

Assignment-5

Control structure

Task 1: Conditional Statements

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

Tasks:

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

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

```
Enter the number of available tickets: 5  
Enter the number of tickets to book: 4  
Tickets booked successfully! Remaining tickets: 1  
Process finished with exit code 0
```

Task 2: Nested Conditional Statements

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

```

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
73     class EventType(Enum):
74         MOVIE = 'Movie'
75         SPORTS = 'Sports'
76         CONCERT = 'Concert'
77
78     class Event:
79         def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type):
80             self.event_name = event_name
81             self.event_date = event_date
82             self.event_time = event_time
83             self.venue_name = venue_name
84             self.total_seats = total_seats
85             self.available_seats = total_seats
86             self.ticket_price = ticket_price
87             self.event_type = event_type
88             self.booked_tickets = 0
89
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_ Parameter self of main.Event.get_event_time
105            self: Event
106        def set_event_time(self, event_time):
107            self.event_time = event_time
108
109        def get_venue_name(self):
110            return self.venue_name
111
112        def set_venue_name(self, venue_name):
113            self.venue_name = venue_name
114
115        def get_total_seats(self):
116            return self.total_seats
117
118        def set_total_seats(self, total_seats):
119            self.total_seats = total_seats
120
121        def get_available_seats(self):
122            return self.available_seats
123
124        def set_available_seats(self, available_seats):
125            self.available_seats = available_seats
126
127        def get_ticket_price(self):
128            return self.ticket_price
```

```

130     def set_ticket_price(self, ticket_price):
131         self.ticket_price = ticket_price
132
133     def get_event_type(self):
134         return self.event_type
135
136     def set_event_type(self, event_type):
137         self.event_type = event_type
138
139     def display_event_details(self):
140         print("Event Details:")
141         print(f"Event Name: {self.event_name}")
142         print(f"Event Date: {self.event_date}")
143         print(f"Event Time: {self.event_time}")
144         print(f"Venue Name: {self.venue_name}")
145         print(f"Total Seats: {self.total_seats}")
146         print(f"Available Seats: {self.available_seats}")
147         print(f"Ticket Price: ${self.ticket_price:.2f}")
148         print(f"Event Type: {self.event_type}")
149
150     def calculate_total_revenue(self):
151         return self.ticket_price * self.booked_tickets
152
153     def get_booked_no_of_tickets(self):
154         return self.booked_tickets
155
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

```

```

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,

- o address
- **Methods and Constructors:**
- o **display_venue_details():** Display venue details.
- o 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:**
 - 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.

```

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

- **Methods:**

1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
2. **display_event_details():** Display movie details, including genre.

```
320     class EventType(Enum):
321         MOVIE = 'Movie'
322         SPORTS = 'Sports'
323         CONCERT = 'Concert'
324
325     class Movie(Event):
326         def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
327                      genre, actor_name, actress_name):
328             super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
329             self.genre = genre
330             self.actor_name = actor_name
331             self.actress_name = actress_name
332
333         def get_genre(self):
334             return self.genre
335
336         def set_genre(self, genre):
337             self.genre = genre
338
339         def get_actor_name(self):
340             return self.actor_name
341
342         def set_actor_name(self, actor_name):
343             self.actor_name = actor_name
344
345         def get_actress_name(self):
346             return self.actress_name
347
348         def set_actress_name(self, actress_name):
349             self.actress_name = actress_name
350
```

```
341
342         def set_actor_name(self, actor_name):
343             self.actor_name = actor_name
344
345         def get_actress_name(self):
346             return self.actress_name
347
348         def set_actress_name(self, actress_name):
349             self.actress_name = actress_name
350
351         def display_event_details(self):
352             super().display_event_details()
353             print(f"Genre: {self.genre}")
354             print(f"Actor: {self.actor_name}")
355             print(f"Actress: {self.actress_name}")
356
357     if __name__ == "__main__":
358         movie_date = date(2023, 1, 15)
359         movie_time = time(20, 0)
360         movie_type = EventType.MOVIE
361         movie = Movie("Blockbuster Movie", movie_date, movie_time, "Cinema City", 500, 10.0, movie_type,
362                       "Action", "John Doe", "Jane Doe")
363
364         movie.display_event_details()
365
```

```

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.

```

368 class Concert(Event):
369     def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
370                  artist, concert_type):
371         super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
372         self.artist = artist
373         self.concert_type = concert_type
374
375     # Getter and Setter methods
376     def get_artist(self):
377         return self.artist
378
379     def set_artist(self, artist):
380         self.artist = artist
381
382     def get_concert_type(self):
383         return self.concert_type
384
385     def set_concert_type(self, concert_type):
386         self.concert_type = concert_type
387
388     def display_event_details(self):
389         super().display_event_details()
390         print(f"Artist: {self.artist}")
391         print(f"Concert Type: {self.concert_type}")
392
393 if __name__ == "__main__":
394     concert_date = date(2023, 2, 20)
395     concert_time = time(19, 30)
396     concert_type = EventType.CONCERT
397     concert = Concert("Live Concert", concert_date, concert_time, "Music Arena", 1000, 50.0, concert_type,
398                       "Rock Band", "Rock")
399     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.

```
402 class Sports(Event):  
403     def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,  
404                  sport_name, teams_name):  
405         super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)  
406         self.sport_name = sport_name  
407         self.teams_name = teams_name  
408  
409     # Getter and Setter methods  
410     def get_sport_name(self):  
411         return self.sport_name  
412  
413     def set_sport_name(self, sport_name):  
414         self.sport_name = sport_name  
415  
416     def get_teams_name(self):  
417         return self.teams_name  
418  
419     def set_teams_name(self, teams_name):  
420         self.teams_name = teams_name  
421
```

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

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

```
638 class Concert(Event):
639     def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
640                  artist, concert_type):
641         super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
642         self.artist = artist
643         self.concert_type = concert_type
644
645     def display_event_details(self):
646         super().display_event_details()
647         print(f"Artist: {self.artist}")
648         print(f"Concert Type: {self.concert_type}")
649
650     def book_tickets(self, num_tickets):
651         if 0 < num_tickets <= self.available_seats:
652             self.available_seats -= num_tickets
653             return self.ticket_price * num_tickets
654         else:
655             print("Not enough available seats.")
656             return 0.0
657
658     def cancel_tickets(self, num_tickets):
659         if 0 < num_tickets <= (self.total_seats - self.available_seats):
660             self.available_seats += num_tickets
661             return self.ticket_price * num_tickets
662         else:
663             print("Invalid number of tickets to cancel.")
664             return 0.0
665
666 class Sports(Event):
667     def __init__(self, event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
668                  sport_name, teams_name):
669         super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
670         self.sport_name = sport_name
671         self.teams_name = teams_name
672
673     def display_event_details(self):
674         super().display_event_details()
675         print(f"Sport Name: {self.sport_name}")
676         print(f"Teams: {self.teams_name}")
677
678     def book_tickets(self, num_tickets):
679         if 0 < num_tickets <= self.available_seats:
680             self.available_seats -= num_tickets
681             return self.ticket_price * num_tickets
682         else:
683             print("Not enough available seats.")
684             return 0.0
685
686     def cancel_tickets(self, num_tickets):
687         if 0 < num_tickets <= (self.total_seats - self.available_seats):
688             self.available_seats += num_tickets
689             return self.ticket_price * num_tickets
690         else:
691             print("Invalid number of tickets to cancel.")
692             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()
```

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

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

```
866
867     elif choice == "6":
868         print("Exiting the Ticket Booking System.")
869         break
870
```

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: abc event
Enter event date (YYYY-MM-DD): 2024-01-01
Enter event time (HH:MM): 09:00
Enter total seats: 50
Enter ticket price: 20
Enter event type (Movie/Concert/Sports): Sports
Enter venue name: abc

```

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     from datetime import date, time
906
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.

```

50 class Movie(Event):
51     def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, genre, actor_name, actress_name):
52         super().__init__(event_name, event_date, event_time, venue, total_seats, ticket_price, EventType.MOVIE)
53         self.genre = genre
54         self.actor_name = actor_name
55         self.actress_name = actress_name
56
57
58
59 class Concert(Event):
60     def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, artist, concert_type):
61         super().__init__(event_name, event_date, event_time, venue, total_seats, ticket_price, EventType.CONCERT)
62         self.artist = artist
63         self.concert_type = concert_type
64
65 class Sport(Event):
66     def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, sport_name, teams_name):
67         super().__init__(event_name, event_date, event_time, venue, total_seats, ticket_price, EventType.SPORTS)
68         self.sport_name = sport_name
69         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.

```

80     class Customer:
81         def __init__(self, customer_name, email, phone_number):
82             self.customer_name = customer_name
83             self.email = email
84             self.phone_number = phone_number
85
86         def display_customer_details(self):
87             print("Customer Details:")
88             print(f"Customer Name: {self.customer_name}")
89             print(f"Email: {self.email}")
90             print(f"Phone Number: {self.phone_number}")
91
92         # Getter and Setter methods
93         def get_customer_name(self):
94             return self.customer_name
95
96         def set_customer_name(self, customer_name):
97             self.customer_name = customer_name
98
99         def get_email(self):
100            return self.email
101
102         def set_email(self, email):
103             self.email = email
104
105         def get_phone_number(self):
106             return self.phone_number
107
108         def set_phone_number(self, phone_number):
109             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             return self.total_cost

```

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

- **Attributes**

- array of events

- **Methods and 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)

```

```

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                     print(f"\nBooking with ID {booking_id} not found.")
193
194         def get_available_no_of_tickets(self):
195             return self.selected_event.available_seats
196
197         def get_event_details(self):
198             self.selected_event.display_event_details()
199
200         def display_menu(self):
201             print("\nMenu:")
202             print("1. Create Event")
203             print("2. Get Event Details")
204             print("3. Book Tickets")
205             print("4. Cancel Booking")
206             print("5. Get Available Seats")

```

```
206
207
208
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
236
237             elif choice == "3":
238                 event_index = int(input("Enter the index of the event to book tickets: "))
239                 if 0 <= event_index < len(self.events):
240                     self.selected_event = self.events[event_index]
241                     num_tickets = int(input("Enter the number of tickets to book: "))
242                     array_of_customers = []
243                     self.book_tickets(self.selected_event.event_name, num_tickets, array_of_customers)
244                 else:
245                     print("Invalid event index.")
246
247             elif choice == "4":
248                 booking_id = int(input("Enter the booking ID to cancel: "))
249                 self.cancel_booking(booking_id)
250
251             elif choice == "5":
252                 print(f"\nAvailable Seats: {self.get_available_no_of_tickets()}")
253
254             elif choice == "6":
255                 print("Exiting the Ticket Booking System.")
256                 break
257
258 if __name__ == "__main__":

```

```
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                 else:
235                     print("Invalid event index.")
236
237             elif choice == "3":
238                 event_index = int(input("Enter the index of the event to book tickets: "))
239                 if 0 <= event_index < len(self.events):
240                     self.selected_event = self.events[event_index]
241                     num_tickets = int(input("Enter the number of tickets to book: "))
242                     array_of_customers = []
243                     self.book_tickets(self.selected_event.event_name, num_tickets, array_of_customers)
244                 else:
245                     print("Invalid event index.")
246
247             elif choice == "4":
248                 booking_id = int(input("Enter the booking ID to cancel: "))
249                 self.cancel_booking(booking_id)
250
251             elif choice == "5":
252                 print(f"\nAvailable Seats: {self.get_available_no_of_tickets()}")
253
254             elif choice == "6":
255                 print("Exiting the Ticket Booking System.")
256                 break
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.

```

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

```

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

```

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

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

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

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

```

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

- **Attributes**

- array of events

```

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 0.0
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
364     total_cost = self.calculate_booking_cost(num_tickets)
365     booking = Booking(array_of_customers, self.selected_event, num_tickets)
366     booking.total_cost = total_cost
367
368     print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
369     booking.display_booking_details()
370 else:
371     print(f"\nEvent with name '{event_name}' not found.")
372
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 ⚡ def get_booking_details(self, booking_id: int) -> None:
384     for event in self.events:
385         for booking in event.bookings:
386             if booking.booking_id == booking_id:
387                 print("\nBooking Details:")
388                 booking.display_booking_details()
389             return
390     print(f"\nBooking with ID {booking_id} not found.")
391

```

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
```

```
425
426             self.booking_system_provider.create_event(event_name, date, time_str, total_seats, ticket_price,
427                                         event_type, venue)
428             print("Event created successfully!")
429
430         elif choice == "2":
431             self.booking_system_provider.get_event_details()
432
433         elif choice == "3":
434             event_name = input("Enter the name of the event to book tickets: ")
435             num_tickets = int(input("Enter the number of tickets to book: "))
436             array_of_customers = []
437             self.booking_system_provider.book_tickets(event_name, num_tickets, array_of_customers)
438
439         elif choice == "4":
440             booking_id = int(input("Enter the booking ID to cancel: "))
441             self.booking_system_provider.cancel_booking(booking_id)
442
443         elif choice == "5":
444             print(f"\nAvailable Seats: {self.booking_system_provider.get_available_no_of_tickets()}")
445
446         elif choice == "6":
447             print("Exiting the Ticket Booking System.")
448             break
449
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.

```
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 e:
635         print(f"An unexpected error occurred: {e}")
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:

```

```

755     def book_tickets(self, event_name: str, num_tickets: int, set_of_customers: Set[Customer]) -> None:
756         self.selected_event = None
757         for event in self.events:
758             if event.event_name == event_name:
759                 self.selected_event = event
760                 break
761
762             if self.selected_event is not None:
763                 for _ in range(num_tickets):
764                     customer_name = input("Enter customer name: ")
765                     email = input("Enter email: ")
766                     phone_number = input("Enter phone number: ")
767                     customer = Customer(customer_name, email, phone_number)
768                     set_of_customers.add(customer)
769
770                     total_cost = self.calculate_booking_cost(num_tickets)
771                     booking = Booking(set_of_customers, self.selected_event, num_tickets)
772                     booking.total_cost = total_cost
773
774                     self.selected_event.available_seats -= num_tickets
775                     self.selected_event.bookings.add(booking)
776
777                     print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
778                     booking.display_booking_details()
779             else:
780                 raise EventNotFoundException(event_name)

```

```

782     def cancel_booking(self, booking_id: int) -> None:
783         for event in self.events:
784             for booking in event.bookings:
785                 if booking.booking_id == booking_id:
786                     event.available_seats += booking.num_tickets
787                     event.bookings.remove(booking)
788                     print(f"\nBooking with ID {booking_id} canceled successfully.")
789                     return
790             raise InvalidBookingIDException(booking_id)
791
792     def get_booking_details(self, booking_id: int) -> None:
793         for event in self.events:
794             for booking in event.bookings:
795                 if booking.booking_id == booking_id:
796                     print("\nBooking Details:")
797                     booking.display_booking_details()
798                     return
799             raise InvalidBookingIDException(booking_id)
800
801     def sort_events(self) -> List[Event]:
802         comparator = EventComparator()
803         return sorted(self.events, key=functools.cmp_to_key(comparator.compare))
804

```

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, 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.

```

807     from typing import List
808     from abc import ABC, abstractmethod
809
810     class IBookingSystemRepository(ABC):
811         @abstractmethod
812         def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
813                         event_type: str, venue: Venu) -> Event:
814             pass
815
816         @abstractmethod
817         def get_event_details(self) -> List[Event]:
818             pass
819
820         @abstractmethod
821         def get_available_no_of_tickets(self) -> int:
822             pass
823
824         @abstractmethod
825         def calculate_booking_cost(self, num_tickets: int) -> float:
826             pass
827
828         @abstractmethod
829         def book_tickets(self, event_name: str, num_tickets: int, list_of_customers: List[Customer]) -> None:
830             pass
831
832         @abstractmethod
833         def cancel_booking(self, booking_id: int) -> None:
834             pass

```

```

835
836         @abstractmethod
837         def get_booking_details(self, booking_id: int) -> Booking:
838             pass
839

```

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

```

868     def book_tickets(self, event_name: str, num_tickets: int, list_of_customers: List[Customer]) -> None:
869         self.selected_event = None
870         for event in self.events:
871             if event.event_name == event_name:
872                 self.selected_event = event
873                 break
874
875         if self.selected_event is not None:
876             if self.selected_event.available_seats >= num_tickets:
877                 for _ in range(num_tickets):
878                     customer_name = input("Enter customer name: ")
879                     email = input("Enter email: ")
880                     phone_number = input("Enter phone number: ")
881                     customer = Customer(customer_name, email, phone_number)
882                     list_of_customers.append(customer)
883
884             total_cost = self.calculate_booking_cost(num_tickets)
885             booking = Booking(set(list_of_customers), self.selected_event, num_tickets)
886             booking.total_cost = total_cost
887
888             self.selected_event.available_seats -= num_tickets
889             self.bookings.add(booking)
890
891             print(f"\nBooking successful! Total cost: ${total_cost:.2f}")
892             booking.display_booking_details()
893         else:
894             raise ValueError("Not enough available seats for booking.")
895         else:
896             raise EventNotFoundException(event_name)
897

```

```

841     from typing import List, Set
842
843     class BookingSystemRepositoryImpl(IBookingSystemRepository):
844         def __init__(self):
845             # Simulating a database with in-memory data structures
846             self.events: List[Event] = []
847             self.bookings: Set[Booking] = set()
848
849         def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
850                         event_type: str, venue: Venu) -> Event:
851             new_event = Event(event_name, date, time, venue, total_seats, total_seats, ticket_price, EventType[event_type.upper()])
852             self.events.append(new_event)
853             return new_event
854
855         def get_event_details(self) -> List[Event]:
856             return self.events
857
858         def get_available_no_of_tickets(self) -> int:
859             total_available_tickets = sum(event.available_seats for event in self.events)
860             return total_available_tickets
861
862         def calculate_booking_cost(self, num_tickets: int) -> float:
863             if num_tickets > 0 and self.selected_event:
864                 return num_tickets * self.selected_event.ticket_price
865             else:
866                 raise ValueError("Invalid number of tickets or event not selected.")
867

```

```

898     def cancel_booking(self, booking_id: int) -> None:
899         booking_to_cancel = next((booking for booking in self.bookings if booking.booking_id == booking_id), None)
900         if booking_to_cancel:
901             self.selected_event.available_seats += booking_to_cancel.num_tickets
902             self.bookings.remove(booking_to_cancel)
903             print(f"\nBooking with ID {booking_id} canceled successfully.")
904         else:
905             raise InvalidBookingIDException(booking_id)
906
907     def get_booking_details(self, booking_id: int) -> Booking:
908         booking_details = next((booking for booking in self.bookings if booking.booking_id == booking_id), None)
909         if booking_details:
910             return booking_details
911         else:
912             raise InvalidBookingIDException(booking_id)

```

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**.

```

919     from abc import ABC, abstractmethod
920
921     class IBookingSystemRepository(ABC):
922         @abstractmethod
923         def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
924                           event_type: str, venue: Venu) -> Event:
925             pass
926
927         @abstractmethod
928         def get_event_details(self) -> List[Event]:
929             pass
930
931     class BookingSystemRepositoryImpl(IBookingSystemRepository):
932         class TicketBookingSystem:
933             def __init__(self, booking_system_repository: IBookingSystemRepository):
934                 self.booking_system_repository = booking_system_repository
935
936             def handle_sql_exception(self):
937                 try:
938                     pass
939                 except SQLException as e:
940                     print(f"SQLException: {e}")
941                     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."

```
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
```

```
1005     def get_available_seats(self):
1006         try:
1007             # Get available seats using the repository
1008             available_seats = self.booking_system_repository.get_available_no_of_tickets()
1009             print(f"Available seats: {available_seats}")
1010         except SQLException as e:
1011             print(f"Error: {e}")
1012
1013     def get_event_details(self):
1014         # Get event details using the repository
1015         event_details = self.booking_system_repository.get_event_details()
1016         print("\n===== Event Details =====")
1017         for event in event_details:
1018             print(event.display_event_details())
1019
1020     def run_ticket_booking_system(self):
1021         while True:
1022             self.display_menu()
1023             choice = input("Enter your choice: ")
1024
1025             if choice == "1":
1026                 self.create_event()
1027             elif choice == "2":
1028                 self.book_tickets()
1029             elif choice == "3":
1030                 self.cancel_tickets()
1031             elif choice == "4":
1032                 self.get_available_seats()
1033             elif choice == "5":
1034                 self.get_event_details()
1035             elif choice == "6":
1036                 print("Exiting Ticket Booking System. Goodbye!")
1037                 break
1038             else:
1039                 print("Invalid choice. Please enter a valid option.")
1040
```