustomer {i + 1}:")
    name = input(" Name: ")
    email = input(" Email: ")
    phone = input(" Phone: ")
    customers.append(Customer(name, email, phone))

self.system.book_tickets(event_name, num, customers)

def cancel_tickets(self):
    booking_id = int(input("Enter booking ID to cancel: "))
    booking = next((b for b in self.system.bookings if b.booking_id == booking_id), None)
    if not booking:
        raise InvalidBookingIDException()

    self.system.cancel_booking(booking_id)

def get_booking_details(self):
    booking_id = int(input("Enter booking ID to view: "))
    booking = next((b for b in self.system.bookings if b.booking_id == booking_id), None)
    if not booking:
        raise InvalidBookingIDException()

    self.system.get_booking_details(booking_id)

def run(self):
    print("\n>Welcome to the Ticket Booking System ")
    while True:
        try:
            print("\n<img alt='star icon' data-bbox='205 735 225 755' style='vertical-align: middle;"/> 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. get_booking_details")
            print("7. exit")
        except KeyboardInterrupt:
            print("\nExiting...")
```

```
command = input("\nEnter command: ").strip().lower()

if command == "create_event":
    name = input("Enter event name: ")
    date = input("Enter event date (YYYY-MM-DD): ")
    time = input("Enter event time (HH:MM): ")
    venue_name = input("Enter venue name: ")
    address = input("Enter venue address: ")
    venue = Venue(venue_name, address)
    total_seats = int(input("Enter total seats: "))
    price = float(input("Enter ticket price: "))
    event_type = input("Enter event type (Movie/Concert/Sports): ")
    self.system.create_event(name, date, time, total_seats, price, event_type, venue)

elif command == "book_tickets":
    self.book_tickets()

elif command == "cancel_tickets":
    self.cancel_tickets()

elif command == "get_available_seats":
    self.system.get_available_no_of_tickets()

elif command == "get_event_details":
    self.system.get_event_details()

elif command == "get_booking_details":
    self.get_booking_details()

elif command == "exit":
    print("👋 Exiting Ticket Booking System. Goodbye!")
    break

else:
    print("❌ Invalid command. Please try again.")

except EventNotFoundException as e:
    print(f"❌ {e}")
except InvalidBookingIDException as e:
    print(f"❌ {e}")
```

```
except AttributeError:  
    print("X NullPointerException: Something was not initialized properly.")  
except Exception as e:  
    print(f"X Unexpected Error: {e}")
```

The image shows two side-by-side screenshots of the PyCharm IDE interface, both displaying the file `ticket_booking_system.py`. The left screenshot shows the original code with the error handling logic. The right screenshot shows the code after the modification where the exception handling block has been removed.

**Left Screenshot (Original Code):**

```
class TicketBookingSystem: 2 usages  
    def __init__(self):  
        self.system = BookingSystemServiceProviderImpl()  
  
    def book_tickets(self): 1 usage  
        event_name = input("Enter event name to book: ")  
        event = next((e for e in self.system.events if e.event_name.lower() == event_name.lower()), None)  
        if not event:  
            raise EventNotFoundException()  
  
        num = int(input("Enter number of tickets: "))  
        customers = []  
        for i in range(num):  
            print(f"\nCustomer {i + 1}:")  
            name = input(" Name: ")  
            email = input(" Email: ")  
            phone = input(" Phone: ")  
            customers.append(Customer(name, email, phone))  
  
        self.system.book_tickets(event_name, num, customers)  
  
    def cancel_tickets(self): 1 usage  
        booking_id = int(input("Enter booking ID to cancel: "))  
        booking = next((b for b in self.system.bookings if b.booking_id == booking_id), None)  
        if not booking:  
            raise InvalidBookingIDException()
```

**Right Screenshot (Modified Code):**

```
class TicketBookingSystem: 2 usages  
    def cancel_tickets(self): 1 usage  
  
        self.system.cancel_booking(booking_id)  
  
    def get_booking_details(self): 1 usage  
        booking_id = int(input("Enter booking ID to view: "))  
        booking = next((b for b in self.system.bookings if b.booking_id == booking_id), None)  
        if not booking:  
            raise InvalidBookingIDException()  
  
        self.system.get_booking_details(booking_id)  
  
    def run(self): 1 usage  
        print("\n\t Welcome to the Ticket Booking System \t")  
        while True:  
            try:  
                print("\n★ 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. get_booking_details")  
                print("7. exit")
```

The screenshot shows two instances of the PyCharm IDE. Both instances have the same project structure and are editing the file `ticket_booking_system.py`.

**Project Structure:**

- main
- bean
- dao
- entity
- exception
- service
- util
- script.py

**Code Content (Top Editor):**

```
10     class TicketBookingSystem: 2 usages
47         def run(self): 1 usage
60
61             command = input("\nEnter command: ").strip().lower()
62
63             if command == "create_event":
64                 name = input("Enter event name: ")
65                 date = input("Enter event date (YYYY-MM-DD): ")
66                 time = input("Enter event time (HH:MM): ")
67                 venue_name = input("Enter venue name: ")
68                 address = input("Enter venue address: ")
69                 venue = Venue(venue_name, address)
70                 total_seats = int(input("Enter total seats: "))
71                 price = float(input("Enter ticket price: "))
72                 event_type = input("Enter event type (Movie/Concert/Sports): ")
73                 self.system.create_event(name, date, time, total_seats, price, event_type, venue)
74
75             elif command == "book_tickets":
76                 self.book_tickets()
77
78             elif command == "cancel_tickets":
79                 self.cancel_tickets()
80
81             elif command == "get_available_seats":
82                 self.system.get_available_no_of_tickets()
```

**Code Content (Bottom Editor):**

```
10     class TicketBookingSystem: 2 usages
47         def run(self): 1 usage
80
81             elif command == "get_available_seats":
82                 self.system.get_available_no_of_tickets()
83
84             elif command == "get_event_details":
85                 self.system.get_event_details()
86
87             elif command == "get_booking_details":
88                 self.get_booking_details()
89
90             elif command == "exit":
91                 print("👉 Exiting Ticket Booking System. Goodbye!")
92                 break
93
94             else:
95                 print("✖ Invalid command. Please try again.")
96
97         except EventNotFoundException as e:
98             print(f"✖ {e}")
99         except InvalidBookingIDException as e:
100            print(f"✖ {e}")
101        except AttributeError:
102            print("✖ NullPointerException: Something was not initialized properly.")
103        except Exception as e:
104            print(f"✖ Unexpected Error: {e}")
```

### 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.
- a. **Booking** class attribute `customers` to List of customers

**Code :**

```
from datetime import datetime
```

```
from typing import List
from entity.customer import Customer
from entity.event import Event

class Booking:
    _booking_counter = 1 # static booking ID

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

        self.customers = customers # ✅ List of Customer objects
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
        self.booking_date = datetime.now()

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

    def display_booking_details(self):
        print(f"\n📝 Booking ID : {self.booking_id}")
        print(f"Booking Date : {self.booking_date.strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"No. of Tickets : {self.num_tickets}")
        print(f"Total Cost (₹) : {self.total_cost:.2f}")

        print("\n👤 Event Info:")
        self.event.display_event_details()

        print("\n👥 Customer Info:")
        for idx, customer in enumerate(self.customers, 1):
            print(f"\n--- Customer {idx} ---")
            customer.display_customer_details()
```

```
class Booking:
    def __init__(self, customers: List[Customer], event: Event, num_tickets: int):
        self.booking_id = Booking._booking_counter
        Booking._booking_counter += 1
        self.customers = customers # List of Customer objects
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
        self.booking_date = datetime.now()

    def calculate_total_cost(self) -> float:
        return self.num_tickets * self.event.ticket_price

    def display_booking_details(self):
        print(f"\nBooking ID : {self.booking_id}")
        print(f"Booking Date : {self.booking_date.strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"No. of Tickets : {self.num_tickets}")
        print(f"Total Cost : {self.total_cost:.2f}")

        print("\nEvent Info:")
        self.event.display_event_details()

        print("\nCustomer Info:")
        for idx, customer in enumerate(self.customers, 1):
            print(f"\nCustomer {idx} ---")
            customer.display_customer_details()
```

b. **BookingSystem** class attribute events to List of events and perform the same operation.

#### Code

```
from service.i_booking_service import IBookingSystemServiceProvider
from bean.event_service_provider_impl import EventServiceProviderImpl
from entity.booking import Booking
class BookingSystemServiceProviderImpl(EventServiceProviderImpl,
IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__() # Initializes self.events from parent
        self.bookings = [] # Booking list

    def calculate_booking_cost(self, num_tickets: int) -> float:
        # Assumes one event context; usually handled during booking
        print(f"\nCost for {num_tickets} tickets is calculated based on event during booking.")
        return 0.0

    def book_tickets(self, event_name: str, num_tickets: int, customers: list):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if not event:
            print("X Event not found.")
```

```
return

if event.available_seats < num_tickets:
    print("X Not enough tickets available.")
    return

if len(customers) != num_tickets:
    print("X Number of customers must match number of tickets.")
    return

event.book_tickets(num_tickets)
booking = Booking(customers, event, num_tickets)
self.bookings.append(booking)
print("✓ Booking successful!")
booking.display_booking_details()

def cancel_booking(self, booking_id: int):
    booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
    if not booking:
        print("X Booking not found.")
        return
    booking.event.cancel_booking(booking.num_tickets)
    self.bookings.remove(booking)
    print("✓ Booking cancelled successfully.")

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

```
class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
    def book_tickets(self, event_name: str, num_tickets: int, customers: list):
        print("X Number of customers must match number of tickets.")
        return

    event.book_tickets(num_tickets)
    booking = Booking(customers, event, num_tickets)
    self.bookings.append(booking)
    print("✓ Booking successful!")
    booking.display_booking_details()

    def cancel_booking(self, booking_id: int):
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if not booking:
            print("X Booking not found.")
            return
        booking.event.cancel_booking(booking.num_tickets)
        self.bookings.remove(booking)
        print("✓ Booking cancelled successfully.")

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

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.

Code:

```
from typing import Set
```

```
class Booking:
    def __init__(self, customers: Set[Customer], event: Event, num_tickets: int):
        self.customers = customers # Now a set
        self.bookings.append(Booking(set(customers), event, num_tickets))
```

```
from typing import Set
```

```
class BookingSystemServiceProviderImpl:
    def __init__(self):
        self.events: Set[Event] = set() # Set of events
        self.bookings = []
    def __eq__(self, other):
        return isinstance(other, Customer) and self.email == other.email
```

```
def __hash__(self):
```

```
        return hash(self.email)

    def __eq__(self, other):
        return isinstance(other, Event) and self.event_name.lower() == other.event_name.lower() and
               self.event_date == other.event_date

    def __hash__(self):
        return hash((self.event_name.lower(), self.event_date))
    sorted_events = sorted(
        self.events,
        key=lambda e: (e.event_name.lower(), e.venue.venue_name.lower()))
    )

    for event in sorted_events:
        event.display_event_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.

**Code :**

```
from typing import Dict
from entity.customer import Customer
```

```
class Booking:
    def __init__(self, customers: Dict[int, Customer], event, num_tickets: int):
        self.customers = customers #  Map: ticket_no → Customer

    # Convert list to dict {1: customer1, 2: customer2, ...}
    customer_dict = {i+1: customers[i] for i in range(len(customers))}

    booking = Booking(customer_dict, event, num_tickets)

    for ticket_no, customer in self.customers.items():
        print(f"\n--- Ticket {ticket_no} ---")
        customer.display_customer_details()

    self.events: Dict[str, Event] = {} #  event_name.lower() → Event
    event = self.events.get(event_name.lower())

    self.events[event.event_name.lower()] = event
```

```
sorted_events = sorted(self.events.values(), key=lambda e: (e.event_name.lower(),
e.venue.venue_name.lower()))
for idx, event in enumerate(sorted_events, 1):
    event.display_event_details()
```

made all the changes

**Task 11: Database Connectivity.**

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

**Venue**

**Code :**

```
# entity/venue.py
```

```
class Venue:
    def __init__(self, venue_id=None, venue_name="", address=""):
        self.venue_id = venue_id      #  Add venue_id for database
        self.venue_name = venue_name
        self.address = address

    def get_venue_id(self):
        return self.venue_id

    def set_venue_id(self, venue_id):
        self.venue_id = venue_id

    def get_venue_name(self):
        return self.venue_name

    def get_address(self):
        return self.address

    def set_venue_name(self, name):
        self.venue_name = name

    def set_address(self, address):
        self.address = address

    def display_venue_details(self):
        print("\n ━━━━ Venue Details")
        print(f"Venue ID : {self.venue_id}")
        print(f"Venue Name : {self.venue_name}")
        print(f"Address : {self.address}")
```

The screenshot shows a PyCharm interface with the following details:

- Project Structure:** The project is named "TicketBookingSystem". It contains several packages and modules:
  - main:** booking\_system.py, MainModule.py, ticket\_booking\_system.py.
  - bean:** booking\_system\_service.py, event\_service\_provider\_iml.py.
  - dao:** entity.
  - entity:** booking.py, concert.py, customer.py, event.py, movie.py, sports.py, venue.py.
  - exception:** (empty folder).
  - service:** i\_booking\_service.py, i\_Event\_Service.py.
  - util:** script.py.
  - ticketBookingSystem.iml:** (IntelliJ module descriptor).
- Code Editor:** The file "venue.py" is open in the editor. The code defines a class "Venue" with methods for setting and getting venue ID, name, and address, and a method to display venue details.
- Status Bar:** Shows the file is 25:1, CRLF, UTF-8, 4 spaces, Python 3.11 (PyCharmMiscProject).

Event :

```
# entity/event.py
from datetime import date, time
from entity.venue import Venue
```

class Event:

```
    def __init__(self, event_id=None, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type=""):

        self.event_id = event_id #  Add event_id
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue = venue or Venue()
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    # ----- Getters -----
    def get_event_id(self):
        return self.event_id
```

```
def get_event_name(self):
    return self.event_name

def get_event_date(self):
    return self.event_date

def get_event_time(self):
    return self.event_time

def get_venue(self):
    return self.venue

def get_total_seats(self):
    return self.total_seats

def get_available_seats(self):
    return self.available_seats

def get_ticket_price(self):
    return self.ticket_price

def get_event_type(self):
    return self.event_type

# ----- Setters -----
def set_event_id(self, eid):
    self.event_id = eid

def set_event_name(self, name):
    self.event_name = name

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

def set_event_time(self, event_time):
    self.event_time = event_time

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

def set_total_seats(self, seats):
    self.total_seats = seats
```

```
def set_available_seats(self, seats):
    self.available_seats = seats

def set_ticket_price(self, price):
    self.ticket_price = price

def set_event_type(self, e_type):
    self.event_type = e_type

# ----- Display -----
def display_event_details(self):
    print("\n 📃 Event Details")
    print(f"Event ID : {self.event_id}")
    print(f"Name : {self.event_name}")
    print(f"Date : {self.event_date}")
    print(f"Time : {self.event_time}")
    print(f"Type : {self.event_type}")
    print(f"Seats : {self.available_seats}/{self.total_seats}")
    print(f"Ticket Price: ₹{self.ticket_price:.2f}")
    self.venue.display_venue_details()
```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The left sidebar shows a tree view of the project structure under 'Project'. It includes main modules like 'booking\_system.py', 'MainModule.py', and 'ticket\_booking\_system', along with sub-modules such as 'bean', 'entity', 'exception', 'service', 'util', and 'script.py'. A file named 'venue.py' is currently selected.
- Code Editor:** The right pane displays the content of 'venue.py'. The code defines a 'Venue' class with attributes for venue\_id, venue\_name, and address, and methods for setting and getting these values. It also includes a 'display\_venue\_details' method that prints the venue's ID, name, and address.
- Status Bar:** At the bottom, the status bar shows '25:1 CRLF UTF-8 4 spaces Python 3.11 (PyCharmMiscProject)'.

Customer :

Code :

```
# entity/customer.py
```

```
class Customer:  
    def __init__(self, customer_id=None, customer_name="", email="", phone_number=""):  
        self.customer_id = customer_id      #  Add this  
        self.customer_name = customer_name  
        self.email = email  
        self.phone_number = phone_number  
  
    # ----- Getters -----  
    def get_customer_id(self):  
        return self.customer_id  
  
    def get_customer_name(self):  
        return self.customer_name  
  
    def get_email(self):  
        return self.email  
  
    def get_phone_number(self):  
        return self.phone_number  
  
    # ----- Setters -----  
    def set_customer_id(self, cid):  
        self.customer_id = cid  
  
    def set_customer_name(self, name):  
        self.customer_name = name  
  
    def set_email(self, email):  
        self.email = email  
  
    def set_phone_number(self, phone_number):  
        self.phone_number = phone_number  
  
    # ----- Display -----  
    def display_customer_details(self):  
        print("\n👤 Customer Details")  
        print(f"Customer ID : {self.customer_id}")  
        print(f"Name      : {self.customer_name}")  
        print(f"Email     : {self.email}")  
        print(f"Phone Number : {self.phone_number}")
```

```
class Customer: 4 usages
    def __init__(self, customer_id=None, customer_name="", email="", phone_number="")
        self.customer_id = customer_id          # ✓ Add this
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number
```

Booking :

Code :

```
# entity/booking.py
```

```
class Booking:
    def __init__(self, booking_id=None, event_id=None, customer_ids=None,
                 num_tickets=0, total_cost=0.0, booking_date=None):
        self.booking_id = booking_id
        self.event_id = event_id          # Foreign key reference to Event table
        self.customer_ids = customer_ids or []  # List of customer IDs
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date

    # ----- Getters -----
    def get_booking_id(self):
        return self.booking_id

    def get_event_id(self):
        return self.event_id

    def get_customer_ids(self):
        return self.customer_ids

    def get_num_tickets(self):
        return self.num_tickets

    def get_total_cost(self):
        return self.total_cost

    def get_booking_date(self):
        return self.booking_date

    # ----- Setters -----
    def set_booking_id(self, bid):
        self.booking_id = bid
```

```
def set_event_id(self, eid):
    self.event_id = eid

def set_customer_ids(self, ids):
    self.customer_ids = ids

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

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

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

# ----- Display -----
def display_booking_details(self):
    print("\nBooking Details")
    print(f"Booking ID : {self.booking_id}")
    print(f"Event ID : {self.event_id}")
    print(f"Customer IDs : {' , '.join(map(str, self.customer_ids))}")
    print(f"No. of Tickets: {self.num_tickets}")
    print(f"Total Cost : ₹{self.total_cost:.2f}")
    print(f"Booking Date : {self.booking_date}")
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'TicketBookingSystem'. The 'entity' folder contains several files: booking.py, concert.py, customer.py, event.py, movie.py, sports.py, and venue.py. The 'booking.py' file is currently open in the code editor. The code defines a class 'Booking' with methods for initializing attributes like booking\_id, event\_id, customer\_ids, num\_tickets, total\_cost, and booking\_date. It also includes getters for these attributes.

```
class Booking:
    def __init__(self, booking_id=None, event_id=None, customer_ids=None, num_tickets=0, total_cost=0.0, booking_date=None):
        self.booking_id = booking_id
        self.event_id = event_id
        self.customer_ids = customer_ids or []
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date

    # ----- Getters -----
    def get_booking_id(self):
        return self.booking_id

    def get_event_id(self):
        return self.event_id

    def get_customer_ids(self):
        return self.customer_ids

    def get_num_tickets(self):
        return self.num_tickets

    def get_total_cost(self):
        return self.total_cost
```

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

Performed in the previous steps

This screenshot is identical to the one above, showing the PyCharm interface with the 'entity' folder and 'booking.py' file selected. The code editor shows the same definition of the 'Booking' class.

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

Code :

```
from abc import ABC, abstractmethod
from entity.venue import Venue
```

```
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
```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "TicketBookingSystem". It contains several packages and modules:
  - ticket\_booking\_system:** Contains bean, dao, entity, exception, service, util, and a script.py file.
  - booking\_system\_service\_provider:** Contains a main/i\_event\_service.py file.
  - entity:** Contains booking.py, concert.py, customer.py, event.py, movie.py, sports.py, and venue.py files.
  - service:** Contains i\_booking\_service.py and i\_event\_service.py files.
- i\_event\_service.py Content:** The code defines the IEventServiceProvider abstract class with three abstract methods: create\_event, get\_event\_details, and get\_available\_no\_of\_tickets, each returning a pass statement.
- Status Bar:** Shows the file is 211 lines long, uses CRLF line endings, is in UTF-8 encoding, has 4 spaces for indentation, and is using Python 3.11 (PyCharmMiscProject).

Code :

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

```
class IBookingSystemServiceProvider(ABC):
```

```
    @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
```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "TicketBookingSystem". It contains several packages and modules:
  - main:** booking\_system.py, MainModule.py, ticket\_booking\_system.
  - bean:** booking\_system\_service.py, event\_service\_provider\_impl.py.
  - dao:**
  - entity:** booking.py, concert.py, customer.py, event.py, movie.py, sports.py, venue.py.
  - exception:**
  - service:** Ibooking\_service.py (selected), Ievent\_service.py, util, script.py.
- Code Editor:** The file `Ibooking_service.py` is open. It contains the abstract methods defined in the question.
- Bottom Status Bar:** Shows the current file path: `PyCharmMiscProject > service > Ibooking_service.py`, and other details like encoding (CRLF), line endings (UTF-8), and Python version (Python 3.11).

Code:

```
# main/event_service_provider_impl.py
```

```
from service.i_event_service import IEventServiceProvider
from entity.venue import Venue
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime

class EventServiceProviderImpl(IEventServiceProvider):
```

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

    if event_type.lower() == "movie":
        genre = input("Enter genre: ")
        actor = input("Enter actor name: ")
        actress = input("Enter actress name: ")
        event = Movie(event_name, event_date, event_time, venue,
                      total_seats, ticket_price, "Movie",
                      genre, actor, actress)

    elif event_type.lower() == "concert":
        artist = input("Enter artist/band name: ")
        concert_type = input("Enter concert type: ")
        event = Concert(event_name, event_date, event_time, venue,
                        total_seats, ticket_price, "Concert",
                        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,
                       total_seats, ticket_price, "Sports",
                       sport_name, teams_name)

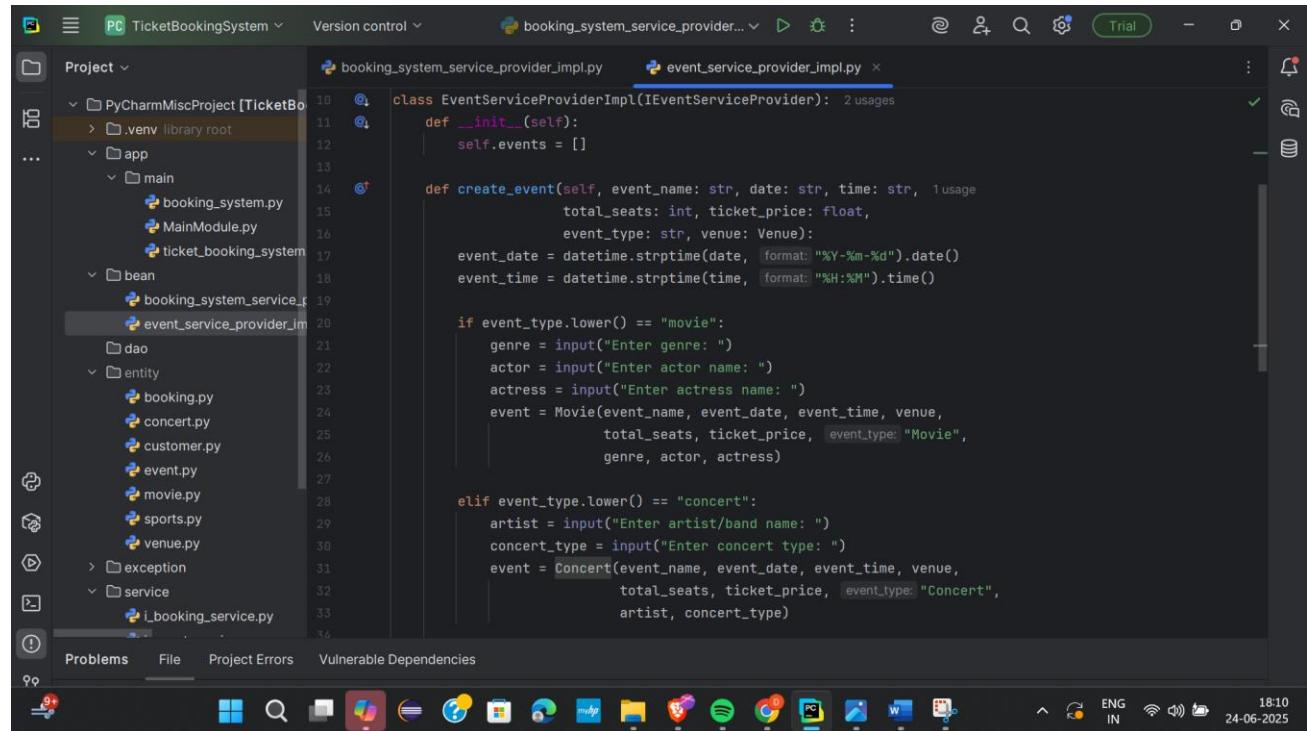
    else:
        print("X Invalid event type.")
        return None

    self.events.append(event)
    print("✓ Event created successfully!")
    return event

def get_event_details(self):
    if not self.events:
        print("⚠ No events available.")
    return []
```

```
for idx, event in enumerate(self.events, start=1):
    print(f"\n❖ Event {idx}:")
    event.display_event_details()
return self.events

def get_available_no_of_tickets(self):
    total = sum(event.get_available_seats() for event in self.events)
    print(f"\n▣ Total Available Tickets: {total}")
    return total
```



Code :

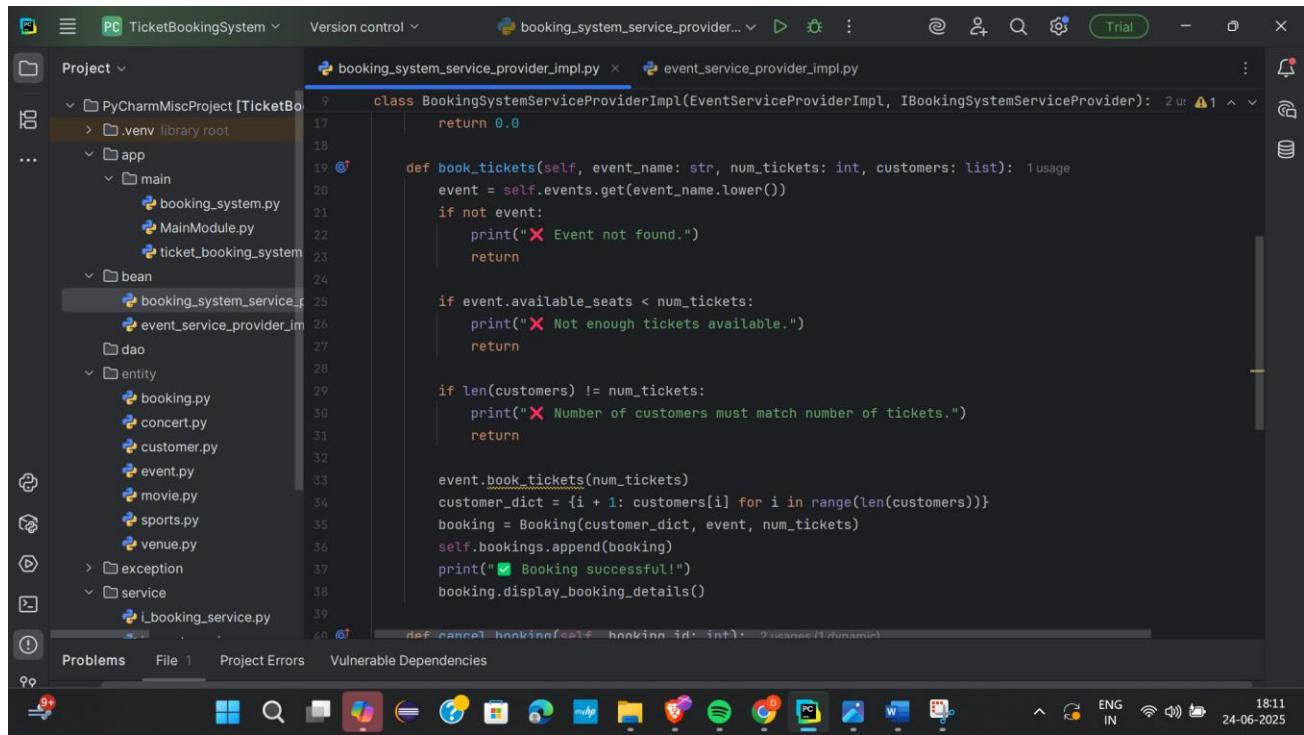
```
# Updated BookingSystemServiceProviderImpl class using Dict for Task 10.3
from service.i_booking_service import IBookingSystemServiceProvider
from bean.event_service_provider_impl import EventServiceProviderImpl
from entity.booking import Booking

from entity.event import Event
from typing import Dict

class BookingSystemServiceProviderImpl(EventServiceProviderImpl,
IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__()
        self.events: Dict[str, Event] = {}
        self.bookings = []
```

```
def calculate_booking_cost(self, num_tickets: int) -> float:  
    print(f" 12 Cost for {num_tickets} tickets is calculated based on event during booking.")  
    return 0.0  
  
def book_tickets(self, event_name: str, num_tickets: int, customers: list):  
    event = self.events.get(event_name.lower())  
    if not event:  
        print("X Event not found.")  
        return  
  
    if event.available_seats < num_tickets:  
        print("X Not enough tickets available.")  
        return  
  
    if len(customers) != num_tickets:  
        print("X Number of customers must match number of tickets.")  
        return  
  
    event.book_tickets(num_tickets)  
    customer_dict = {i + 1: customers[i] for i in range(len(customers))}  
    booking = Booking(customer_dict, event, num_tickets)  
    self.bookings.append(booking)  
    print("✓ Booking successful!")  
    booking.display_booking_details()  
  
def cancel_booking(self, booking_id: int):  
    booking = next((b for b in self.bookings if b.booking_id == booking_id), None)  
    if not booking:  
        print("X Booking not found.")  
        return  
    booking.event.cancel_booking(booking.num_tickets)  
    self.bookings.remove(booking)  
    print("✓ Booking cancelled successfully.")  
  
def get_booking_details(self, booking_id: int):  
    booking = next((b for b in self.bookings if b.booking_id == booking_id), None)  
    if booking:  
        booking.display_booking_details()  
    else:  
        print("X Booking not found.")
```

```
def get_event_details(self):
    sorted_events = sorted(self.events.values(), key=lambda e: (e.event_name.lower(),
e.venue.venue_name.lower()))
    for idx, event in enumerate(sorted_events, 1):
        print(f"\n❖ Event {idx}:")
        event.display_event_details()
```



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

- **create\_event(event\_name: str, date:str, time:str, total\_seats: int, ticket\_price: float, event\_type: str, venu: Venu):** Create a new event with the specified details and event type (movie, sport or concert) and return event object and should store in database.
- **getEventDetails():** return array of event details from the database.
- **getAvailableNoOfTickets():** return the total available tickets from the database.
- **calculate\_booking\_cost(num\_tickets):** Calculate and set the total cost of the booking.
- **book\_tickets(eventName:str, num\_tickets, listOfCustomer):** Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class and stored in database.
- **cancel\_booking(booking\_id):** Cancel the booking and update the available seats and stored in database.
- **get\_booking\_details(booking\_id):** get the booking details from database.

**Code :**

```
from abc import ABC, abstractmethod
from entity.venue import Venue
```

```
from entity.customer import Customer
from entity.event import Event
from entity.booking import Booking
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) -> Event:
        """Insert event into database and return created Event object"""
        pass

    @abstractmethod
    def get_event_details(self) -> List[Event]:
        """Fetch all events from the database"""
        pass

    @abstractmethod
    def get_available_no_of_tickets(self) -> int:
        """Return total available seats across all events"""
        pass

    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int, event: Event) -> float:
        """Return cost = tickets × price"""
        pass

    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, customers: List[Customer]) ->
        Booking:
        """Book specified number of tickets and save to DB"""
        pass

    @abstractmethod
    def cancel_booking(self, booking_id: int) -> bool:
        """Cancel a booking by ID and update available seats"""
        pass

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

"""Return booking details from the DB""""

pass

The screenshot shows the VS Code interface with the project structure on the left and the code editor on the right. The code editor displays the file `i_booking_system_repository.py`. The implementation of the `get_booking_details` method is shown:

```
from abc import ABC, abstractmethod
from entity.venue import Venue
from entity.customer import Customer
from entity.event import Event
from entity.booking import Booking
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) -> Event:
        """Insert event into database and return created Event object"""

    @abstractmethod
    def get_event_details(self) -> List[Event]:
        """Fetch all events from the database"""

    @abstractmethod
    def get_available_no_of_tickets(self) -> int:
        """Return total available seats across all events"""

    pass
```

The screenshot shows the VS Code interface with the project structure on the left and the code editor on the right. The code editor displays the file `i_booking_system_repository.py`. The implementation of the `calculate_booking_cost` method is shown:

```
class IBookingSystemRepository(ABC):

    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int, event: Event) -> float:
        """Return cost = tickets * price"""

    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, customers: List[Customer]) -> Booking:
        """Book specified number of tickets and save to DB"""

    @abstractmethod  # usage (1 dynamic)
    def cancel_booking(self, booking_id: int) -> bool:
        """Cancel a booking by ID and update available seats"""

    @abstractmethod
    def get_booking_details(self, booking_id: int) -> Booking:
        """Return booking details from the DB"""

    pass
```

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

Code :

```
import mysql.connector
from service.i_booking_system_repository import IBookingSystemRepository
from util.db_util import DBUtil
from entity.venue import Venue
from entity.customer import Customer
from entity.event import Event
from entity.booking import Booking
from typing import List

class BookingSystemRepositoryImpl(IBookingSystemRepository):

    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue: Venue) -> Event:
        conn = DBUtil.get_db_conn()
        try:
            cursor = conn.cursor()
            # Insert venue first (if not exists)
            cursor.execute("INSERT INTO venue (venue_name, address) VALUES (%s, %s)",
                           (venue.venue_name, venue.address))
            venue_id = cursor.lastrowid

            # Insert event
            cursor.execute("""
                INSERT INTO event (event_name, event_date, event_time, total_seats,
available_seats,
                           ticket_price, event_type, venue_id)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                """, (event_name, date, time, total_seats, total_seats, ticket_price, event_type,
venue_id))

            conn.commit()
            print("☑ Event saved to database.")
            return Event(cursor.lastrowid, event_name, date, time, venue, total_seats, total_seats,
ticket_price,
                           event_type)
        except mysql.connector.Error as e:
            print("☒ DB Error:", e)
        finally:
            conn.close()

    def get_event_details(self) -> List[Event]:
        events = []
```

```
conn = DBUtil.get_db_conn()
try:
    cursor = conn.cursor()
    cursor.execute("""
        SELECT e.event_id,
               e.event_name,
               e.event_date,
               e.event_time,
               e.total_seats,
               e.available_seats,
               e.ticket_price,
               e.event_type,
               v.venue_id,
               v.venue_name,
               v.address
        FROM event e
        JOIN venue v ON e.venue_id = v.venue_id
    """)
    for row in cursor.fetchall():
        venue = Venue(row[8], row[9], row[10])
        event = Event(*row[:8], venue)
        events.append(event)
except mysql.connector.Error as e:
    print("X DB Error:", e)
finally:
    conn.close()
return events

def get_available_no_of_tickets(self) -> int:
    conn = DBUtil.get_db_conn()
    try:
        cursor = conn.cursor()
        cursor.execute("SELECT SUM(available_seats) FROM event")
        result = cursor.fetchone()
        return result[0] if result and result[0] is not None else 0
    except mysql.connector.Error as e:
        print("X DB Error:", e)
    finally:
        conn.close()

def calculate_booking_cost(self, num_tickets: int, event: Event) -> float:
    return num_tickets * event.ticket_price
```

```
def book_tickets(self, event_name: str, num_tickets: int, customers: List[Customer]) -> Booking:
    conn = DBUtil.get_db_conn()
    try:
        cursor = conn.cursor()
        # Get event
        cursor.execute("SELECT event_id, available_seats, ticket_price FROM event WHERE
LOWER(event_name) = %s",
                      (event_name.lower(),))
        row = cursor.fetchone()
        if not row:
            print("X Event not found.")
            return None

        event_id, available_seats, price = row
        if available_seats < num_tickets:
            print("X Not enough tickets available.")
            return None

        # Update seats
        cursor.execute("UPDATE event SET available_seats = available_seats - %s WHERE event_id =
%s",
                      (num_tickets, event_id))

        # Insert customers and collect their IDs
        customer_ids = []
        for customer in customers:
            cursor.execute("INSERT INTO customer (customer_name, email, phone_number) VALUES
(%s, %s, %s)",
                          (customer.customer_name, customer.email, customer.phone_number))
            customer_ids.append(cursor.lastrowid)

        total_cost = num_tickets * price

        # Insert booking
        cursor.execute(
            "INSERT INTO booking (event_id, num_tickets, total_cost, booking_date) VALUES (%s, %s,
%s, NOW())",
            (event_id, num_tickets, total_cost))
        booking_id = cursor.lastrowid

        # Link customers to booking (if needed in your schema)
        # Optional join table logic here
```

```
conn.commit()
print("☑ Booking completed.")
return Booking(booking_id, event_id, customer_ids, num_tickets, total_cost, None)
except mysql.connector.Error as e:
    print("☒ DB Error:", e)
finally:
    conn.close()

def cancel_booking(self, booking_id: int) -> bool:
    conn = DBUtil.get_db_conn()
    try:
        cursor = conn.cursor()
        cursor.execute("SELECT event_id, num_tickets FROM booking WHERE booking_id = %s",
                      (booking_id,))
        booking = cursor.fetchone()
        if not booking:
            print("☒ Booking not found.")
            return False

        event_id, tickets = booking
        cursor.execute("UPDATE event SET available_seats = available_seats + %s WHERE event_id = %s",
                      (tickets, event_id))
        cursor.execute("DELETE FROM booking WHERE booking_id = %s", (booking_id,))
        conn.commit()
        print("☑ Booking cancelled.")
        return True
    except mysql.connector.Error as e:
        print("☒ DB Error:", e)
        return False
    finally:
        conn.close()

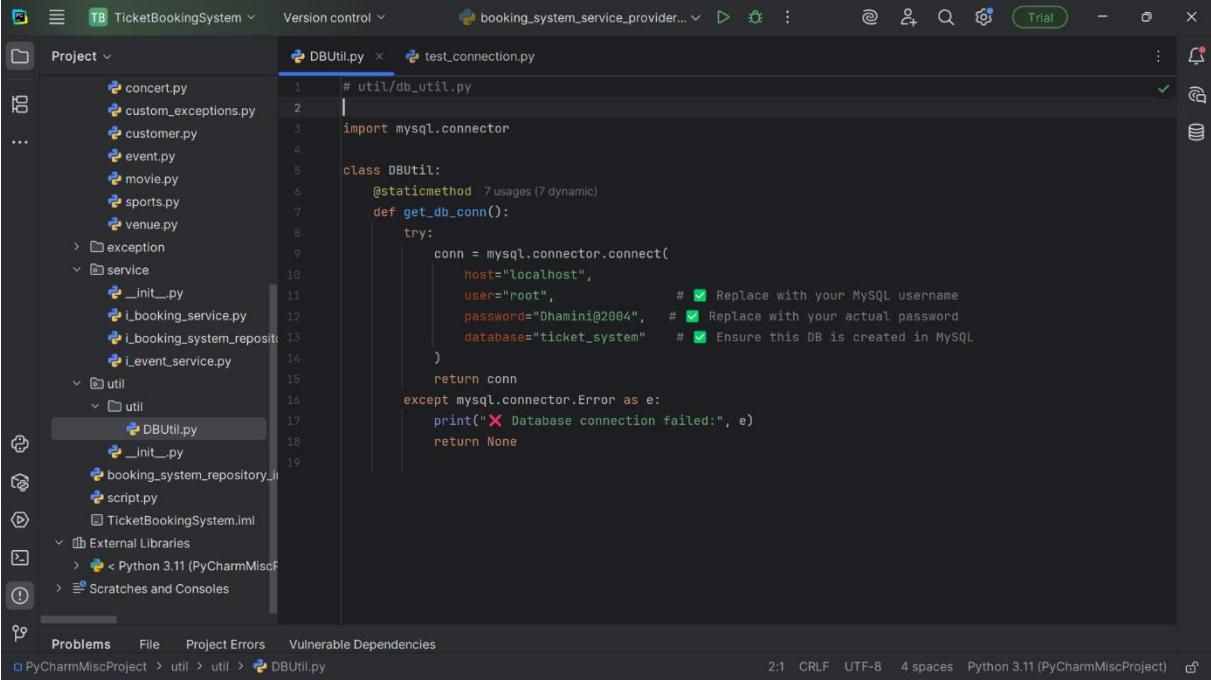
def get_booking_details(self, booking_id: int) -> Booking:
    conn = DBUtil.get_db_conn()
    try:
        cursor = conn.cursor()
        cursor.execute("SELECT event_id, num_tickets, total_cost, booking_date FROM booking
                       WHERE booking_id = %s",
                      (booking_id,))
        result = cursor.fetchone()
        if not result:
```

```
print("X Booking not found.")  
return None  
  
event_id, num_tickets, total_cost, booking_date = result  
return Booking(booking_id, event_id, [], num_tickets, total_cost, booking_date)  
except mysql.connector.Error as e:  
    print("X DB Error:", e)  
    return None  
finally:  
    conn.close()
```

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

- **static getDBConn():**
  - **Connection** Establish a connection to the database and return Connection reference
- Code :
- ```
import mysql.connector
```

```
class DBUtil:  
    @staticmethod  
    def get_db_conn():  
        try:  
            conn = mysql.connector.connect(  
                host="localhost",  
                user="root",          #  Replace with your MySQL username  
                password="password", #  Replace with your actual password  
                database="ticket_system" #  Ensure this DB is created in MySQL  
            )  
            return conn  
        except mysql.connector.Error as e:  
            print("X Database connection failed:", e)  
        return None
```

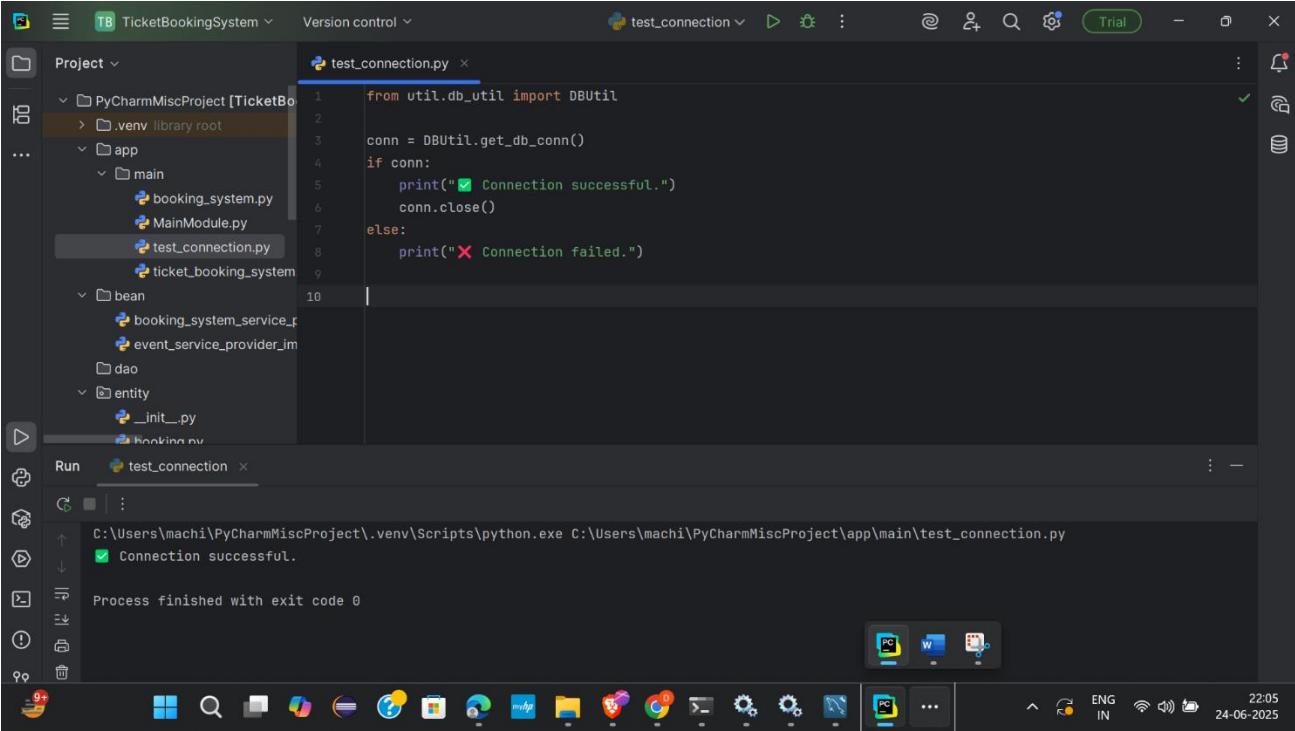


The screenshot shows the PyCharm interface with the project 'TicketBookingSystem' open. The left sidebar displays a tree view of files and packages. The 'util' package is expanded, showing 'DBUtil.py' and '\_init\_.py'. The main editor window shows the code for 'DBUtil.py':

```
# util/db_util.py
import mysql.connector

class DBUtil:
    @staticmethod
    def get_db_conn():
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",          # Replace with your MySQL username
                password="Dhamini@2084", # Replace with your actual password
                database="ticket_system" # Ensure this DB is created in MySQL
            )
            return conn
        except mysql.connector.Error as e:
            print("✗ Database connection failed:", e)
            return None
```

The status bar at the bottom indicates the file is Python 3.11 (PyCharmMiscProject).



The screenshot shows the PyCharm interface with the project 'PyCharmMiscProject [TicketBo]' open. The left sidebar shows a different project structure. The main editor window shows the code for 'test\_connection.py':

```
from util.db_util import DBUtil
conn = DBUtil.get_db_conn()
if conn:
    print("✓ Connection successful.")
    conn.close()
else:
    print("✗ Connection failed.")
```

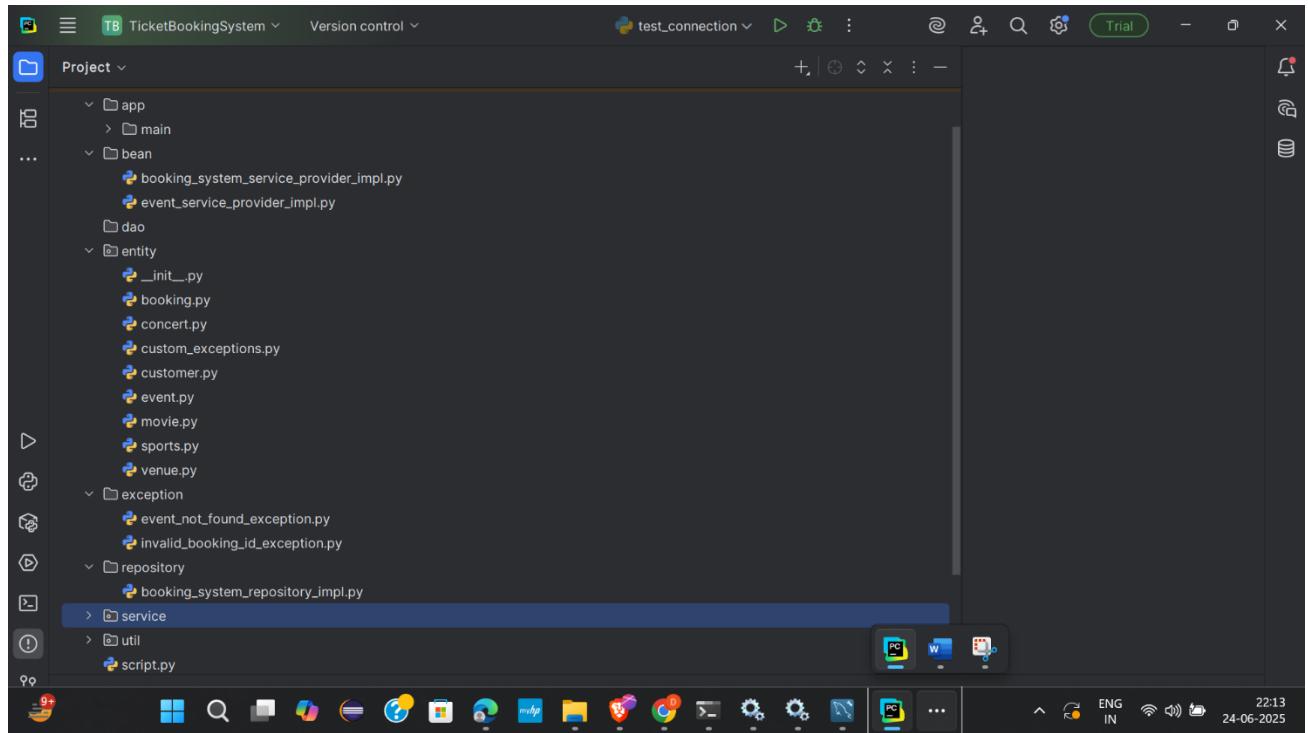
The 'Run' tab at the bottom shows the output of the run command:

```
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\app\main\test_connection.py
✓ Connection successful.

Process finished with exit code 0
```

The status bar at the bottom indicates the run was successful.

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

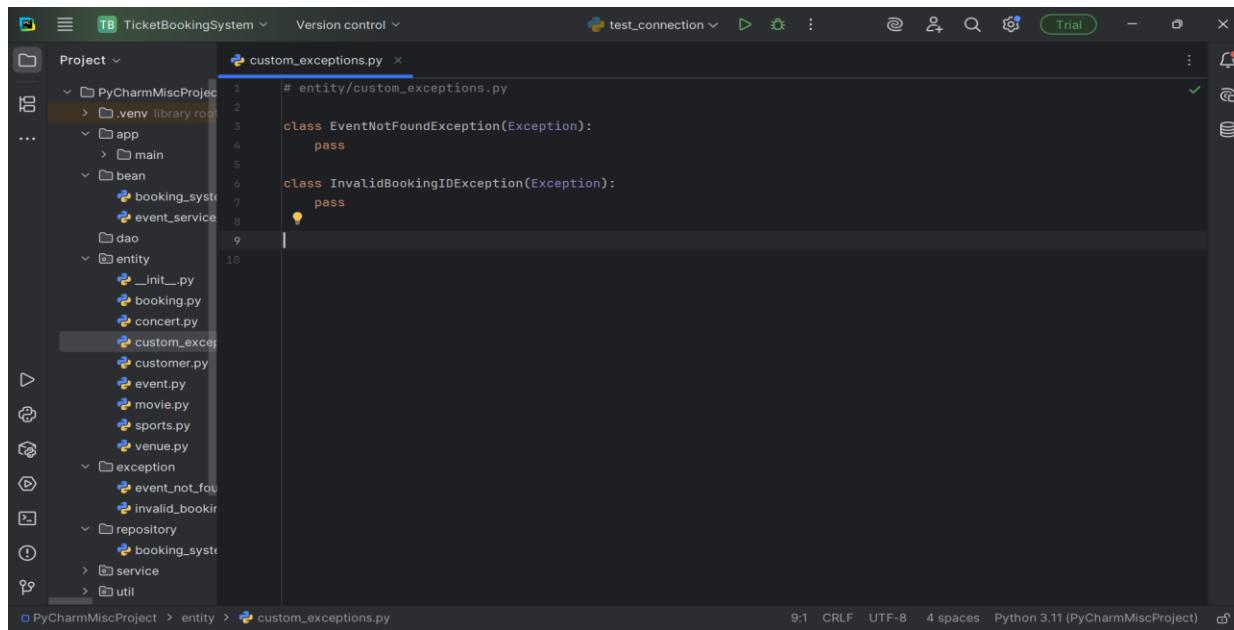


8. Should throw appropriate exception as mentioned in above task along with handle **SQLException**.

Code :

```
class EventNotFoundException(Exception):
    pass
```

```
class InvalidBookingIDException(Exception):
    pass
```



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."

Code :

```
# app/ticket_booking_system.py

from repository.booking_system_repository_impl import BookingSystemRepositoryImpl
from entity.venue import Venue
from entity.customer import Customer
from entity.custom_exceptions import EventNotFoundException, InvalidBookingIDException

class TicketBookingSystem:
    def __init__(self):
        self.repo = BookingSystemRepositoryImpl()

    def run(self):
        print(">Welcome 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. get_booking_details")
        print("7. exit")

    while True:
        command = input("\nEnter command: ").strip().lower()

        try:
            if command == "create_event":
                name = input("Event name: ")
                date = input("Date (YYYY-MM-DD): ")
                time = input("Time (HH:MM): ")
                seats = int(input("Total seats: "))
                price = float(input("Ticket price: "))
                type_ = input("Event type (Movie, Concert, Sports): ")
                vname = input("Venue name: ")
                vaddr = input("Venue address: ")
                venue = Venue(vname, vaddr)


```

```
    self.repo.create_event(name, date, time, seats, price, type_, venue)

    elif command == "book_tickets":
        event_name = input("Event name to book: ")
        num = int(input("How many tickets? "))
        customers = []
        for i in range(num):
            print(f"\nCustomer {i+1}")
            name = input("Name: ")
            email = input("Email: ")
            phone = input("Phone: ")
            customers.append(Customer(name, email, phone))
        self.repo.book_tickets(event_name, num, customers)

    elif command == "cancel_tickets":
        bid = int(input("Booking ID to cancel: "))
        self.repo.cancel_booking(bid)

    elif command == "get_available_seats":
        self.repo.get_available_no_of_tickets()

    elif command == "get_event_details":
        self.repo.get_event_details()

    elif command == "get_booking_details":
        bid = int(input("Booking ID to view: "))
        self.repo.get_booking_details(bid)

    elif command == "exit":
        print("👋 Goodbye!")
        break

    else:
        print("❌ Invalid command.")

except EventNotFoundException as e:
    print("❗", e)
except InvalidBookingIDException as e:
    print("❗", e)
except Exception as e:
    print("❌ Error:", e)

# Run the system
if __name__ == "__main__":
```

```
app = TicketBookingSystem()  
app.run()
```

The screenshot shows two instances of the PyCharm IDE side-by-side, both displaying the same Python file, `ticket_booking_system.py`. The project structure on the left of both editors shows a folder named `TicketBookingSystem` containing several subfolders like `PyCharmMiscProject`, `.venv`, `app`, `bean`, `entity`, `exception`, `repository`, `service`, and `util`. The main code area in the top editor (lines 1-26) defines a `TicketBookingSystem` class with an `__init__` method that initializes a repository, and a `run` method that prints a welcome message and a list of commands (create\_event, book\_tickets, cancel\_tickets, get\_available\_seats, get\_event\_details, get\_booking\_details, exit). It then enters a loop where it takes user input and tries to execute the command. The bottom editor (lines 8-50) continues the `run` method implementation, showing how it handles the "create\_event" command by prompting for event details (name, date, time, seats, price, type, venue name, address), creating a `Venue` object, and calling the repository's `create_event` method. It then handles the "book\_tickets" command by prompting for event name, number of tickets, and customer details (name, email, phone), creating `Customer` objects, and calling the repository's `book_tickets` method.

```
# app/ticket_booking_system.py  
from repository.booking_system_repository_impl import BookingSystemRepositoryImpl  
from entity.venue import Venue  
from entity.customer import Customer  
from entity.custom_exceptions import EventNotFoundException, InvalidBookingIDException  
  
class TicketBookingSystem: 3 usages  
    def __init__(self):  
        self.repo = BookingSystemRepositoryImpl()  
  
    def run(self): 2 usages  
        print("■■ Welcome 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. get_booking_details")  
        print("7. exit")  
  
        while True:  
            command = input("\nEnter command: ").strip().lower()  
  
            try:  
  
                if command == "create_event":  
                    name = input("Event name: ")  
                    date = input("Date (YYYY-MM-DD): ")  
                    time = input("Time (HH:MM): ")  
                    seats = int(input("Total seats: "))  
                    price = float(input("Ticket price: "))  
                    type_ = input("Event type (Movie, Concert, Sports): ")  
                    vname = input("Venue name: ")  
                    vaddr = input("Venue address: ")  
                    venue = Venue(vname, vaddr)  
  
                    self.repo.create_event(name, date, time, seats, price, type_, venue)  
  
                elif command == "book_tickets":  
                    event_name = input("Event name to book: ")  
                    num = int(input("How many tickets? "))  
                    customers = []  
                    for i in range(num):  
                        print(f"\nCustomer {i+1}")  
                        name = input("Name: ")  
                        email = input("Email: ")  
                        phone = input("Phone: ")  
                        customers.append(Customer(name, email, phone))  
                    self.repo.book_tickets(event_name, num, customers)  
  
            except EventNotFoundException as e:  
                print(f"Error: {e}")  
  
            except InvalidBookingIDException as e:  
                print(f"Error: {e}")
```

```
8     class TicketBookingSystem: 3 usages
12         def run(self): 2 usages
13             if command == "cancel_tickets":
14                 bid = int(input("Booking ID to cancel: "))
15                 self.repo.cancel_booking(bid)
16
17             elif command == "get_available_seats":
18                 self.repo.get_available_no_of_tickets()
19
20             elif command == "get_event_details":
21                 self.repo.get_event_details()
22
23             elif command == "get_booking_details":
24                 bid = int(input("Booking ID to view: "))
25                 self.repo.get_booking_details(bid)
26
27             elif command == "exit":
28                 print("👋 Goodbye!")
29                 break
30
31             else:
32                 print("🔴 Invalid command.")
33
34         except EventNotFoundException as e:
35             print("❗", e)
36
37         except InvalidBookingIDException as e:
38             print("❗", e)
39
40         except Exception as e:
41             print("🔴 Error:", e)
42
43
44     # Run the system
45
46     if __name__ == "__main__":
47         app = TicketBookingSystem()
48         app.run()
```

Output :

From ticket booking system

1. Create event details

```
Run ticket_booking_system ×
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\app\ticket_booking_system.py
■■ 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. get_booking_details
  7. exit

➤ Enter command: create_event
Event name: Rock Nigh
Date (YYYY-MM-DD): 2025-07-10

Run ticket_booking_system ×
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\app\ticket_booking_system.py
➤ Event name: Rock Nigh
Event date: 2025-07-10
➤ Time (HH:MM): 19:00
➤ Total seats: 50
➤ Ticket price: 299.99
➤ Event type (Movie, Concert, Sports): Concert
➤ Venue name: Chennai Arena
➤ Venue address: Nungambakkam, Chennai
✓ Event saved to database.

➤ Enter command: |
PyCharmMiscProject > app > ticket_booking_system.py
10:50  CRLF  UTF-8  4 spaces  Py
```

## 2. Booking tickets for events

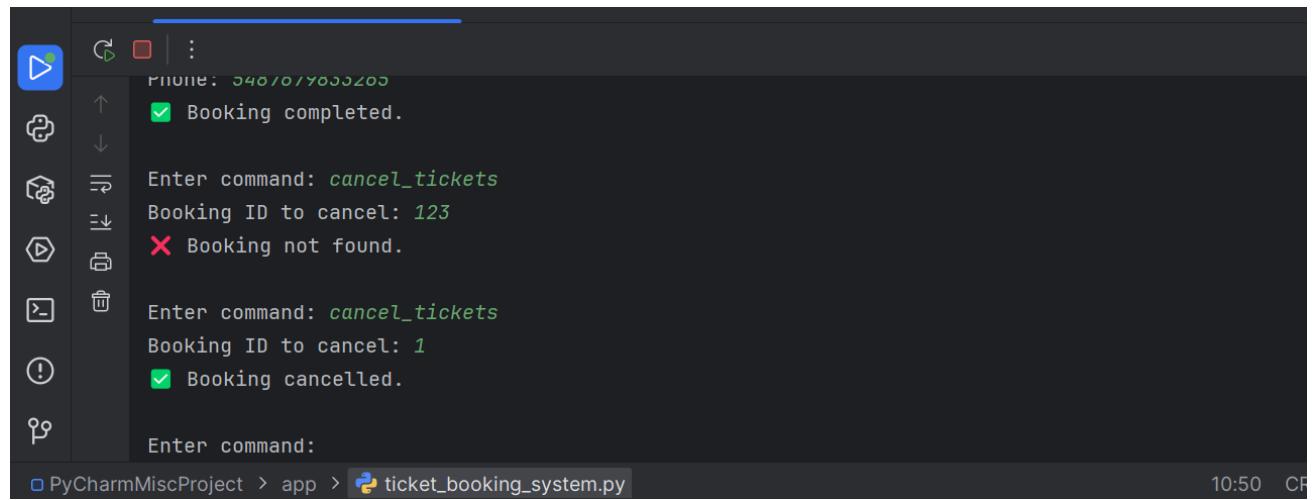
```
Run ticket_booking_system ×
C:\Users\machi\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\machi\PyCharmMiscProject\app\ticket_booking_system.py
✓ Event saved to database.

Enter command: book_tickets
Event name to book: Rock Nigh
How many tickets? 2

➤ Customer 1
Name: dhamini
Email: dhamini@gmail.com
Phone: 98667564573653

➤ Customer 2
Name: yamini
Email: yamini@gmail.com
Phone: 5487679833265
✓ Booking completed.

➤ Enter command:
PyCharmMiscProject > app > ticket_booking_system.py
10:50  CRLF  UTF-8  4 spaces
```



PyCharm interface showing the output of a Python script. The terminal window displays the following text:

```
Phone: 9480010000
✓ Booking completed.

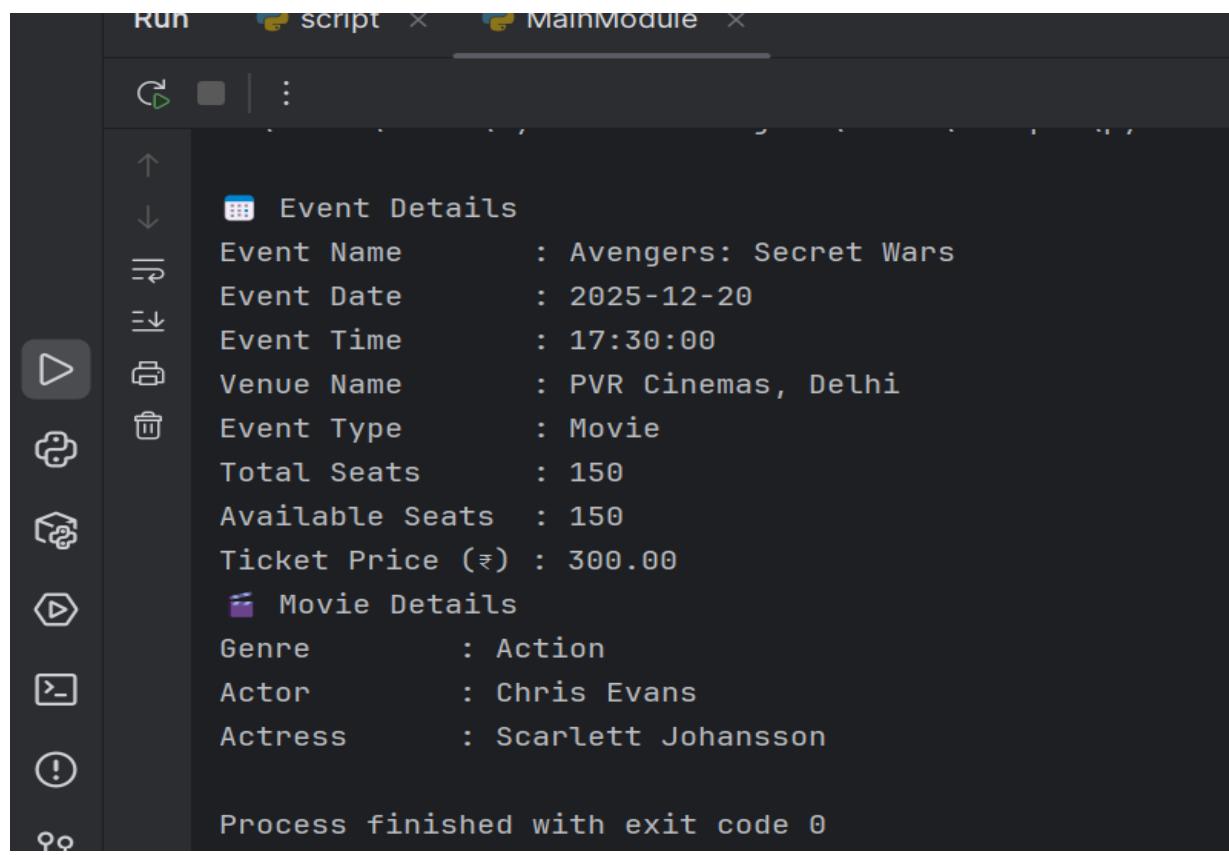
Enter command: cancel_tickets
Booking ID to cancel: 123
✗ Booking not found.

Enter command: cancel_tickets
Booking ID to cancel: 1
✓ Booking cancelled.

Enter command:
```

The status bar at the bottom shows the project name "PyCharmMiscProject", the file path "app/ticket\_booking\_system.py", and the time "10:50 CR".

3. Checking bookid details to cancel



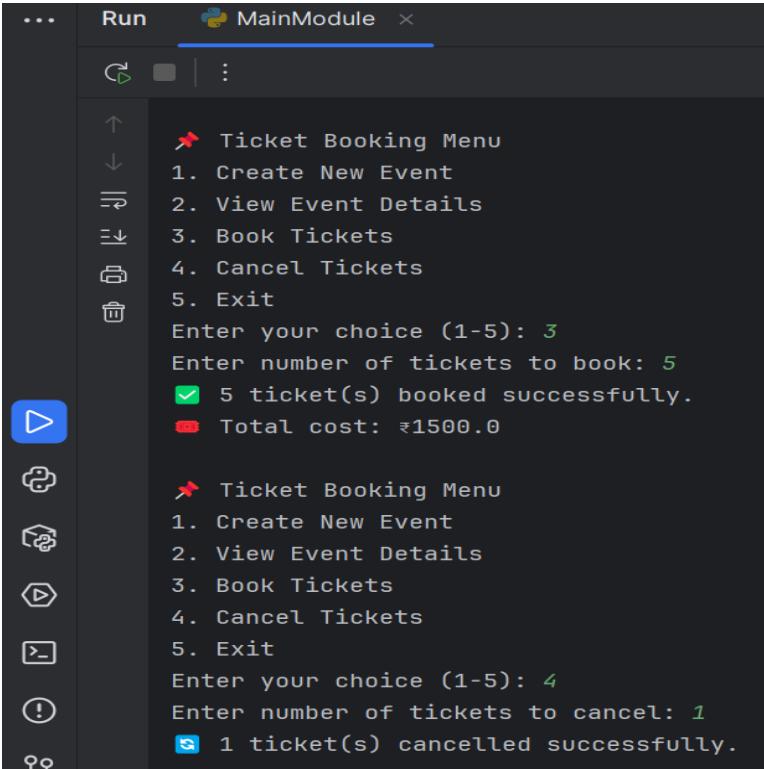
PyCharm interface showing the output of a Python script. The terminal window displays the following text:

```
Run script MainModule
Event Details
Event Name      : Avengers: Secret Wars
Event Date     : 2025-12-20
Event Time      : 17:30:00
Venue Name      : PVR Cinemas, Delhi
Event Type      : Movie
Total Seats     : 150
Available Seats : 150
Ticket Price (₹) : 300.00
Movie Details
Genre          : Action
Actor          : Chris Evans
Actress         : Scarlett Johansson

Process finished with exit code 0
```

4. Getting event details

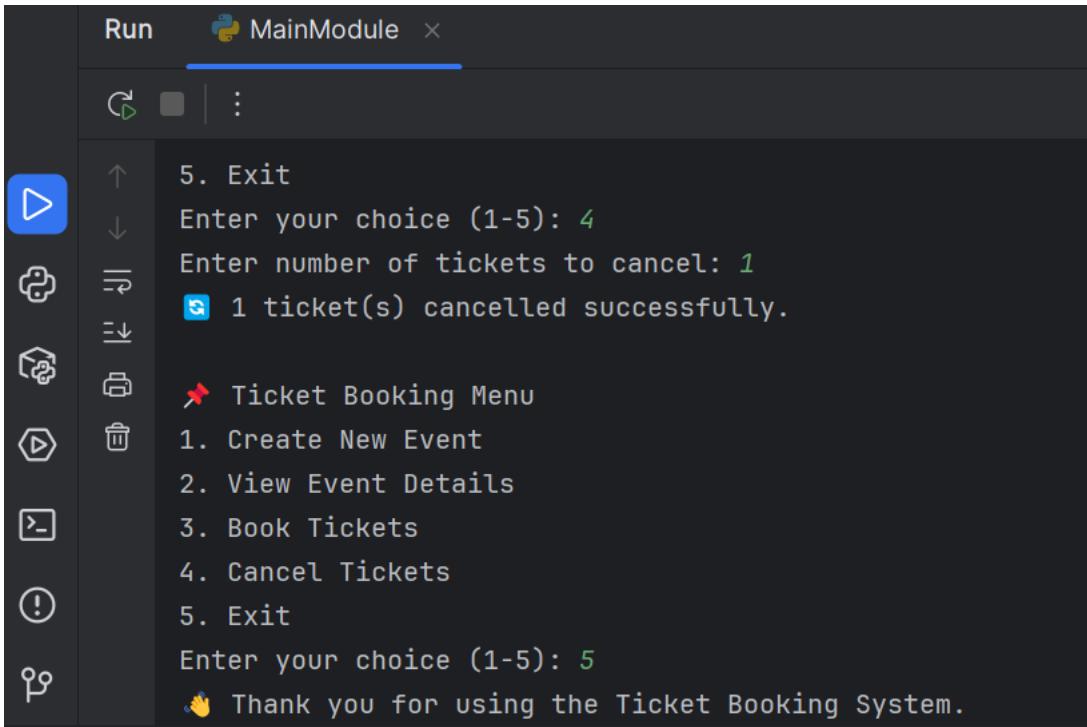
5. Ticket booked successfully and 1 ticket canceled



```
... Run MainModule ×
... | : 
↑ ↗ Ticket Booking Menu
↓ 1. Create New Event
→ 2. View Event Details
→ 3. Book Tickets
→ 4. Cancel Tickets
→ 5. Exit
Enter your choice (1-5): 3
Enter number of tickets to book: 5
✓ 5 ticket(s) booked successfully.
₹ Total cost: ₹1500.0

↗ Ticket Booking Menu
1. Create New Event
2. View Event Details
3. Book Tickets
4. Cancel Tickets
5. Exit
Enter your choice (1-5): 4
Enter number of tickets to cancel: 1
✗ 1 ticket(s) cancelled successfully.
```

6. Exiting the ticket system



```
... Run MainModule ×
... | : 
↑ 5. Exit
Enter your choice (1-5): 4
Enter number of tickets to cancel: 1
✗ 1 ticket(s) cancelled successfully.

↗ Ticket Booking Menu
1. Create New Event
2. View Event Details
3. Book Tickets
4. Cancel Tickets
5. Exit
Enter your choice (1-5): 5
👋 Thank you for using the Ticket Booking System.
```