

Assignment-5

Control structure

Task 1: Conditional Statements

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

Tasks:

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

```
def book_tickets(available_tickets, num_booking_tickets):  
    if available_tickets >= num_booking_tickets:  
        remaining_tickets = available_tickets - num_booking_tickets  
        print(f"Tickets booked successfully! Remaining tickets: {remaining_tickets}")  
    else:  
        print("Sorry, tickets unavailable.")  
  
available_tickets = int(input("Enter the number of available tickets: "))  
num_booking_tickets = int(input("Enter the number of tickets to book: "))  
book_tickets(available_tickets, num_booking_tickets)  
  
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.

```

12     def calculate_ticket_cost(ticket_type, num_tickets):
13         silver_price = 50.0
14         gold_price = 100.0
15         diamond_price = 150.0
16
17         if ticket_type == "Silver":
18             total_cost = silver_price * num_tickets
19         elif ticket_type == "Gold":
20             total_cost = gold_price * num_tickets
21         elif ticket_type == "Diamond":
22             total_cost = diamond_price * num_tickets
23         else:
24             print("Invalid ticket type. Please choose from Silver, Gold, or Diamond.")
25             return None
26
27     return total_cost
28
29
30 ticket_type = input("Enter the ticket type (Silver/Gold/Diamond): ")
31 num_tickets = int(input("Enter the number of tickets to book: "))
32 total_cost = calculate_ticket_cost(ticket_type, num_tickets)
33 if total_cost is not None:
34     print(f"Tickets booked successfully! Total cost: ${total_cost:.2f}")
35

```

Enter the ticket type (Silver/Gold/Diamond): Silver

Enter the number of tickets to book: 2

Tickets booked successfully! Total cost: \$100.00

Process finished with exit code 0

Task 3: Looping

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

```

39     def calculate_ticket_cost(ticket_type, num_tickets):
40         silver_price = 50.0
41         gold_price = 100.0
42         diamond_price = 150.0
43
44         if ticket_type == "Silver":
45             total_cost = silver_price * num_tickets
46         elif ticket_type == "Gold":
47             total_cost = gold_price * num_tickets
48         elif ticket_type == "Diamond":
49             total_cost = diamond_price * num_tickets
50         else:
51             print("Invalid ticket type. Please choose from Silver, Gold, or Diamond.")
52             return None
53
54     return total_cost
55
56 while True:
57     ticket_type = input("Enter the ticket type (Silver/Gold/Diamond) or type 'Exit' to stop: ")
58
59     if ticket_type.lower() == "exit":
60         print("Exiting the ticket booking system.")
61         break
62
63     num_tickets = int(input("Enter the number of tickets to book: "))
64     total_cost = calculate_ticket_cost(ticket_type, num_tickets)
65
66     if total_cost is not None:
67         print(f"Tickets booked successfully! Total cost: ${total_cost:.2f}")

```

```
Enter the ticket type (Silver/Gold/Diamond) or type 'Exit' to stop: silver
Enter the number of tickets to book: 1
Tickets booked successfully! Total cost: $50.00
Enter the ticket type (Silver/Gold/Diamond) or type 'Exit' to stop: gold
Enter the number of tickets to book: 2
Tickets booked successfully! Total cost: $200.00
Enter the ticket type (Silver/Gold/Diamond) or type 'Exit' to stop: exit
Exiting 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

```
70     from datetime import date, time, datetime
71
72     from enum import Enum
73
74     class EventType(Enum):
75         MOVIE = 'Movie'
76         SPORTS = 'Sports'
77         CONCERT = 'Concert'
78
79     class Event:
80         def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type):
81             self.event_name = event_name
82             self.event_date = event_date
83             self.event_time = event_time
84             self.venue_name = venue_name
85             self.total_seats = total_seats
86             self.available_seats = total_seats
87             self.ticket_price = ticket_price
88             self.event_type = event_type
89             self.booked_tickets = 0
90
91         # Getter and Setter methods
92         def get_event_name(self):
93             return self.event_name
94
95         def set_event_name(self, event_name):
96             self.event_name = event_name
97
98         def get_event_date(self):
99             return self.event_date
```

```
100        def set_event_date(self, event_date):
101            self.event_date = event_date
102
103        def get_event_time(self):
104            return self.event_time
105            Parameter self of main.Event.get_event_time
106            self: Event
107
108        def set_event_time(self, event_time):
109            self.event_time = event_time
110
111        def get_venue_name(self):
112            return self.venue_name
113
114        def set_venue_name(self, venue_name):
115            self.venue_name = venue_name
116
117        def get_total_seats(self):
118            return self.total_seats
119
120        def set_total_seats(self, total_seats):
121            self.total_seats = total_seats
122
123        def get_available_seats(self):
124            return self.available_seats
125
126        def set_available_seats(self, available_seats):
127            self.available_seats = available_seats
128
129        def get_ticket_price(self):
130            return self.ticket_price
```

```

156     def set_ticket_price(self, ticket_price):
157         self.ticket_price = ticket_price
158
159     def get_event_type(self):
160         return self.event_type
161
162     def set_event_type(self, event_type):
163         self.event_type = event_type
164
165     def display_event_details(self):
166         print("Event Details:")
167         print(f"Event Name: {self.event_name}")
168         print(f"Event Date: {self.event_date}")
169         print(f"Event Time: {self.event_time}")
170         print(f"Venue Name: {self.venue_name}")
171         print(f"Total Seats: {self.total_seats}")
172         print(f"Available Seats: {self.available_seats}")
173         print(f"Ticket Price: ${self.ticket_price:.2f}")
174         print(f"Event Type: {self.event_type}")
175
176     def calculate_total_revenue(self):
177         return self.ticket_price * self.booked_tickets
178
179     def get_booked_no_of_tickets(self):
180         return self.booked_tickets
181
182     def book_tickets(self, num_tickets):
183         if num_tickets > 0 and num_tickets <= self.available_seats:
184             self.booked_tickets += num_tickets
185             self.available_seats -= num_tickets
186

```

```

156     def book_tickets(self, num_tickets):
157         if num_tickets > 0 and num_tickets <= self.available_seats:
158             self.booked_tickets += num_tickets
159             self.available_seats -= num_tickets
160             print(f"{num_tickets} tickets booked successfully!")
161         else:
162             print("Invalid number of tickets or not enough available seats.")
163
164     def cancel_booking(self, num_tickets):
165         if num_tickets > 0 and num_tickets <= self.booked_tickets:
166             self.booked_tickets -= num_tickets
167             self.available_seats += num_tickets
168             print(f"{num_tickets} tickets canceled successfully!")
169         else:
170             print("Invalid number of tickets or not enough booked tickets.")


```

```

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

```

2. Venue Class

- **Attributes:**

- venue_name,

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

```

194     from datetime import date, time
195     from enum import Enum
196
197     class EventType(Enum):
198         MOVIE = 'Movie'
199         SPORTS = 'Sports'
200         CONCERT = 'Concert'
201
202     class Venue:
203         def __init__(self, venue_name, address):
204             self.venue_name = venue_name
205             self.address = address
206             # Getter and Setter methods
207             def get_venue_name(self):
208                 return self.venue_name
209
210             def set_venue_name(self, venue_name):
211                 self.venue_name = venue_name
212
213             def get_address(self):
214                 return self.address
215
216             def set_address(self, address):
217                 self.address = address
218
219             def display_venue_details(self):
220                 print("Venue Details:")
221                 print(f"Venue Name: {self.venue_name}")
222                 print(f"Address: {self.address}")

```

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.

```
224     class Customer:
225         def __init__(self, customer_name, email, phone_number):
226             self.customer_name = customer_name
227             self.email = email
228             self.phone_number = phone_number
229
230         # Getter and Setter methods
231         def get_customer_name(self):
232             return self.customer_name
233
234         def set_customer_name(self, customer_name):
235             self.customer_name = customer_name
236
237         def get_email(self):
238             return self.email
239
240         def set_email(self, email):
241             self.email = email
242
243         def get_phone_number(self):
244             return self.phone_number
245
246         def set_phone_number(self, phone_number):
247             self.phone_number = phone_number
248
249     def display_customer_details(self):
250         print("Customer Details:")
251         print(f"Customer Name: {self.customer_name}")
252         print(f"Email: {self.email}")
253         print(f"Phone Number: {self.phone_number}")
```

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

- **Methods and 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.

return the total available tickets

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

```

255     class Booking:
256         def __init__(self, event, customer, num_tickets):
257             self.event = event
258             self.customer = customer
259             self.num_tickets = num_tickets
260             self.total_cost = 0.0
261
262         def calculate_booking_cost(self):
263             self.total_cost = self.event.ticket_price * self.num_tickets
264
265         def book_tickets(self):
266             if self.num_tickets > 0 and self.num_tickets <= self.event.available_seats:
267                 self.event.book_tickets(self.num_tickets)
268                 self.calculate_booking_cost()
269                 print(f"{self.num_tickets} tickets booked successfully! Total cost: ${self.total_cost:.2f}")
270             else:
271                 print("Invalid number of tickets or not enough available seats.")
272
273         def cancel_booking(self):
274             if self.num_tickets > 0 and self.num_tickets <= self.event.booked_tickets:
275                 self.event.cancel_booking(self.num_tickets)
276                 self.calculate_booking_cost() # Update total cost after canceling tickets
277                 print(f"{self.num_tickets} tickets canceled successfully! Total cost: ${self.total_cost:.2f}")
278             else:
279                 print("Invalid number of tickets or not enough booked tickets.")
280
281         def get_available_no_of_tickets(self):
282             return self.event.available_seats
283
284         def get_event_details(self):
285             return self.event.display_event_details()

```

Task 5: Inheritance and polymorphism

1. Inheritance

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

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

- **Attributes:**

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.

```
120     class EventType(Enum):
121         MOVIE = 'Movie'
122         SPORTS = 'Sports'
123         CONCERT = 'Concert'
124
125     class Movie(Event):
126         def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
127                      genre, actor_name, actress_name):
128             super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
129             self.genre = genre
130             self.actor_name = actor_name
131             self.actress_name = actress_name
132
133         def get_genre(self):
134             return self.genre
135
136         def set_genre(self, genre):
137             self.genre = genre
138
139         def get_actor_name(self):
140             return self.actor_name
141
142         def set_actor_name(self, actor_name):
143             self.actor_name = actor_name
144
145         def get_actress_name(self):
146             return self.actress_name
147
148         def set_actress_name(self, actress_name):
149             self.actress_name = actress_name
150
```

```
141
142         def set_actor_name(self, actor_name):
143             self.actor_name = actor_name
144
145         def get_actress_name(self):
146             return self.actress_name
147
148         def set_actress_name(self, actress_name):
149             self.actress_name = actress_name
150
151         def display_event_details(self):
152             super().display_event_details()
153             print(f"Genre: {self.genre}")
154             print(f"Actor: {self.actor_name}")
155             print(f"Actress: {self.actress_name}")
156
157     if __name__ == "__main__":
158         movie_date = date(2023, 1, 15)
159         movie_time = time(20, 0)
160         movie_type = EventType.MOVIE
161         movie = Movie("Blockbuster Movie", movie_date, movie_time, "Cinema City", 500, 10.0, movie_type,
162                       "Action", "John Doe", "Jane Doe")
163
164         movie.display_event_details()
```

```
Event Details:  
Event Name: Blockbuster Movie  
Event Date: 2023-01-15  
Event Time: 20:00:00  
Venue Name: Cinema City  
Total Seats: 500  
Available Seats: 500  
Ticket Price: $10.00  
Event Type: Eventtype.MOVIE  
Genre: Action
```

- Create another subclass **Concert** that inherits from **Event**. Add the following attributes and methods:
 - **Attributes:**
 1. artist: Name of the performing artist or band.
 2. type: (Theatrical, Classical, Rock, Recital)
 - **Methods:**
 3. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
 4. **display_concert_details():** Display concert details, including the artist.

```
168 class Concert(Event):  
169     def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,  
170                  artist, concert_type):  
171         super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)  
172         self.artist = artist  
173         self.concert_type = concert_type  
174  
175     # Getter and Setter methods  
176     def get_artist(self):  
177         return self.artist  
178  
179     def set_artist(self, artist):  
180         self.artist = artist  
181  
182     def get_concert_type(self):  
183         return self.concert_type  
184  
185     def set_concert_type(self, concert_type):  
186         self.concert_type = concert_type  
187  
188     def display_event_details(self):  
189         super().display_event_details()  
190         print(f"Artist: {self.artist}")  
191         print(f"Concert Type: {self.concert_type}")  
192  
193 if __name__ == "__main__":  
194     concert_date = date(2023, 2, 20)  
195     concert_time = time(19, 30)  
196     concert_type = EventType.CONCERT  
197     concert = Concert("Live Concert", concert_date, concert_time, "Music Arena", 1000, 50.0, concert_type,  
198                           "Rock Band", "Rock")  
199     concert.display_event_details()
```

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

- o **Attributes:**

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

- o **Methods:**

1. Implement default constructors and overload the constructor with Customer

attributes, generate getter and setter methods.

2. **display_sport_details():** Display concert details, including the artist.

```
01 class Sports(Event):
02     def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
03                  sport_name, teams_name):
04         super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
05         self.sport_name = sport_name
06         self.teams_name = teams_name
07
08     # Getter and Setter methods
09     def get_sport_name(self):
10         return self.sport_name
11
12     def set_sport_name(self, sport_name):
13         self.sport_name = sport_name
14
15     def get_teams_name(self):
16         return self.teams_name
17
18     def set_teams_name(self, teams_name):
19         self.teams_name = teams_name
20
21
```

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

- o **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu_name:str):** Create a new event with the specified details

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

- o **display_event_details(event: Event):** Accepts an event object and calls its

display_event_details() method to display event details. o **book_tickets(event: Event, num_tickets: int):**

1. Accepts an event object and the number of tickets to be booked.
2. Checks if there are enough available seats for the booking.
3. If seats are available, updates the available seats and returns the total cost

of the booking.

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

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

an event.

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

1. User can book tickets and view the event details as per their choice in menu (movies, sports, concerts).
2. Display event details using the `display_event_details()` method without knowing the specific event type (demonstrate polymorphism).
3. Make bookings using the `book_tickets()` and cancel tickets `cancel_tickets()` method.

```
443     from datetime import date, time
444
445     class EventType:
446         MOVIE = 'Movie'
447         SPORTS = 'Sports'
448         CONCERT = 'Concert'
449
450     class Venue:
451         def __init__(self, venue_name, address):
452             self.venue_name = venue_name
453             self.address = address
454
455     class Event:
456         def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type):
457             self.event_name = event_name
458             self.event_date = event_date
459             self.event_time = event_time
460             self.venue_name = venue_name
461             self.total_seats = total_seats
462             self.available_seats = total_seats
463             self.ticket_price = ticket_price
464             self.event_type = event_type
465
466         def display_event_details(self):
467             print("Event Details:")
468             print(f"Event Name: {self.event_name}")
469             print(f"Event Date: {self.event_date}")
470             print(f"Event Time: {self.event_time}")
471             print(f"Venue Name: {self.venue_name}")
472             print(f"Total Seats: {self.total_seats}")
473             print(f"Available Seats: {self.available_seats}")
474             print(f"Ticket Price: ${self.ticket_price:.2f}")
475             print(f"Event Type: {self.event_type}")
476
477         def book_tickets(self, num_tickets):
478             if 0 < num_tickets <= self.available_seats:
479                 self.available_seats -= num_tickets
480                 return self.ticket_price * num_tickets
481             else:
482                 print("Not enough available seats.")
483                 return 0.0
```

```
485     def cancel_tickets(self, num_tickets):
486         if 0 < num_tickets <= (self.total_seats - self.available_seats):
487             self.available_seats += num_tickets
488             return self.ticket_price * num_tickets
489         else:
490             print("Invalid number of tickets to cancel.")
491             return 0.0
492
493 class TicketBookingSystem:
494     def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue_name):
495         event_date = date(map(int, date.split('-')))
496         event_time = time(map(int, time.split(':')))
497         return Event(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
498
499     def display_event_details(self, event):
500         event.display_event_details()
501
502     def book_tickets(self, event, num_tickets):
503         return event.book_tickets(num_tickets)
504
505     def cancel_tickets(self, event, num_tickets):
506         return event.cancel_tickets(num_tickets)
507
508 def main():
509     ticket_booking_system = TicketBookingSystem()
510
511     while True:
512         print("\nMenu:")
513         print("1. Create Event")
514         print("2. Display Event Details")
515         print("3. Book Tickets")
516         print("4. Cancel Tickets")
517         print("5. Exit")
518
519         choice = input("Enter your choice (1-5): ")
```

```

521     if choice == "1":
522         event_name = input("Enter event name: ")
523         date = input("Enter event date (YYYY-MM-DD): ")
524         time_str = input("Enter event time (HH:MM): ")
525         total_seats = int(input("Enter total seats: "))
526         ticket_price = float(input("Enter ticket price: "))
527         event_type = input("Enter event type (Movie/Sports/Concert): ")
528         venue_name = input("Enter venue name: ")
529
530         event = ticket_booking_system.create_event(event_name, date, time_str, total_seats, ticket_price, event_type, venue_name)
531         print("Event created successfully!")
532
533     elif choice == "2":
534         if 'event' in locals():
535             ticket_booking_system.display_event_details(event)
536         else:
537             print("No event created yet. Please create an event first.")
538
539     elif choice == "3":
540         if 'event' in locals():
541             num_tickets = int(input("Enter the number of tickets to book: "))
542             total_cost = ticket_booking_system.book_tickets(event, num_tickets)
543             if total_cost > 0:
544                 print(f"Tickets booked successfully! Total cost: ${total_cost:.2f}")
545             else:
546                 print("No event created yet. Please create an event first.")
547
548     elif choice == "4":
549         if 'event' in locals():
550             num_tickets = int(input("Enter the number of tickets to cancel: "))
551             total_refund = ticket_booking_system.cancel_tickets(event, num_tickets)
552             if total_refund > 0:
553                 print(f"Tickets canceled successfully! Total refund: ${total_refund:.2f}")
554             else:
555                 print("No event created yet. Please create an event first.")
556
557     elif choice == "5":
558         print("Exiting the Ticket Booking System.")
559         break
560
561     else:
562         print("Invalid choice. Please enter a number between 1 and 5.")
563
564     if name == "main__":
565         main()
566
567

```

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

```
↑ 3. Book Tickets
↓ 4. Cancel Tickets
5. Exit
→ Enter your choice (1-5): 1
📝 Enter event name: ABC Event
📅 Enter event date (YYYY-MM-DD): 2024-01-01
🕒 Enter event time (HH:MM): 09:00
Seats Enter total seats: 50
Enter ticket price: 20
Enter event type (Movie/Sports/Concert): Concert
Enter venue name: ABC
```

Task 6: Abstraction Requirements:

Event Abstraction:

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

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

2. Concrete Event Classes:

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

- Movie.
- Concert.
- Sport.

```
569 from abc import ABC, abstractmethod
570 from enum import Enum
571
572 class EventType(Enum):
573     MOVIE = 'Movie'
574     SPORTS = 'Sports'
575     CONCERT = 'Concert'
576
577 @l class Event(ABC):
578     def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type):
579         self.event_name = event_name
580         self.event_date = event_date
581         self.event_time = event_time
582         self.venue_name = venue_name
583         self.total_seats = total_seats
584         self.available_seats = total_seats
585         self.ticket_price = ticket_price
586         self.event_type = event_type
587
588     @abstractmethod
589     def display_event_details(self):
590         pass
591
592     @abstractmethod
593     def book_tickets(self, num_tickets):
594         pass
595
596     @abstractmethod
597     def cancel_tickets(self, num_tickets):
598         pass
```

```
599     class Movie(Event):
600         def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
601                      genre, actor_name, actress_name):
602             super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
603             self.genre = genre
604             self.actor_name = actor_name
605             self.actress_name = actress_name
606
607         def display_event_details(self):
608             super().display_event_details()
609             print(f"Genre: {self.genre}")
610             print(f"Actor: {self.actor_name}")
611             print(f"Actress: {self.actress_name}")
612
613         def book_tickets(self, num_tickets):
614             if 0 < num_tickets <= self.available_seats:
615                 self.available_seats -= num_tickets
616                 return self.ticket_price * num_tickets
617             else:
618                 print("Not enough available seats.")
619                 return 0.0
620
621         def cancel_tickets(self, num_tickets):
622             if 0 < num_tickets <= (self.total_seats - self.available_seats):
623                 self.available_seats += num_tickets
624                 return self.ticket_price * num_tickets
625             else:
626                 print("Invalid number of tickets to cancel.")
627                 return 0.0
628
629
```

```
30     class Concert(Event):
31         def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
32                      artist, concert_type):
33             super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
34             self.artist = artist
35             self.concert_type = concert_type
36
37     def display_event_details(self):
38         super().display_event_details()
39         print(f"Artist: {self.artist}")
40         print(f"Concert Type: {self.concert_type}")
41
42     def book_tickets(self, num_tickets):
43         if 0 < num_tickets <= self.available_seats:
44             self.available_seats -= num_tickets
45             return self.ticket_price * num_tickets
46         else:
47             print("Not enough available seats.")
48             return 0.0
49
50     def cancel_tickets(self, num_tickets):
51         if 0 < num_tickets <= (self.total_seats - self.available_seats):
52             self.available_seats += num_tickets
53             return self.ticket_price * num_tickets
54         else:
55             print("Invalid number of tickets to cancel.")
56             return 0.0
57
58     class Sports(Event):
59         def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
60                      sport_name, teams_name):
61             super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
62             self.sport_name = sport_name
63             self.teams_name = teams_name
64
65         def display_event_details(self):
66             super().display_event_details()
67             print(f"Sport Name: {self.sport_name}")
68             print(f"Teams: {self.teams_name}")
69
70         def book_tickets(self, num_tickets):
71             if 0 < num_tickets <= self.available_seats:
72                 self.available_seats -= num_tickets
73                 return self.ticket_price * num_tickets
74             else:
75                 print("Not enough available seats.")
76                 return 0.0
77
78         def cancel_tickets(self, num_tickets):
79             if 0 < num_tickets <= (self.total_seats - self.available_seats):
80                 self.available_seats += num_tickets
81                 return self.ticket_price * num_tickets
82             else:
83                 print("Invalid number of tickets to cancel.")
84                 return 0.0
```

3. BookingSystem Abstraction:

- Create an abstract class **BookingSystem** that represents the ticket booking system. It should

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

```
694     from abc import ABC, abstractmethod
695
696     class BookingSystem(ABC):
697         @abstractmethod
698         def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue_name):
699             pass
700
701         @abstractmethod
702         def display_event_details(self, event):
703             pass
704
705         @abstractmethod
706         def book_tickets(self, event, num_tickets):
707             pass
708
709         @abstractmethod
710         def cancel_tickets(self, event, num_tickets):
711             pass
712
713     class TicketBookingSystem(BookingSystem):
714         def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue_name):
715             pass
716
717         def display_event_details(self, event):
718             pass
719
720         def book_tickets(self, event, num_tickets):
721             pass
722
723         def cancel_tickets(self, event, num_tickets):
724             pass
725
726 if __name__ == "__main__":
727     ticket_booking_system = TicketBookingSystem()
```

. Concrete **TicketBookingSystem** Class:

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

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

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

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

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

```
781 class TicketBookingSystem:
782     def __init__(self):
783         self.events = []
784
785     def create_event(self, event_name, date, time_str, total_seats, ticket_price, event_type, venue_name):
786         event_date = date(map(int, date.split('-')))
787         event_time = time(map(int, time_str.split(':')))
788         event = Event(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
789         self.events.append(event)
790
791     def display_event_details(self, event):
792         event.display_event_details()
793
794     def book_tickets(self, event, num_tickets):
795         return event.book_tickets(num_tickets)
796
797     def cancel_tickets(self, event, num_tickets):
798         return event.cancel_tickets(num_tickets)
799
800     def get_available_seats(self, event):
801         return event.available_seats
802
803
804 if __name__ == "__main__":
805     ticket_booking_system = TicketBookingSystem()
```

```
806
807 if __name__ == "__main__":
808     ticket_booking_system = TicketBookingSystem()
809
810     while True:
811         print("\nMenu:")
812         print("1. Create Event")
813         print("2. Display Event Details")
814         print("3. Book Tickets")
815         print("4. Cancel Tickets")
816         print("5. Get Available Seats")
817         print("6. Exit")
818
819         choice = input("Enter your choice (1-6): ")
820
821         if choice == "1":
822             event_name = input("Enter event name: ")
823             date = input("Enter event date (YYYY-MM-DD): ")
824             time_str = input("Enter event time (HH:MM): ")
825             total_seats = int(input("Enter total seats: "))
826             ticket_price = float(input("Enter ticket price: "))
827             event_type_str = input("Enter event type (Movie/Concert/Sports): ")
828             venue_name = input("Enter venue name: ")
829
830             event = ticket_booking_system.create_event(event_name, date, time_str, total_seats, ticket_price, event_type_str, venue_name)
831             print("Event created successfully!")
832
833         elif choice == "2":
834             event_index = int(input("Enter the index of the event to display details: "))
835             if 0 <= event_index < len(ticket_booking_system.events):
836                 ticket_booking_system.display_event_details(ticket_booking_system.events[event_index])
```

```
836         else:
837             print("Invalid event index.")
838
839     elif choice == "3":
840         event_index = int(input("Enter the index of the event to book tickets: "))
841         if 0 <= event_index < len(ticket_booking_system.events):
842             num_tickets = int(input("Enter the number of tickets to book: "))
843             total_cost = ticket_booking_system.book_tickets(ticket_booking_system.events[event_index], num_tickets)
844             if total_cost > 0:
845                 print(f"Tickets booked successfully! Total cost: ${total_cost:.2f}")
846             else:
847                 print("Invalid event index.")
848
849     elif choice == "4":
850         event_index = int(input("Enter the index of the event to cancel tickets: "))
851         if 0 <= event_index < len(ticket_booking_system.events):
852             num_tickets = int(input("Enter the number of tickets to cancel: "))
853             total_refund = ticket_booking_system.cancel_tickets(ticket_booking_system.events[event_index], num_tickets)
854             if total_refund > 0:
855                 print(f"Tickets canceled successfully! Total refund: ${total_refund:.2f}")
856             else:
857                 print("Invalid event index.")
858
859     elif choice == "5":
860         event_index = int(input("Enter the index of the event to get available seats: "))
861         if 0 <= event_index < len(ticket_booking_system.events):
862             available_seats = ticket_booking_system.get_available_seats(ticket_booking_system.events[event_index])
863             print(f"Available seats: {available_seats}")
864         else:
865             print("Invalid event index.")
866
```

```
866
867         elif choice == "6":
868             print("Exiting the Ticket Booking System.")
869             break
```

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

6. Exit

Enter your choice (1-6): **6**

Exiting the Ticket Booking System.

Process finished with exit code 0

```
Run main ...
4. Cancel Tickets
5. Get Available Seats
6. Exit

Enter your choice (1-6): 1
Enter event name: event
Enter event date (YYYY-MM-DD): 2024-01-01
Enter event time (HH:MM): 09:00
Enter total seats: 100
Enter ticket price: 10
Enter event type (Movie/Concert/Sports): Sports
Enter venue name: 
```

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.

```
871 class Venue:
872     def __init__(self, venue_name, address):
873         self.venue_name = venue_name
874         self.address = address
875
876     def display_venue_details(self):
877         print("Venue Details:")
878         print(f"Venue Name: {self.venue_name}")
879         print(f"Address: {self.address}")
880
881     def get_venue_name(self):
882         return self.venue_name
883
884     def set_venue_name(self, venue_name):
885         self.venue_name = venue_name
886
887     def get_address(self):
888         return self.address
889
890     def set_address(self, address):
891         self.address = address
```

Event Class:

- Attributes:

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

- **Methods and 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.

```
904     from enum import Enum
905
906     from datetime import date, time
907
907     class Event:
908         def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, event_type):
909             self.event_name = event_name
910             self.event_date = event_date
911             self.event_time = event_time
912             self.venue = venue
913             self.total_seats = total_seats
914             self.available_seats = total_seats
915             self.ticket_price = ticket_price
916             self.event_type = event_type
917             self.booked_tickets = 0
918
919
920         def calculate_total_revenue(self):
921             return self.ticket_price * self.booked_tickets
922
923         def get_booked_no_of_tickets(self):
924             return self.booked_tickets
925
926         def book_tickets(self, num_tickets):
927             if 0 < num_tickets <= self.available_seats:
928                 self.available_seats -= num_tickets
929                 self.booked_tickets += num_tickets
930                 return True
931             else:
932                 print("Not enough available seats.")
933                 return False
934
935
936         def cancel_booking(self, num_tickets):
937             if 0 < num_tickets <= self.booked_tickets:
938                 self.available_seats += num_tickets
939                 self.booked_tickets -= num_tickets
940                 return True
941             else:
942                 print("Invalid number of tickets to cancel.")
943                 return False
944
```

3. Event sub classes:

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

- o Movie.

- o Concert.

- o Sport.

```
10 class Movie(Event):
11     def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, genre, actor_name, actress_name):
12         super().__init__(event_name, event_date, event_time, venue, total_seats, ticket_price, genre, EventType.MOVIE)
13         self.genre = genre
14         self.actor_name = actor_name
15         self.actress_name = actress_name
16
17
18
19 class Concert(Event):
20     def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, artist, concert_type):
21         super().__init__(event_name, event_date, event_time, venue, total_seats, ticket_price, artist, EventType.CONCERT)
22         self.artist = artist
23         self.concert_type = concert_type
24
25
26 class Sport(Event):
27     def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, sport_name, teams_name):
28         super().__init__(event_name, event_date, event_time, venue, total_seats, ticket_price, sport_name, EventType.SPORTS)
29         self.sport_name = sport_name
30         self.teams_name = teams_name
```

4. Customer Class

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

```

10     class Customer:
11         def __init__(self, customer_name, email, phone_number):
12             self.customer_name = customer_name
13             self.email = email
14             self.phone_number = phone_number
15
16         def display_customer_details(self):
17             print("Customer Details:")
18             print(f"Customer Name: {self.customer_name}")
19             print(f"Email: {self.email}")
20             print(f"Phone Number: {self.phone_number}")
21
22         # Getter and Setter methods
23         def get_customer_name(self):
24             return self.customer_name
25
26         def set_customer_name(self, customer_name):
27             self.customer_name = customer_name
28
29         def get_email(self):
30             return self.email
31
32         def set_email(self, email):
33             self.email = email
34
35         def get_phone_number(self):
36             return self.phone_number
37
38         def set_phone_number(self, phone_number):
39             self.phone_number = phone_number

```

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

- bookingId (should be incremented for each booking)
- array of customer (reference to the customer who made the booking)
- event (reference to the event booked)
- num_tickets(no of tickets and array of customer must equal)
- total_cost
- booking_date (timestamp of when the booking was made)
- **Methods and Constructors:**
 - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
 - **display_booking_details():** Display customer details.

```
121
122     from datetime import datetime
123
124     class Booking:
125         booking_id_counter = 1
126
127         def __init__(self, customers, event, num_tickets):
128             self.booking_id = Booking.booking_id_counter
129             Booking.booking_id_counter += 1
130
131             if len(customers) != num_tickets:
132                 raise ValueError("Number of customers and number of tickets must be equal.")
133
134             self.customers = customers
135             self.event = event
136             self.num_tickets = num_tickets
137             self.total_cost = 0.0
138             self.booking_date = datetime.now()
139
140         def calculate_total_cost(self):
141             self.total_cost = self.event.ticket_price * self.num_tickets
142
143
```

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."

```
149     from typing import List
150
151     class BookingSystem:
152         def __init__(self):
153             self.events = []
154
155         def create_event(self, event_name, date, time_str, total_seats, ticket_price, event_type, venue):
156             event_date = date(*map(int, date.split('-')))
157             event_time = time(*map(int, time_str.split(':')))
158             event = Event(event_name, event_date, event_time, venue, total_seats, ticket_price, event_type)
159             self.events.append(event)
160             return event
161
162         def calculate_booking_cost(self, num_tickets):
163             if num_tickets > 0:
164                 return num_tickets * self.selected_event.ticket_price
165             else:
166                 print("Invalid number of tickets.")
167                 return 0.0
168
169         def book_tickets(self, event_name, num_tickets, array_of_customers):
170             for _ in range(num_tickets):
171                 customer_name = input("Enter customer name: ")
172                 email = input("Enter email: ")
173                 phone_number = input("Enter phone number: ")
174                 customer = Customer(customer_name, email, phone_number)
175                 array_of_customers.append(customer)
176
177             total_cost = self.calculate_booking_cost(num_tickets)
178             booking = Booking(array_of_customers, self.selected_event, num_tickets)
179
180
181             total_cost = self.calculate_booking_cost(num_tickets)
182             booking = Booking(array_of_customers, self.selected_event, num_tickets)
183             booking.total_cost = total_cost
184
185             print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
186             booking.display_booking_details()
187
188         def cancel_booking(self, booking_id):
189             for event in self.events:
190                 for booking in event.bookings:
191                     if booking.booking_id == booking_id:
192                         event.available_seats += booking.num_tickets
193                         event.bookings.remove(booking)
194                         print(f"\nBooking with ID {booking_id} canceled successfully.")
195                         return
196
197             print(f"\nBooking with ID {booking_id} not found.")
198
199         def get_available_no_of_tickets(self):
200             return self.selected_event.available_seats
201
202         def get_event_details(self):
203             self.selected_event.display_event_details()
204
205         def display_menu(self):
206             print("\nMenu:")
207             print("1. Create Event")
208             print("2. Get Event Details")
209             print("3. Book Tickets")
210             print("4. Cancel Booking")
211             print("5. Get Available Seats")
```

```
177             total_cost = self.calculate_booking_cost(num_tickets)
178             booking = Booking(array_of_customers, self.selected_event, num_tickets)
179             booking.total_cost = total_cost
180
181             print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
182             booking.display_booking_details()
183
184         def cancel_booking(self, booking_id):
185             for event in self.events:
186                 for booking in event.bookings:
187                     if booking.booking_id == booking_id:
188                         event.available_seats += booking.num_tickets
189                         event.bookings.remove(booking)
190                         print(f"\nBooking with ID {booking_id} canceled successfully.")
191                         return
192
193             print(f"\nBooking with ID {booking_id} not found.")
194
195         def get_available_no_of_tickets(self):
196             return self.selected_event.available_seats
197
198         def get_event_details(self):
199             self.selected_event.display_event_details()
200
201         def display_menu(self):
202             print("\nMenu:")
203             print("1. Create Event")
204             print("2. Get Event Details")
205             print("3. Book Tickets")
206             print("4. Cancel Booking")
207             print("5. Get Available Seats")
```

```

209     def main(self):
210         while True:
211             self.display_menu()
212             choice = input("Enter your choice (1-6): ")
213
214             if choice == "1":
215                 event_name = input("Enter event name: ")
216                 date = input("Enter event date (YYYY-MM-DD): ")
217                 time_str = input("Enter event time (HH:MM): ")
218                 total_seats = int(input("Enter total seats: "))
219                 ticket_price = float(input("Enter ticket price: "))
220                 event_type_str = input("Enter event type (Movie/Concert/Sports): ")
221                 venue_name = input("Enter venue name: ")
222
223                 event_type = EventType[event_type_str.upper()]
224                 venue = Venue(venue_name, "Venue Address") # You can provide the actual address
225
226                 event = self.create_event(event_name, date, time_str, total_seats, ticket_price, event_type, venue)
227                 print("Event created successfully!")
228
229             elif choice == "2":
230                 event_index = int(input("Enter the index of the event to get details: "))
231                 if 0 <= event_index < len(self.events):
232                     self.selected_event = self.events[event_index]
233                     self.get_event_details()
234
235             elif choice == "3":
236                 event_index = int(input("Enter the index of the event to book tickets: "))
237                 if 0 <= event_index < len(self.events):
238                     self.selected_event = self.events[event_index]
239                     num_tickets = int(input("Enter the number of tickets to book: "))
240                     array_of_customers = []
241                     self.book_tickets(self.selected_event.event_name, num_tickets, array_of_customers)
242                 else:
243                     print("Invalid event index.")
244
245             elif choice == "4":
246                 booking_id = int(input("Enter the booking ID to cancel: "))
247                 self.cancel_booking(booking_id)
248
249             elif choice == "5":
250                 print("\nAvailable seats: {self.get_available_no_of_tickets()}")
251
252             elif choice == "6":
253                 print("Exiting the Ticket Booking System.")
254                 break
255
256
257
258 if __name__ == "__main__":

```

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

3. Create interface/abstract class **IEventServiceProvider** with following methods:
 - **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu: Venu):** Create a new event with the specified details and event type (movie, sport or concert) and return event object.
 - **getEventDetails():** return array of event details from the event class.
 - **getAvailableNoOfTickets():** return the total available tickets.

```
266 from abc import ABC, abstractmethod
267 from typing import List
268
269 class IEventServiceProvider(ABC):
270     @abstractmethod
271     def create_event(self, event_name: str, date: str, time_str: str, total_seats: int, ticket_price: float,
272                      event_type: str, venue: Venue) -> Event:
273         pass
274
275     @abstractmethod
276     def get_event_details(self) -> List[str]:
277         pass
278
279     @abstractmethod
280     def get_available_no_of_tickets(self) -> int:
281         pass
```

3. Create interface/abstract class **IBookingSystemServiceProvider** with following methods:
- **calculate_booking_cost(num_tickets)**: Calculate and set the total cost of the booking.
 - **book_tickets(eventname:str, num_tickets, arrayOfCustomer)**: Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array
also should update the attributes of Booking class.
 - **cancel_booking(booking_id)**: Cancel the booking and update the available seats.
 - **get_booking_details(booking_id)**: get the booking details.

```
285 from abc import ABC, abstractmethod
286 from typing import List
287
288 class IBookingSystemServiceProvider(ABC):
289     @abstractmethod
290     def calculate_booking_cost(self, num_tickets: int) -> float:
291         pass
292
293     @abstractmethod
294     def book_tickets(self, event_name: str, num_tickets: int, array_of_customers: List[Customer]) -> None:
295         pass
296
297     @abstractmethod
298     def cancel_booking(self, booking_id: int) -> None:
299         pass
300
301     @abstractmethod
302     def get_booking_details(self, booking_id: int) -> None:
303         pass
```

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

```

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

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

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

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

```

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

- **Attributes**

- array of events

```

336     from typing import List
337
338     class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
339         def __init__(self):
340             super().__init__()
341
342         def calculate_booking_cost(self, num_tickets: int) -> float:
343             if num_tickets > 0:
344                 return num_tickets * self.selected_event.ticket_price
345             else:
346                 print("Invalid number of tickets.")
347                 return 0
348
349         def book_tickets(self, event_name: str, num_tickets: int, array_of_customers: List[Customer]) -> None:
350             self.selected_event = None
351             for event in self.events:
352                 if event.event_name == event_name:
353                     self.selected_event = event
354                     break
355
356             if self.selected_event is not None:
357                 for _ in range(num_tickets):
358                     customer_name = input("Enter customer name: ")
359                     email = input("Enter email: ")
360                     phone_number = input("Enter phone number: ")
361                     customer = Customer(customer_name, email, phone_number)
362                     array_of_customers.append(customer)
363

```

```
363         total_cost = self.calculate_booking_cost(num_tickets)
364         booking = Booking(array_of_customers, self.selected_event, num_tickets)
365         booking.total_cost = total_cost
366
367         print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
368         booking.display_booking_details()
369     else:
370         print(f"\nEvent with name '{event_name}' not found.")
371
372 @if
373 def cancel_booking(self, booking_id: int) -> None:
374     for event in self.events:
375         for booking in event.bookings:
376             if booking.booking_id == booking_id:
377                 event.available_seats += booking.num_tickets
378                 event.bookings.remove(booking)
379                 print(f"\nBooking with ID {booking_id} canceled successfully.")
380             return
381     print(f"\nBooking with ID {booking_id} not found.")
382
383 @if
384 def get_booking_details(self, booking_id: int) -> None:
385     for event in self.events:
386         for booking in event.bookings:
387             if booking.booking_id == booking_id:
388                 print("\nBooking Details:")
389                 booking.display_booking_details()
390             return
391     print(f"\nBooking with ID {booking_id} not found.")
```

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

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

booking system by entering commands such as "create_event", "book_tickets",

"cancel_tickets", "get_available_seats", "get_event_details," and "exit."

```
396     class TicketBookingSystem:
397         def __init__(self, booking_system_provider):
398             self.booking_system_provider = booking_system_provider
399
400         def display_menu(self):
401             print("\nMenu:")
402             print("1. Create Event")
403             print("2. Get Event Details")
404             print("3. Book Tickets")
405             print("4. Cancel Booking")
406             print("5. Get Available Seats")
407             print("6. Exit")
408
409         def main(self):
410             while True:
411                 self.display_menu()
412                 choice = input("Enter your choice (1-6): ")
413
414                 if choice == "1":
415                     event_name = input("Enter event name: ")
416                     date = input("Enter event date (YYYY-MM-DD): ")
417                     time_str = input("Enter event time (HH:MM): ")
418                     total_seats = int(input("Enter total seats: "))
419                     ticket_price = float(input("Enter ticket price: "))
420                     event_type_str = input("Enter event type (Movie/Concert/Sports): ")
421                     venue_name = input("Enter venue name: ")
422
423                     event_type = EventType[event_type_str.upper()]
424                     venue = Venue(venue_name, "Venue Address") # You can provide the actual address
425
```

```
426
427             self.booking_system_provider.create_event(event_name, date, time_str, total_seats, ticket_price,
428                                         event_type, venue)
429             print("Event created successfully!")
430
431         elif choice == "2":
432             self.booking_system_provider.get_event_details()
433
434         elif choice == "3":
435             event_name = input("Enter the name of the event to book tickets: ")
436             num_tickets = int(input("Enter the number of tickets to book: "))
437             array_of_customers = []
438             self.booking_system_provider.book_tickets(event_name, num_tickets, array_of_customers)
439
440         elif choice == "4":
441             booking_id = int(input("Enter the booking ID to cancel: "))
442             self.booking_system_provider.cancel_booking(booking_id)
443
444         elif choice == "5":
445             print(f"\nAvailable Seats: {self.booking_system_provider.get_available_no_of_tickets()}")
446
447         elif choice == "6":
448             print("Exiting the Ticket Booking System.")
449             break
450
451 if __name__ == "__main__":
452     booking_system_provider = BookingSystemServiceProviderImpl()
453     ticket_booking_system = TicketBookingSystem(booking_system_provider)
454     ticket_booking_system.main()
```

Task 9: Exception Handling

throw the exception whenever needed and Handle in main method,

1. **EventNotFoundException** throw this exception when user try to book the tickets for Event not listed in the menu.
2. **InvalidBookingIDException** throw this exception when user entered the invalid bookingId when he tries to view the booking or cancel the booking.
3. **NullPointerException** handle in main method.

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

```
643     class EventNotFoundException(Exception):  
644         def __init__(self, event_name):  
645             self.event_name = event_name  
646             super().__init__(f"Event '{event_name}' not found.")  
647  
648     class InvalidBookingIDException(Exception):  
649         def __init__(self, booking_id):  
650             self.booking_id = booking_id  
651             super().__init__(f"Invalid Booking ID '{booking_id}'")  
652  
653     class NullPointerException(Exception):  
654         def __init__(self, message):  
655             super().__init__(message)  
656  
657 class TicketBookingSystem:  
658     def __init__(self, booking_system_provider):  
659         self.booking_system_provider = booking_system_provider  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670
```

```
633  
634         except Exception as g:  
635             print(f"An unexpected error occurred: {g}")  
636  
637 >     if __name__ == "__main__":  
638         try:  
639             booking_system_provider = BookingSystemServiceProviderImpl()  
640             ticket_booking_system = TicketBookingSystem(booking_system_provider)  
641             ticket_booking_system.main()  
642         except Exception as e:  
643             print(f"An unexpected error occurred: {e}")  
644  
645
```

Task 10: Collection

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

Task 9: Exception Handling

throw the exception whenever needed and Handle in main method,

1. **EventNotFoundException** throw this exception when user try to book the tickets for Event not listed in the menu.
2. **InvalidBookingIDException** throw this exception when user entered the invalid bookingId when he tries to view the booking or cancel the booking.
3. **NullPointerException** handle in main method.

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

```
553     class EventNotFoundException(Exception):
554         def __init__(self, event_name):
555             self.event_name = event_name
556             super().__init__(f"Event '{event_name}' not found.")
557
558     class InvalidBookingIDException(Exception):
559         def __init__(self, booking_id):
560             self.booking_id = booking_id
561             super().__init__(f"Invalid Booking ID '{booking_id}'.")
562
563     class NullPointerException(Exception):
564         def __init__(self, message):
565             super().__init__(message)
566
567     class TicketBookingSystem:
568         def __init__(self, booking_system_provider):
569             self.booking_system_provider = booking_system_provider
570
```

```
633
634         except Exception as g:
635             print(f"An unexpected error occurred: {g}")
636
637     if __name__ == "__main__":
638         try:
639             booking_system_provider = BookingSystemServiceProviderImpl()
640             ticket_booking_system = TicketBookingSystem(booking_system_provider)
641             ticket_booking_system.main()
642         except Exception as e:
643             print(f"An unexpected error occurred: {e}")
644
645
```

Task 10: Collection

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

```

650
651     class Booking:
652         booking_id_counter = 1
653
654     def __init__(self, customers: List[Customer], event: Event, num_tickets: int):
655         self.booking_id = Booking.booking_id_counter
656         Booking.booking_id_counter += 1
657         self.customers = customers
658         self.event = event
659         self.num_tickets = num_tickets
660         self.total_cost = 0.0
661         self.booking_date = datetime.now()
662
663     class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
664         def __init__(self):
665             super().__init__()
666             self.events = []

```

- 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.

```

725     class Booking:
726         booking_id_counter = 1
727
728     def __init__(self, customers: Set[Customer], event: Event, num_tickets: int):
729         self.booking_id = Booking.booking_id_counter
730         Booking.booking_id_counter += 1
731         self.customers = customers
732         self.event = event
733         self.num_tickets = num_tickets
734         self.total_cost = 0.0
735         self.booking_date = datetime.now()
736
737     class EventComparator:
738         def compare(self, event1: Event, event2: Event):
739             if event1.event_name != event2.event_name:
740                 return event1.event_name < event2.event_name
741             else:
742                 return event1.venue.venue_name < event2.venue.venue_name
743
744     class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
745         def __init__(self):
746             super().__init__()
747             self.events = set()
748
749         def calculate_booking_cost(self, num_tickets: int) -> float:
750             if num_tickets > 0:
751                 return num_tickets * self.selected_event.ticket_price
752             else:

```

```

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

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

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

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

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

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

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

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

```

Task 11: Database Connectivity.

1. Create **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, venue: 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.

```
307     from typing import List
308
309     from abc import ABC, abstractmethod
310
311     class IBookingSystemRepository(ABC):
312         @abstractmethod
313         def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
314                           event_type: str, venue: Venu) -> Event:
315             pass
316
317         @abstractmethod
318         def get_event_details(self) -> List[Event]:
319             pass
320
321         @abstractmethod
322         def get_available_no_of_tickets(self) -> int:
323             pass
324
325         @abstractmethod
326         def calculate_booking_cost(self, num_tickets: int) -> float:
327             pass
328
329         @abstractmethod
330         def book_tickets(self, event_name: str, num_tickets: int, list_of_customers: List[Customer]) -> None:
331             pass
332
333         @abstractmethod
334         def cancel_booking(self, booking_id: int) -> None:
335             pass
336
337         @abstractmethod
338         def get_booking_details(self, booking_id: int) -> Booking:
339             pass
```

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

```
660 def book_tickets(self, event_name: str, num_tickets: int, list_of_customers: List[Customer]) -> None:
661     self.selected_event = None
662     for event in self.events:
663         if event.event_name == event_name:
664             self.selected_event = event
665             break
666
667     if self.selected_event is not None:
668         if self.selected_event.available_seats >= num_tickets:
669             for _ in range(num_tickets):
670                 customer_name = input("Enter customer name: ")
671                 email = input("Enter email: ")
672                 phone_number = input("Enter phone number: ")
673                 customer = Customer(customer_name, email, phone_number)
674                 list_of_customers.append(customer)
675
676             total_cost = self.calculate_booking_cost(num_tickets)
677             booking = Booking(set(list_of_customers), self.selected_event, num_tickets)
678             booking.total_cost = total_cost
679
680             self.selected_event.available_seats -= num_tickets
681             self.bookings.add(booking)
682
683             print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
684             booking.display_booking_details()
685         else:
686             raise ValueError("Not enough available seats for booking.")
687     else:
688         raise EventNotFoundException(event_name)
689
690
691 from typing import List, Set
692
693 class BookingSystemRepositoryImpl(IBookingSystemRepository):
694     def __init__(self):
695         # Simulating a database with in-memory data structures
696         self.events: List[Event] = []
697         self.bookings: Set[Booking] = set()
698
699     def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
700                     event_type: str, venue: Venu) -> Event:
701         new_event = Event(event_name, date, time, venue, total_seats, total_seats, ticket_price, EventType[event_type.upper()])
702         self.events.append(new_event)
703         return new_event
704
705     def get_event_details(self) -> List[Event]:
706         return self.events
707
708     def get_available_no_of_tickets(self) -> int:
709         total_available_tickets = sum(event.available_seats for event in self.events)
710         return total_available_tickets
711
712     def calculate_booking_cost(self, num_tickets: int) -> float:
713         if num_tickets > 0 and self.selected_event:
714             return num_tickets * self.selected_event.ticket_price
715         else:
716             raise ValueError("Invalid number of tickets or event not selected.")
```

7. Place the interface/abstract class in service package and interface implementation class, concrete class in bean package and **TicketBookingSystemRepository** class in app package.
8. Should throw appropriate exception as mentioned in above task along with handle **SQLException**.

```
719 from abc import ABC, abstractmethod
720
721 @l1 class IBookingSystemRepository(ABC):
722     @abstractmethod
723     def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
724                      event_type: str, venue: Venu) -> Event:
725         pass
726
727     @abstractmethod
728     def get_event_details(self) -> List[Event]:
729         pass
730
731 class BookingSystemRepositoryImpl(IBookingSystemRepository):
732     class TicketBookingSystem:
733         def __init__(self, booking_system_repository: IBookingSystemRepository):
734             self.booking_system_repository = booking_system_repository
735
736         def handle_sql_exception(self):
737             try:
738                 pass
739             except SQLException as e:
740                 print(f"SQLException: {e}")
741                 raise e
```

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

- Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats", "get_event_details," and "exit."

< Previous

Next >

```
947     class TicketBookingSystem:
948         def __init__(self, booking_system_repository: IBookingSystemRepository):
949             self.booking_system_repository = booking_system_repository
950
951         def display_menu(self):
952             print("\n===== Ticket Booking System Menu =====")
953             print("1. Create Event")
954             print("2. Book Tickets")
955             print("3. Cancel Tickets")
956             print("4. Get Available Seats")
957             print("5. Get Event Details")
958             print("6. Exit")
959
960         def create_event(self):
961             event_name = input("Enter event name: ")
962             date = input("Enter event date: ")
963             time = input("Enter event time: ")
964             total_seats = int(input("Enter total seats: "))
965             ticket_price = float(input("Enter ticket price: "))
966             event_type = input("Enter event type (Movie/Sports/Concert): ")
967             venue_name = input("Enter venue name: ")
968             venue_address = input("Enter venue address: ")
969             VENUE = Venu(venue_name, venue_address)
970
971             self.booking_system_repository.create_event(event_name, date, time, total_seats, ticket_price, event_type, VENUE)
972             print("Event created successfully!")
973
```

```
905     def get_available_seats(self):
906         try:
907             # Get available seats using the repository
908             available_seats = self.booking_system_repository.get_available_no_of_tickets()
909             print(f"Available seats: {available_seats}")
910         except SQLException as e:
911             print(f"Error: {e}")
912
913     def get_event_details(self):
914         # Get event details using the repository
915         event_details = self.booking_system_repository.get_event_details()
916         print("\n===== Event Details =====")
917         for event in event_details:
918             print(event.display_event_details())
919
920     def run_ticket_booking_system(self):
921         while True:
922             self.display_menu()
923             choice = input("Enter your choice: ")
924
925             if choice == "1":
926                 self.create_event()
927             elif choice == "2":
928                 self.book_tickets()
929             elif choice == "3":
930                 self.cancel_tickets()
931             elif choice == "4":
932                 self.get_available_seats()
933             elif choice == "5":
934                 self.get_event_details()
935             elif choice == "6":
936                 print("Exiting Ticket Booking System. Goodbye!")
937                 break
938             else:
939                 print("Invalid choice. Please enter a valid option.")
940
```