Sobhika Kamalavasan

Ticket Booking System

Control structure

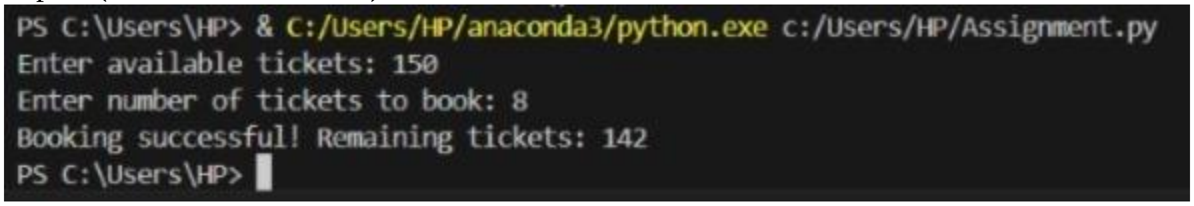
Task 1: Conditional Statements

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

Tasks:

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

```
available_ticket = int(input("Enter available tickets: "))
booking_ticket = int(input("Enter number of tickets to book: "))

if booking_ticket <= available_ticket:
    print(f"Booking successful! Remaining tickets: {available_ticket - booking_ticket}")
else:
    print("Ticket unavailable")
```



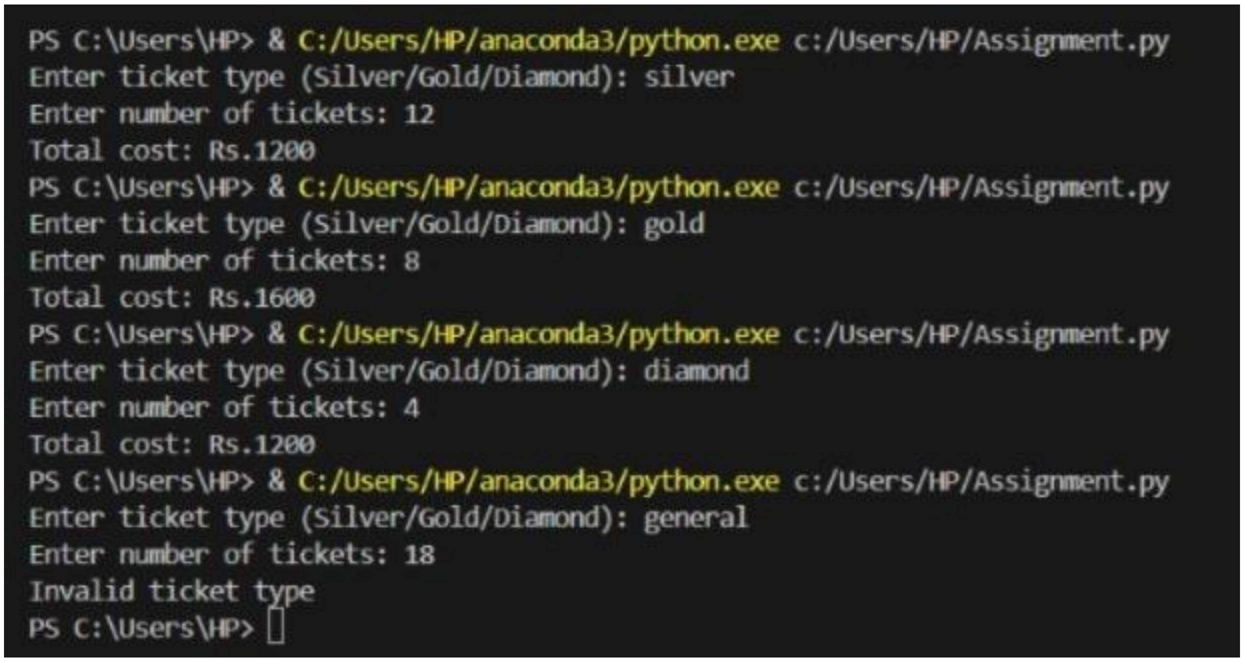
```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
Enter available tickets: 150
Enter number of tickets to book: 8
Booking successful! Remaining tickets: 142
PS C:\Users\HP>
```

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.

```
ticket_type = input("Enter ticket type (Silver/Gold/Diamond): ").lower()

no_of_tickets = int(input("Enter number of tickets: "))


if ticket_type == "silver":

    price = 100

elif ticket_type == "gold":

    price = 200
```

```
elif ticket_type == "diamond":
    price = 300
else:
    print("Invalid ticket type")
    price = 0

if price:
    total_cost = price * no_of_tickets
    print(f"Total cost: Rs.{total_cost}")
```



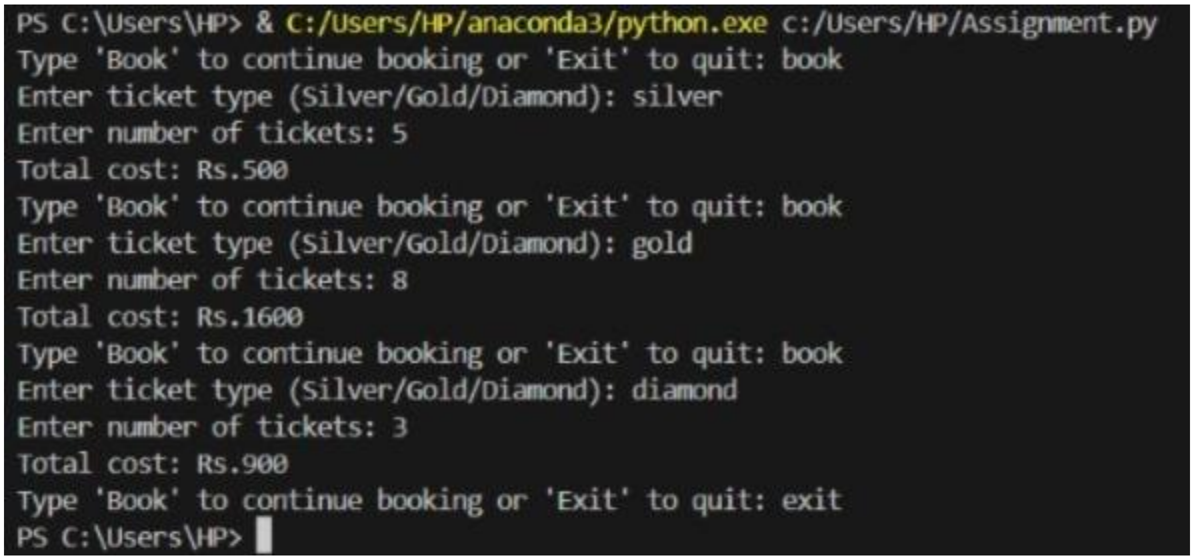
```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
Enter ticket type (Silver/Gold/Diamond): silver
Enter number of tickets: 12
Total cost: Rs.1200
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
Enter ticket type (Silver/Gold/Diamond): gold
Enter number of tickets: 8
Total cost: Rs.1600
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
Enter ticket type (Silver/Gold/Diamond): diamond
Enter number of tickets: 4
Total cost: Rs.1200
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
Enter ticket type (Silver/Gold/Diamond): general
Enter number of tickets: 18
Invalid ticket type
PS C:\Users\HP> 
```

Task 3: Looping

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

```
while True:
    command = input("Type 'Book' to continue booking or 'Exit' to quit: ").lower()
    if command == "exit":
        break
    ticket_type = input("Enter ticket type (Silver/Gold/Diamond): ").lower()
```

```
no_of_tickets = int(input("Enter number of tickets: "))
if ticket_type == "silver":
    price = 100
elif ticket_type == "gold":
    price = 200
elif ticket_type == "diamond":
    price = 300
else:
    print("Invalid ticket type")
    continue
total_cost = price * no_of_tickets
print(f"Total cost: Rs.{total_cost}")
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
Type 'Book' to continue booking or 'Exit' to quit: book
Enter ticket type (Silver/Gold/Diamond): silver
Enter number of tickets: 5
Total cost: Rs.500
Type 'Book' to continue booking or 'Exit' to quit: book
Enter ticket type (Silver/Gold/Diamond): gold
Enter number of tickets: 8
Total cost: Rs.1600
Type 'Book' to continue booking or 'Exit' to quit: book
Enter ticket type (Silver/Gold/Diamond): diamond
Enter number of tickets: 3
Total cost: Rs.900
Type 'Book' to continue booking or 'Exit' to quit: exit
PS C:\Users\HP>
```

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,

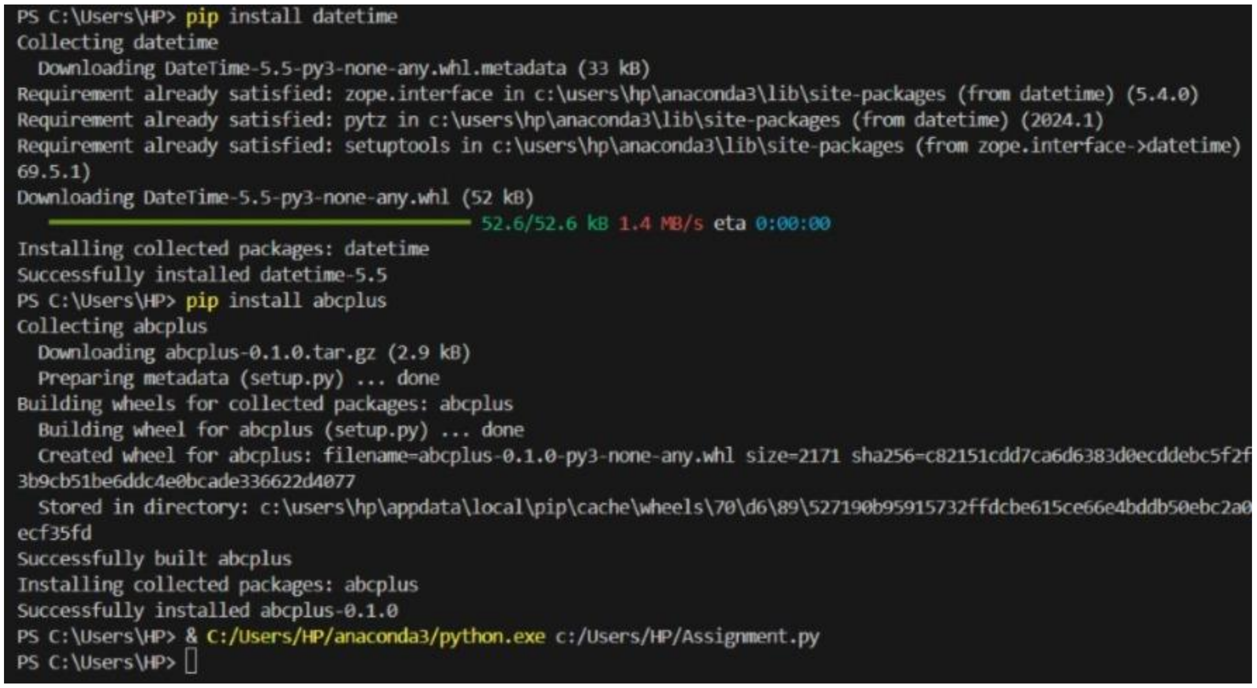
    - total_seats,
    - available_seats,
    - ticket_price DECIMAL,
    - event_type ENUM('Movie', 'Sports', 'Concert')
  - Methods and Constuctors:
    - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
    - calculate_total_revenue(): Calculate and return the total revenue based on the number of tickets sold.
    - getBookedNoOfTickets(): return the total booked tickets
    - book_tickets(num_tickets): Book a specified number of tickets for an event. Initially available seats are equal to the total seats when tickets are booked available seats number should be reduced.
    - cancel_booking(num_tickets): Cancel the booking and update the available seats.
    - display_event_details(): Display event details, including event name, date time seat availability.

2. Venue Class:
   - Attributes:
     - venue_name,
     - address
   - Methods and Constuctors:
     - display_venue_details(): Display venue details.
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
3. Customer Class:
   - Attributes:
     - customer_name,
     - email,
     - phone_number,
   - Methods and Constuctors:
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
     - display_customer_details(): Display customer details.
4. Booking Class to represent the Tiket booking system. Perform the following operation in main method. Note:- Use Event class object for the following operation.
   - Methods and Constuctors:
     - 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

from abc import ABC, abstractmethod

from datetime import datetime



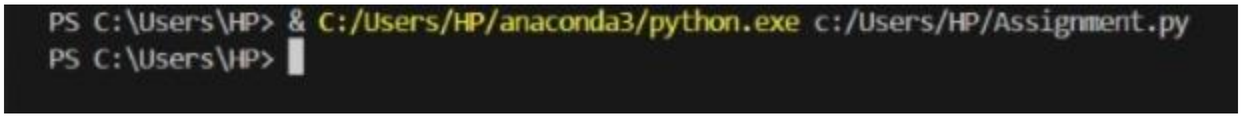
```
PS C:\Users\HP> pip install datetime
Collecting datetime
  Downloading DateTime-5.5-py3-none-any.whl.metadata (33 kB)
Requirement already satisfied: zope.interface in c:\users\hp\anaconda3\lib\site-packages (from datetime) (5.4.0)
Requirement already satisfied: pytz in c:\users\hp\anaconda3\lib\site-packages (from datetime) (2024.1)
Requirement already satisfied: setuptools in c:\users\hp\anaconda3\lib\site-packages (from zope.interface->datetime)
69.5.1)
Downloading DateTime-5.5-py3-none-any.whl (52 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.6/52.6 kB 1.4 MB/s eta 0:00:00
Installing collected packages: datetime
Successfully installed datetime-5.5
PS C:\Users\HP> pip install abcplus
Collecting abcplus
  Downloading abcplus-0.1.0.tar.gz (2.9 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: abcplus
  Building wheel for abcplus (setup.py) ... done
  Created wheel for abcplus: filename=abcplus-0.1.0-py3-none-any.whl size=2171 sha256=c82151cdd7ca6d6383d0ecddebc5f2f
3b9cb51be6ddc4e0bcade336622d4077
  Stored in directory: c:\users\hp\appdata\local\pip\cache\wheels\70\d6\89\527190b95915732ffdcbe615ce66e4bddb50ebc2a0
ecf35fd
Successfully built abcplus
Installing collected packages: abcplus
Successfully installed abcplus-0.1.0
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

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.

class Event:

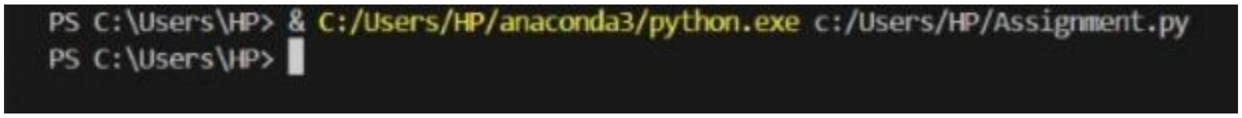
```
    def __init__(self, event_name='', date='', time='', total_seats=0, ticket_price=0.0,
venue_name=''):
        self.event_name = event_name
        self.date = date
        self.time = time
        self.total_seats = total_seats
        self.ticket_price = ticket_price
        self.venue_name = venue_name

    def get_event_name(self): return self.event_name
    def get_date(self): return self.date
    def get_time(self): return self.time
    def get_total_seats(self): return self.total_seats
    def get_ticket_price(self): return self.ticket_price
    def get_venue_name(self): return self.venue_name

    def set_event_name(self, event_name): self.event_name = event_name
    def set_date(self, date): self.date = date
    def set_time(self, time): self.time = time
    def set_total_seats(self, total_seats): self.total_seats = total_seats
    def set_ticket_price(self, ticket_price): self.ticket_price = ticket_price
    def set_venue_name(self, venue_name): self.venue_name = venue_name

    def display_event_details(self):
        print(f"Event: {self.event_name}\nDate: {self.date}\nTime: {self.time}\nVenue:
{self.venue_name}\nSeats: {self.total_seats}\nPrice: {self.ticket_price}")

class Movie(Event):
    def __init__(self, event_name='', date='', time='', total_seats=0, ticket_price=0.0,
venue_name='', genre='', actor_name='', actress_name=''):
        super().__init__(event_name, date, time, total_seats, ticket_price, venue_name)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def get_genre(self): return self.genre
    def get_actor_name(self): return self.actor_name
    def get_actress_name(self): return self.actress_name

    def set_genre(self, genre): self.genre = genre
    def set_actor_name(self, actor_name): self.actor_name = actor_name
    def set_actress_name(self, actress_name): self.actress_name = actress_name

    def display_event_details(self):
        super().display_event_details()
        print(f"Genre: {self.genre}\nActor: {self.actor_name}\nActress:
{self.actress_name}")
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

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

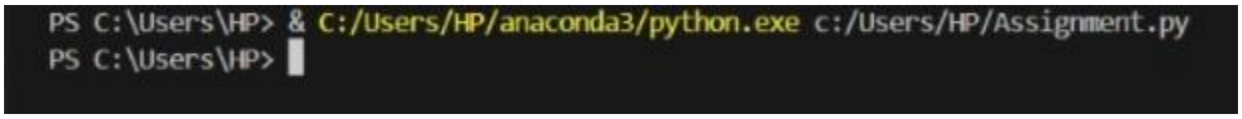
```
class Concert(Event):

    def __init__(self, event_name='', date='', time='', total_seats=0, ticket_price=0.0,
venue_name='', artist='', concert_type=''):

        super().__init__(event_name, date, time, total_seats, ticket_price, venue_name)

        self.artist = artist

        self.concert_type = concert_type


    def get_artist(self): return self.artist

    def get_concert_type(self): return self.concert_type


    def set_artist(self, artist): self.artist = artist

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


    def display_event_details(self):

        super().display_event_details()

        print(f"Artist: {self.artist}\nConcert Type: {self.concert_type}")
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

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

```
class Sports(Event):
    def __init__(self, event_name='', date='', time='', total_seats=0, ticket_price=0.0,
venue_name='', sport_name='', teams_name=''):
        super().__init__(event_name, date, time, total_seats, ticket_price, venue_name)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def get_sport_name(self): return self.sport_name
    def get_teams_name(self): return self.teams_name

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

    def display_event_details(self):
        super().display_event_details()
        print(f"Sport: {self.sport_name}\nTeams: {self.teams_name}")
```

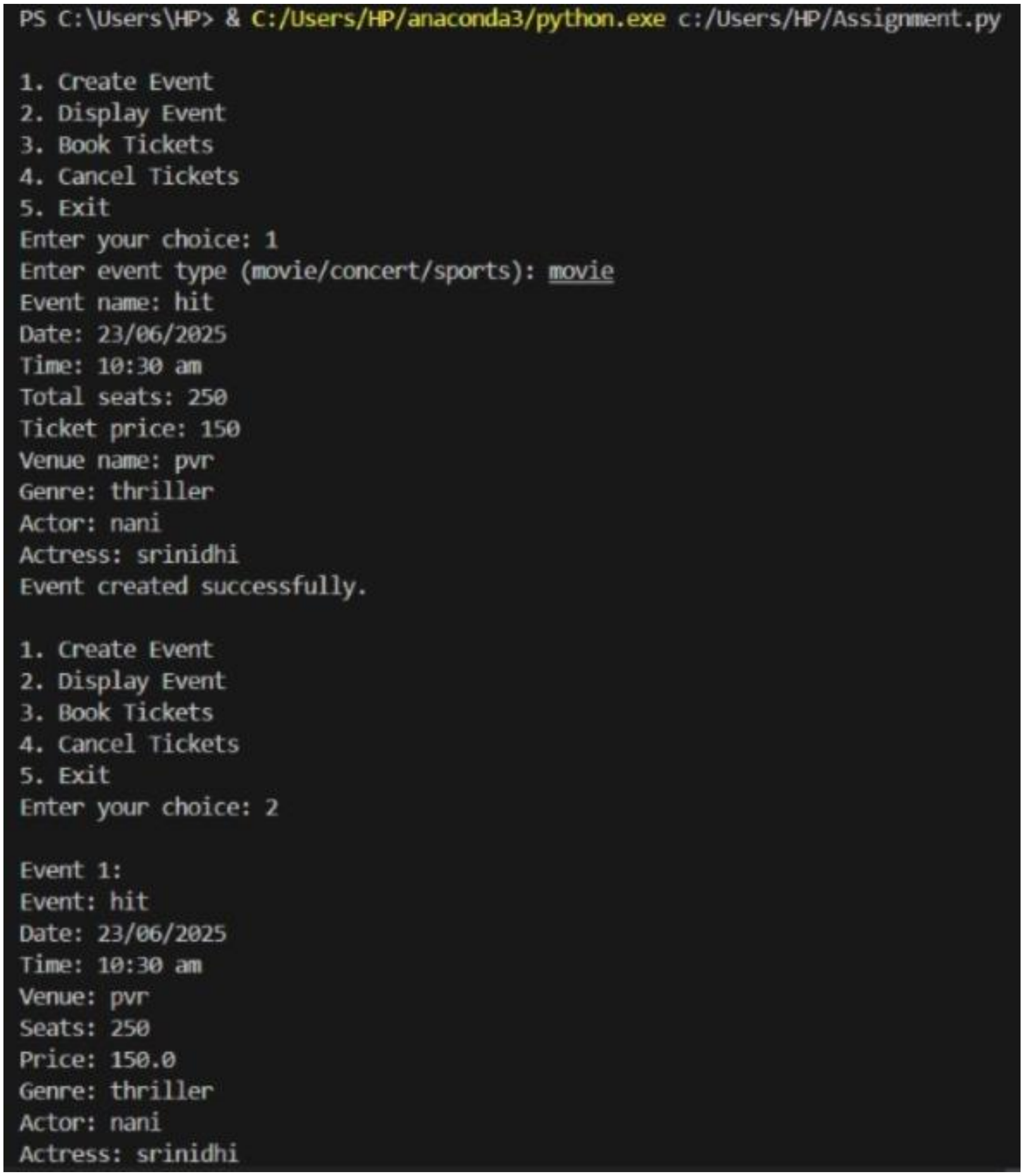


```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

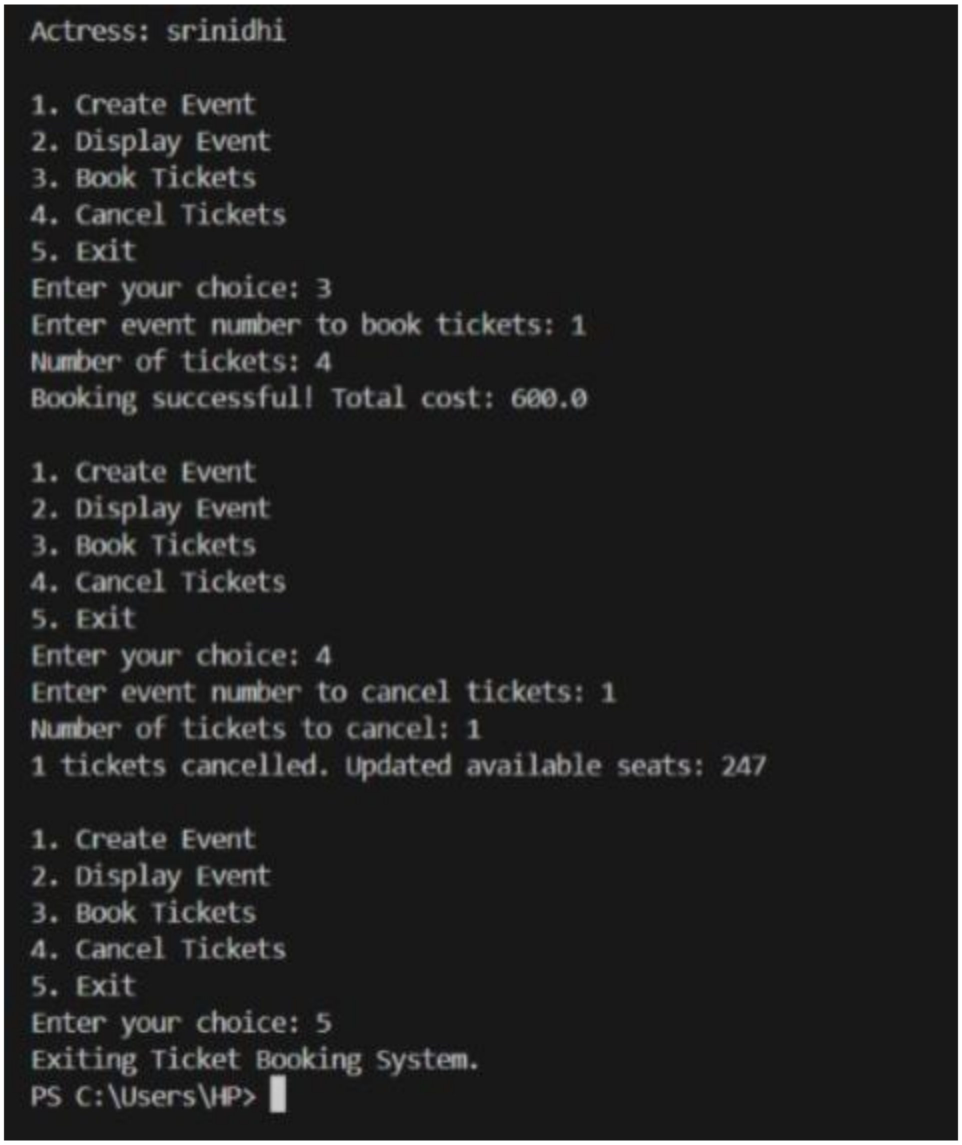
- Create a class TicketBookingSystem with the following methods:
    - create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: f loat, event_type: str, venu_name:str): Create a new event with the specified details and event type (movie, sport or concert) and return event object.
    - display_event_details(event: Event): Accepts an event object and calls its display_event_details() method to display event details.
    - book_tickets(event: Event, num_tickets: int):
        1. Accepts an event object and the number of tickets to be booked.
        2. Checks if there are enough available seats for the booking.
        3. If seats are available, updates the available seats and returns the total cost of the booking.
        4. If seats are not available, displays a message indicating that the event is sold out.
    - cancel_tickets(event: Event, num_tickets): cancel a specified number of tickets for an event.

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

```
class TicketBookingSystem:

    def create_event(self, event_name, date, time, total_seats, ticket_price,
event_type, venue_name, **kwargs):

        if event_type.lower() == 'movie':

            return Movie(event_name, date, time, total_seats, ticket_price,
venue_name, kwargs.get('genre', "), kwargs.get('actor_name', "),
kwargs.get('actress_name', "))

        elif event_type.lower() == 'concert':

            return Concert(event_name, date, time, total_seats, ticket_price,
venue_name, kwargs.get('artist', "), kwargs.get('concert_type', "))

        elif event_type.lower() == 'sports':

            return Sports(event_name, date, time, total_seats, ticket_price,
venue_name, kwargs.get('sport_name', "), kwargs.get('teams_name', "))

        else:

            print("Invalid event type!")

            return None


    def display_event_details(self, event: Event):

        event.display_event_details()


    def book_tickets(self, event: Event, num_tickets: int):

        if event.total_seats >= num_tickets:

            event.total_seats -= num_tickets

            total_cost = num_tickets * event.ticket_price
            print(f"Booking successful! Total cost: {total_cost}")
        else:
            print("Booking failed! Not enough seats available.")

    def cancel_tickets(self, event: Event, num_tickets: int):
        event.total_seats += num_tickets
        print(f"{num_tickets} tickets cancelled. Updated available seats: 
{event.total_seats}")

    def main(self):
        system = TicketBookingSystem()
        events = []

        while True:
            print("\n1. Create Event\n2. Display Event\n3. Book Tickets\n4. Cancel 
Tickets\n5. Exit")
            choice = input("Enter your choice: ")

            if choice == '1':
                etype = input("Enter event type (movie/concert/sports): ").lower()
                name = input("Event name: ")
                date = input("Date: ")
                time = input("Time: ")
                seats = int(input("Total seats: "))
                price = float(input("Ticket price: "))
                venue = input("Venue name: ")

                if etype == 'movie':
                    genre = input("Genre: ")
                    actor = input("Actor: ")

                    actress = input("Actress: ")

                    e = system.create_event(name, date, time, seats, price, etype, venue,
genre=genre, actor_name=actor, actress_name=actress)


                elif etype == 'concert':

                    artist = input("Artist: ")

                    ctype = input("Concert type: ")

                    e = system.create_event(name, date, time, seats, price, etype, venue,
artist=artist, concert_type=ctype)


                elif etype == 'sports':

                    sport = input("Sport Name: ")

                    teams = input("Teams (e.g., India vs Pakistan): ")

                    e = system.create_event(name, date, time, seats, price, etype, venue,
sport_name=sport, teams_name=teams)

                else:

                    e = None


                if e:

                    events.append(e)

                    print("Event created successfully.")


            elif choice == '2':

                for idx, e in enumerate(events):

                    print(f"\nEvent {idx+1}:")

                    system.display_event_details(e)


            elif choice == '3':
                eid = int(input("Enter event number to book tickets: ")) - 1
                if 0 <= eid < len(events):
                    num = int(input("Number of tickets: "))
                    system.book_tickets(events[eid], num)
                else:
                    print("Invalid event ID.")

            elif choice == '4':
                eid = int(input("Enter event number to cancel tickets: ")) - 1
                if 0 <= eid < len(events):
                    num = int(input("Number of tickets to cancel: "))
                    system.cancel_tickets(events[eid], num)
                else:
                    print("Invalid event ID.")

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

            else:
                print("Invalid choice. Try again.")

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



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py

1. Create Event
2. Display Event
3. Book Tickets
4. Cancel Tickets
5. Exit
Enter your choice: 1
Enter event type (movie/concert/sports): movie
Event name: hit
Date: 23/06/2025
Time: 10:30 am
Total seats: 250
Ticket price: 150
Venue name: pvr
Genre: thriller
Actor: nani
Actress: srinidhi
Event created successfully.

1. Create Event
2. Display Event
3. Book Tickets
4. Cancel Tickets
5. Exit
Enter your choice: 2

Event 1:
Event: hit
Date: 23/06/2025
Time: 10:30 am
Venue: pvr
Seats: 250
Price: 150.0
Genre: thriller
Actor: nani
Actress: srinidhi
```



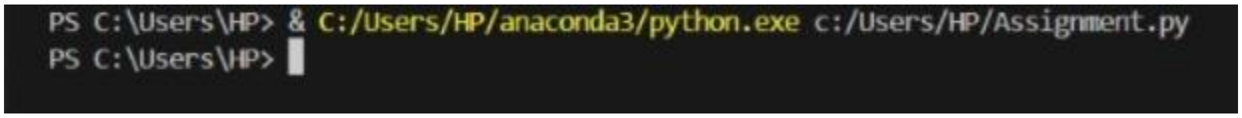
```
Actress: srinidhi

1. Create Event
2. Display Event
3. Book Tickets
4. Cancel Tickets
5. Exit
Enter your choice: 3
Enter event number to book tickets: 1
Number of tickets: 4
Booking successful! Total cost: 600.0

1. Create Event
2. Display Event
3. Book Tickets
4. Cancel Tickets
5. Exit
Enter your choice: 4
Enter event number to cancel tickets: 1
Number of tickets to cancel: 1
1 tickets cancelled. Updated available seats: 247

1. Create Event
2. Display Event
3. Book Tickets
4. Cancel Tickets
5. Exit
Enter your choice: 5
Exiting Ticket Booking System.
PS C:\Users\HP>
```

Task 6: Abstraction

Requirements:

1. Event Abstraction:
   - Create an abstract class Event that represents a generic event. It should include the following attributes and methods as mentioned in TASK 1:

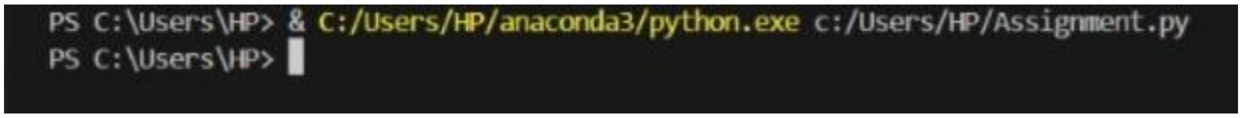
```
class Event(ABC):
    def __init__(self, name, date, time, total_seats, ticket_price, venue):
        self.name = name
        self.date = date
        self.time = time
        self.total_seats = total_seats
        self.ticket_price = ticket_price
        self.venue = venue

    @abstractmethod
    def display_event_details(self):
        pass

    def get_available_seats(self):
        return self.total_seats

    def book_seats(self, num):
        if self.total_seats >= num:
            self.total_seats -= num
            return True
        return False

    def cancel_seats(self, num):
        self.total_seats += num
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

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

```
class Movie(Event):

    def __init__(self, name, date, time, total_seats, ticket_price, venue, genre, actor, actress):

        super().__init__(name, date, time, total_seats, ticket_price, venue)

        self.genre = genre

        self.actor = actor

        self.actress = actress
    def display_event_details(self):
        print(f"[MOVIE] {self.name} | {self.genre} | {self.actor} & {self.actress} | {self.date} {self.time} @ {self.venue} | ₹{self.ticket_price} | Seats Left: {self.total_seats}")


class Concert(Event):
    def __init__(self, name, date, time, total_seats, ticket_price, venue, artist, ctype):
        super().__init__(name, date, time, total_seats, ticket_price, venue)
        self.artist = artist
        self.ctype = ctype


    def display_event_details(self):
        print(f"[CONCERT] {self.name} | {self.ctype} by {self.artist} | {self.date} {self.time} @ {self.venue} | ₹{self.ticket_price} | Seats Left: {self.total_seats}")


class Sport(Event):
    def __init__(self, name, date, time, total_seats, ticket_price, venue, sport_name, teams):
        super().__init__(name, date, time, total_seats, ticket_price, venue)
        self.sport_name = sport_name
        self.teams = teams


    def display_event_details(self):
        print(f"[SPORT] {self.name} | {self.sport_name} | {self.teams} | {self.date} {self.time} @ {self.venue} | ₹{self.ticket_price} | Seats Left: {self.total_seats}")
```

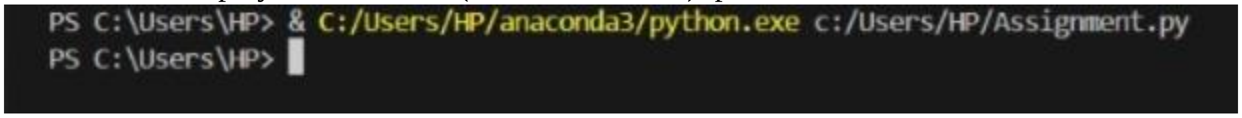


```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

3. BookingSystem Abstraction:

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

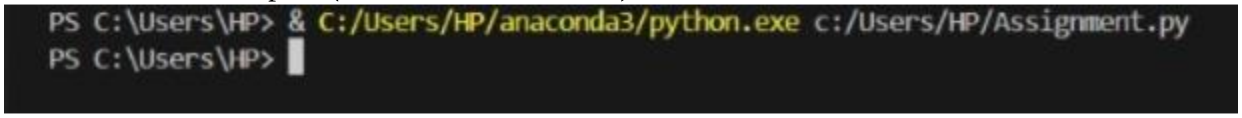
```
class BookingSystem(ABC):
    @abstractmethod
    def create_event(self, *args, **kwargs): pass

    @abstractmethod
    def book_tickets(self, event_index, num): pass

    @abstractmethod
    def cancel_tickets(self, event_index, num): pass

    @abstractmethod
    def get_available_seats(self, event_index): pass

    @abstractmethod
    def display_event_details(self, event_index): pass
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

4. Concrete TicketBookingSystem Class:
    - Create a concrete class TicketBookingSystem that inherits from BookingSystem:
        - TicketBookingSystem: Implement the abstract methods to create events, book tickets, and retrieve available seats. Maintain an array of events in this class.

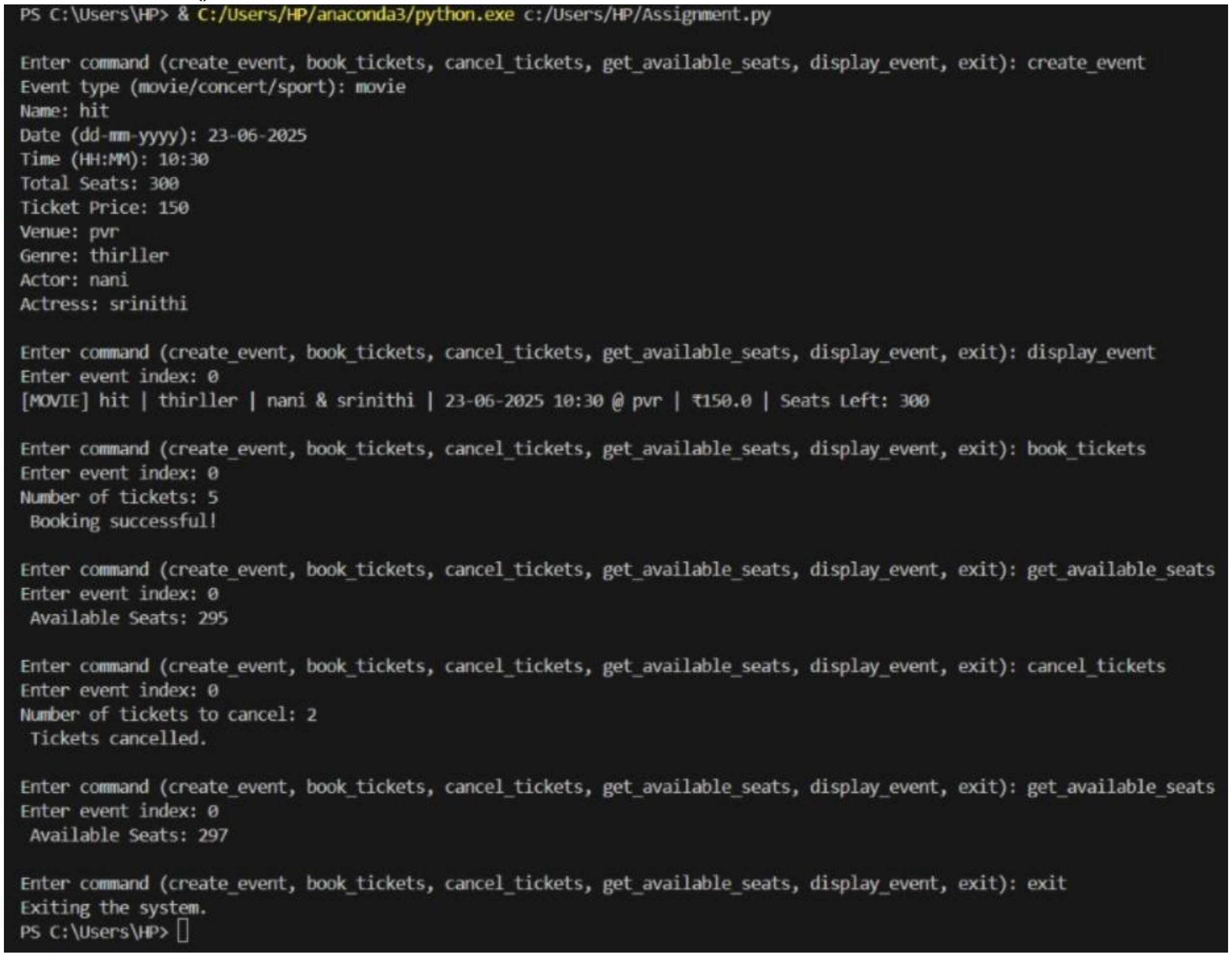
```
class TicketBookingSystem(BookingSystem):
    def __init__(self):
        self.events = []

    def create_event(self, event_type, *args):
        if event_type == "movie":
            self.events.append(Movie(*args))
        elif event_type == "concert":
            self.events.append(Concert(*args))
        elif event_type == "sport":
            self.events.append(Sport(*args))
        else:
            print("Invalid event type.")

    def book_tickets(self, event_index, num):
        if 0 <= event_index < len(self.events):
            if self.events[event_index].book_seats(num):
                print(" Booking successful!")
            else:
                print(" Not enough seats available.")
        else:
            print("Invalid event index.")

    def cancel_tickets(self, event_index, num):
        if 0 <= event_index < len(self.events):
            self.events[event_index].cancel_seats(num)
            print(" Tickets cancelled.")
        else:
            print("Invalid event index.")

    def get_available_seats(self, event_index):
        if 0 <= event_index < len(self.events):
            print(f" Available Seats:
{self.events[event_index].get_available_seats()}")
        else:
            print("Invalid event index.")

    def display_event_details(self, event_index):
        if 0 <= event_index < len(self.events):
            self.events[event_index].display_event_details()
        else:
            print("Invalid event index.")
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

- 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."

```
def main():
    system = TicketBookingSystem()
    while True:
        command = input("\nEnter command (create_event, book_tickets,
cancel_tickets, get_available_seats, display_event, exit): ")

        if command == "create_event":
            etype = input("Event type (movie/concert/sport): ").strip().lower()
            name = input("Name: ")
            date = input("Date (dd-mm-yyyy): ")
            time = input("Time (HH:MM): ")
            seats = int(input("Total Seats: "))
            price = float(input("Ticket Price: "))
            venue = input("Venue: ")

            if etype == "movie":
                genre = input("Genre: ")
                actor = input("Actor: ")
                actress = input("Actress: ")
                system.create_event(etype, name, date, time, seats, price, venue, genre,
actor, actress)
            elif etype == "concert":
                artist = input("Artist: ")
                ctype = input("Concert Type: ")
                system.create_event(etype, name, date, time, seats, price, venue, artist,
ctype)
            elif etype == "sport":
                sport_name = input("Sport Name: ")
                teams = input("Teams: ")
                system.create_event(etype, name, date, time, seats, price, venue,
sport_name, teams)
            else:
                print(" Unknown event type.")

        elif command == "book_tickets":
            idx = int(input("Enter event index: "))
            num = int(input("Number of tickets: "))
            system.book_tickets(idx, num)

        elif command == "cancel_tickets":
            idx = int(input("Enter event index: "))
            num = int(input("Number of tickets to cancel: "))
            system.cancel_tickets(idx, num)

        elif command == "get_available_seats":
            idx = int(input("Enter event index: "))
            system.get_available_seats(idx)

        elif command == "display_event":
            idx = int(input("Enter event index: "))
            system.display_event_details(idx)
        elif command == "exit":
            print("Exiting the system.")
            break

        else:
            print("Unknown command.")

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



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py

Enter command (create_event, book_tickets, cancel_tickets, get_available_seats, display_event, exit): create_event
Event type (movie/concert/sport): movie
Name: hit
Date (dd-mm-yyyy): 23-06-2025
Time (HH:MM): 10:30
Total Seats: 300
Ticket Price: 150
Venue: pvr
Genre: thirller
Actor: nani
Actress: srinithi

Enter command (create_event, book_tickets, cancel_tickets, get_available_seats, display_event, exit): display_event
Enter event index: 0
[MOVIE] hit | thirller | nani & srinithi | 23-06-2025 10:30 @ pvr | ₹150.0 | Seats Left: 300

Enter command (create_event, book_tickets, cancel_tickets, get_available_seats, display_event, exit): book_tickets
Enter event index: 0
Number of tickets: 5
 Booking successful!

Enter command (create_event, book_tickets, cancel_tickets, get_available_seats, display_event, exit): get_available_seats
Enter event index: 0
 Available Seats: 295

Enter command (create_event, book_tickets, cancel_tickets, get_available_seats, display_event, exit): cancel_tickets
Enter event index: 0
Number of tickets to cancel: 2
 Tickets cancelled.

Enter command (create_event, book_tickets, cancel_tickets, get_available_seats, display_event, exit): get_available_seats
Enter event index: 0
 Available Seats: 297

Enter command (create_event, book_tickets, cancel_tickets, get_available_seats, display_event, exit): exit
Exiting the system.
PS C:\Users\HP> 
```

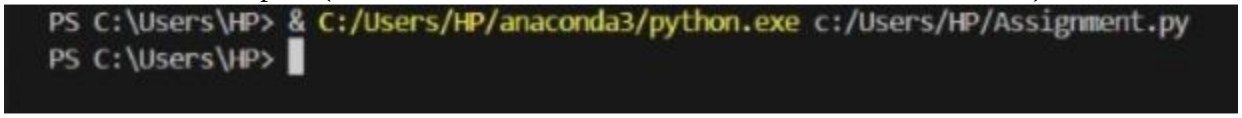
Task 7: Has A Relation / Association

Create a Following classes with the following attributes and methods:

1. Venue Class
    - Attributes:
        - venue_name,
        - address

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

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

    def get_venue_name(self): return self.venue_name
    def get_address(self): return self.address

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

    def display_venue_details(self):
        print(f"Venue: {self.venue_name}, Address: {self.address}")
```

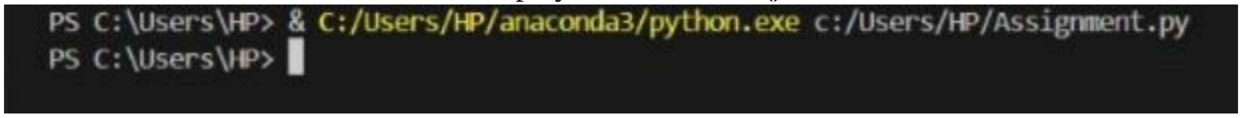


```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

2. Event Class:
   - Attributes:
     - event_name, o 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 Constuctors:
     - 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.

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

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

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

    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            return True
        return False

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

    def display_event_details(self):
        print(f"\nEvent: {self.event_name}\nDate:
{self.event_date}\nTime: {self.event_time}")
        print(f"Type: {self.event_type}, Ticket Price:
₹{self.ticket_price}, Available Seats: {self.available_seats}")
        if self.venue:
            self.venue.display_venue_details()
```

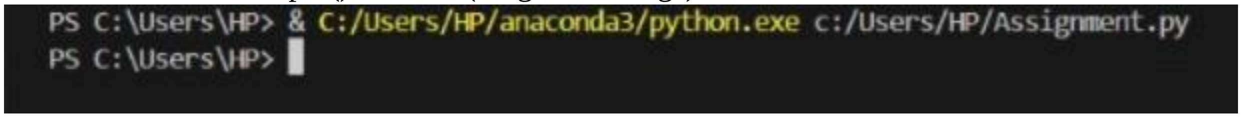


```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

3. Event sub classes:

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

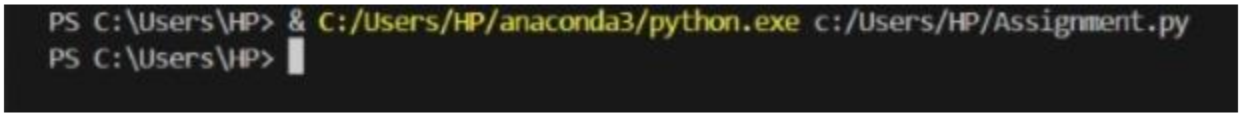
```
class Movie(Event):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

class Concert(Event):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

class Sport(Event):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

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

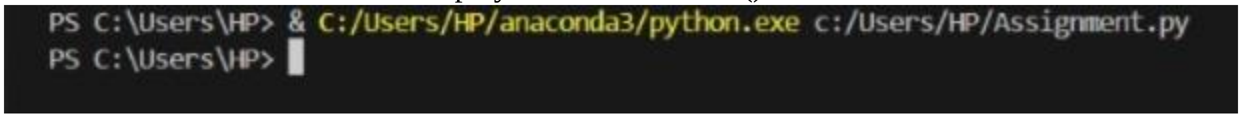
```
class Customer:
    def __init__(self, customer_name="", email="", phone_number=""):
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number

    def display_customer_details(self):
        print(f"Customer: {self.customer_name}, Email: {self.email}, Phone:
{self.phone_number}")
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

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

```
class Booking:
    booking_counter = 1

    def __init__(self, customers, event, num_tickets):
        self.bookingId = Booking.booking_counter
        Booking.booking_counter += 1
        self.customers = customers
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = num_tickets * event.ticket_price
        self.booking_date = datetime.now()

    def display_booking_details(self):
        print(f"\nBooking ID: {self.bookingId}")
        print(f"Booking Date: {self.booking_date.strftime('%Y-%m-%d
%H:%M:%S')}")
        print(f"Total Cost: ₹{self.total_cost}")
        self.event.display_event_details()
        for customer in self.customers:
            customer.display_customer_details()
```

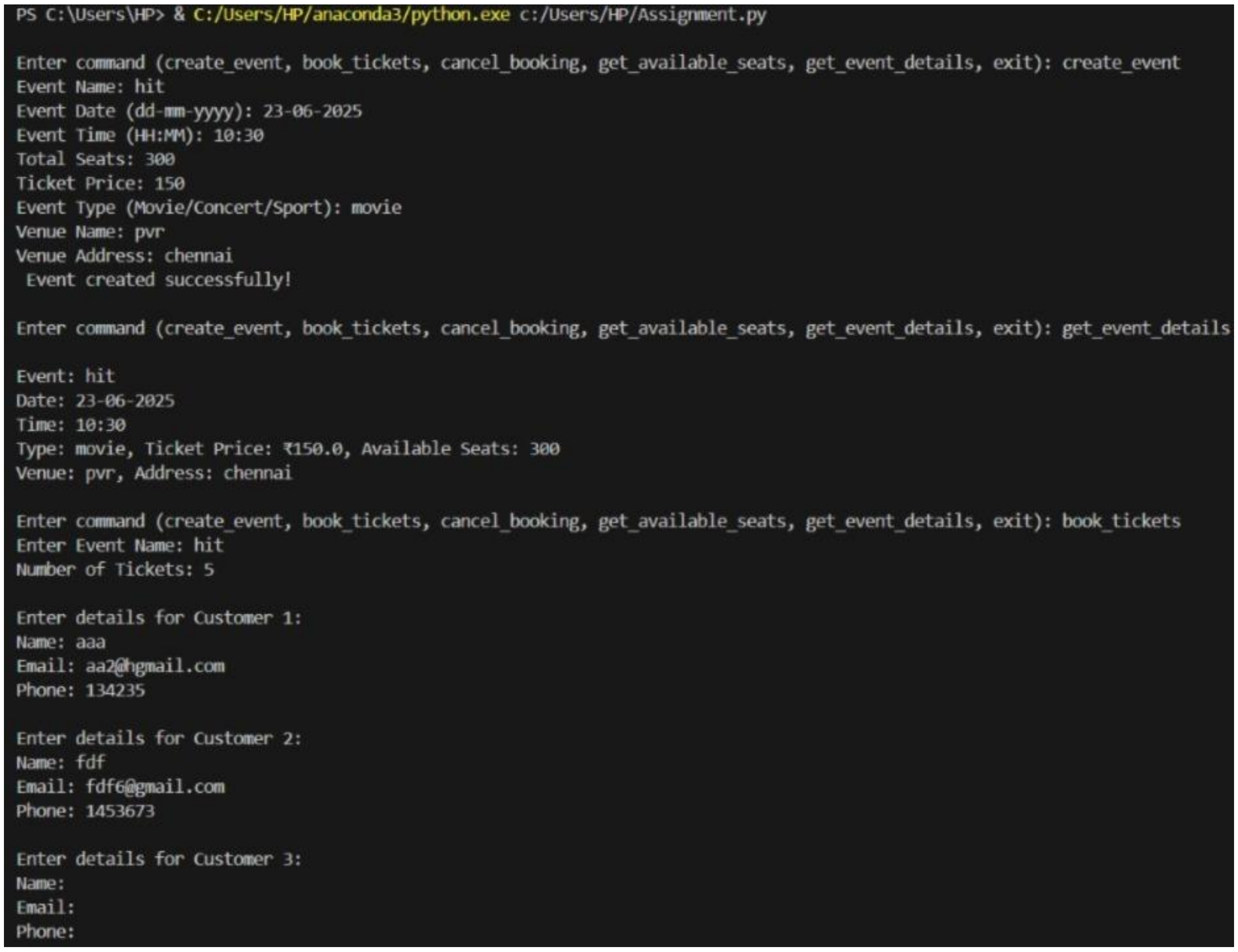


```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

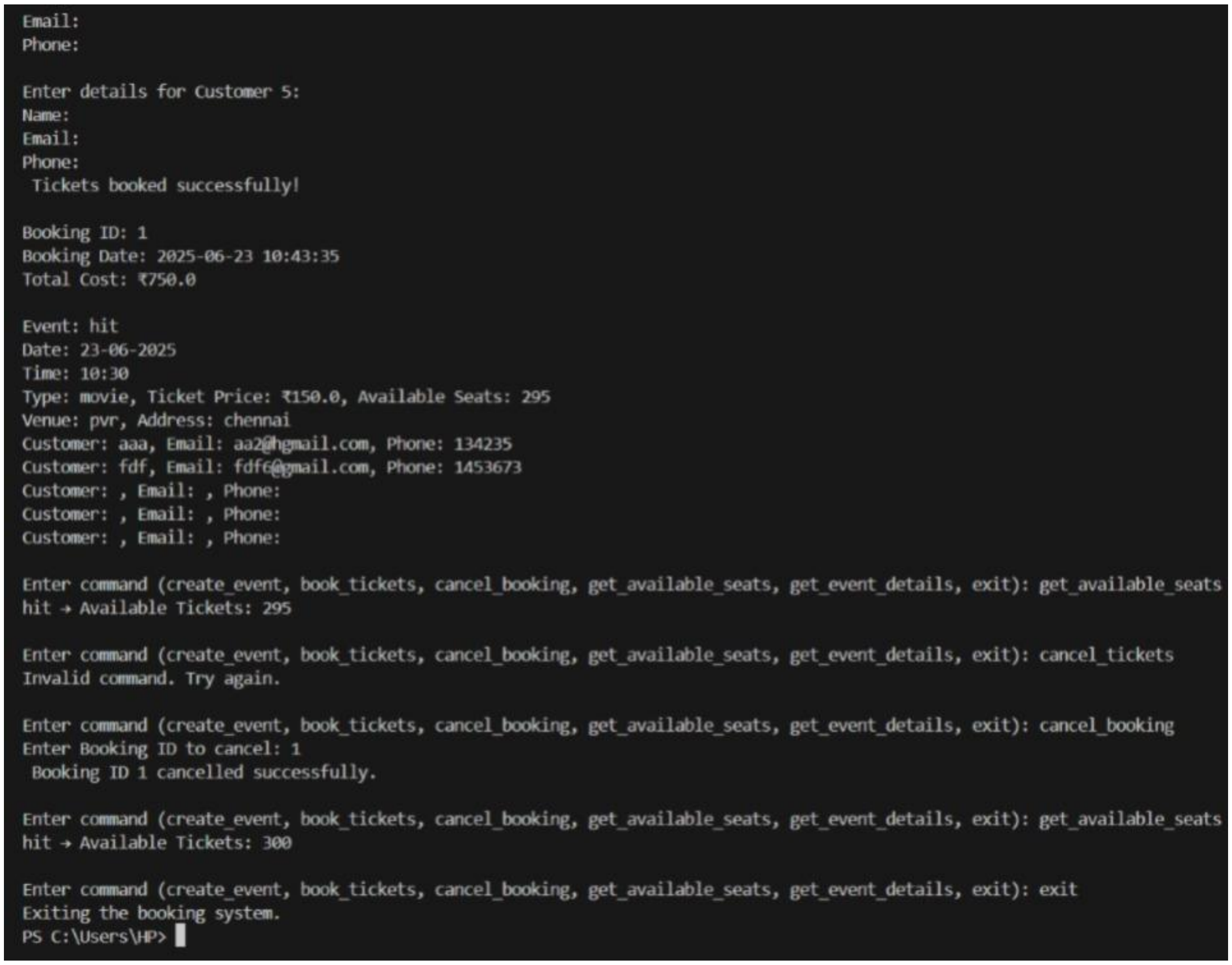
6. BookingSystem Class to represent the Ticket booking system. Perform the following operation in main method. Note: - Use Event class object for the following operation.
   - Attributes:
     - array of events

- Methods and Constuctors:
    - create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: f loat, event_type: str, venu:Venu): Create a new event with the specified details and event type (movie, sport or concert) and return event object.
    - 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."

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

    def create_event(self, event_name, date, time, total_seats, ticket_price,
event_type, venue):
        if event_type.lower() == "movie":
            event = Movie(event_name, date, time, venue, total_seats,
ticket_price, event_type)
        elif event_type.lower() == "concert":
            event = Concert(event_name, date, time, venue, total_seats,
ticket_price, event_type)
        elif event_type.lower() == "sport":
            event = Sport(event_name, date, time, venue, total_seats,
ticket_price, event_type)
        else:
            print("Invalid event type.")
            return None
        self.events.append(event)
        print(" Event created successfully!")
        return event
    def book_tickets(self, event_name, num_tickets, customer_data):
        for event in self.events:
            if event.event_name.lower() == event_name.lower():
                if event.book_tickets(num_tickets):
                    customers = [Customer(*data) for data in customer_data]
                    booking = Booking(customers, event, num_tickets)
                    self.bookings.append(booking)
                    print(" Tickets booked successfully!")
                    booking.display_booking_details()
                    return
                else:
                    print(" Not enough tickets available.")
                    return
        print(" Event not found.")

    def cancel_booking(self, booking_id):
        for i, booking in enumerate(self.bookings):
            if booking.bookingId == booking_id:
                booking.event.cancel_booking(booking.num_tickets)
                del self.bookings[i]
                print(f" Booking ID {booking_id} cancelled successfully.")
                return
        print(" Booking ID not found.")

    def getAvailableNoOfTickets(self):
        for event in self.events:
            print(f"{event.event_name} → Available Tickets: 
{event.available_seats}")

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

# Main Method
def main():
    system = BookingSystem()
    while True:
        cmd = input("\nEnter command (create_event, book_tickets, 
cancel_booking, get_available_seats, get_event_details, exit): ").strip()

        if cmd == "create_event":
            ename = input("Event Name: ")
            edate = input("Event Date (dd-mm-yyyy): ")
            etime = input("Event Time (HH:MM): ")
            seats = int(input("Total Seats: "))
            price = float(input("Ticket Price: "))
            etype = input("Event Type (Movie/Concert/Sport): ")
            vname = input("Venue Name: ")
            vaddr = input("Venue Address: ")
            venue = Venue(vname, vaddr)
            system.create_event(ename, edate, etime, seats, price, etype,
venue)

        elif cmd == "book_tickets":
            ename = input("Enter Event Name: ")
            count = int(input("Number of Tickets: "))
            customer_data = []
            for i in range(count):
                print(f"\nEnter details for Customer {i+1}:")
                name = input("Name: ")
                email = input("Email: ")
                phone = input("Phone: ")
                customer_data.append((name, email, phone))
            system.book_tickets(ename, count, customer_data)

        elif cmd == "cancel_booking":
            bid = int(input("Enter Booking ID to cancel: "))
            system.cancel_booking(bid)

        elif cmd == "get_available_seats":
            system.getAvailableNoOfTickets()

        elif cmd == "get_event_details":
            system.getEventDetails()

        elif cmd == "exit":
            print("Exiting the booking system.")
            break

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

if __name__ == '__main__':
    main()
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py

Enter command (create_event, book_tickets, cancel_booking, get_available_seats, get_event_details, exit): create_event
Event Name: hit
Event Date (dd-mm-yyyy): 23-06-2025
Event Time (HH:MM): 10:30
Total Seats: 300
Ticket Price: 150
Event Type (Movie/Concert/Sport): movie
Venue Name: pvr
Venue Address: chennai
 Event created successfully!

Enter command (create_event, book_tickets, cancel_booking, get_available_seats, get_event_details, exit): get_event_details

Event: hit
Date: 23-06-2025
Time: 10:30
Type: movie, Ticket Price: ₹150.0, Available Seats: 300
Venue: pvr, Address: chennai

Enter command (create_event, book_tickets, cancel_booking, get_available_seats, get_event_details, exit): book_tickets
Enter Event Name: hit
Number of Tickets: 5

Enter details for Customer 1:
Name: aaa
Email: aa2@hgmail.com
Phone: 134235

Enter details for Customer 2:
Name: fdf
Email: fdf6@gmail.com
Phone: 1453673

Enter details for Customer 3:
Name:
Email:
Phone:
```

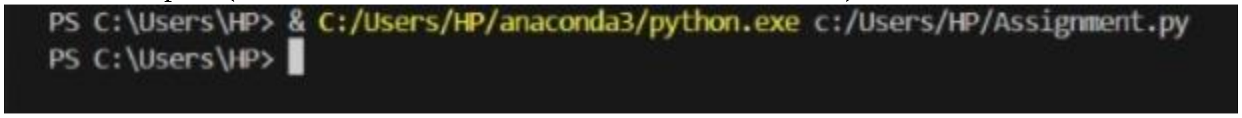


```
Email:
Phone:

Enter details for Customer 5:
Name:
Email:
Phone:
 Tickets booked successfully!

Booking ID: 1
Booking Date: 2025-06-23 10:43:35
Total Cost: ₹750.0

Event: hit
Date: 23-06-2025
Time: 10:30
Type: movie, Ticket Price: ₹150.0, Available Seats: 295
Venue: pvr, Address: chennai
Customer: aaa, Email: aa2@hgmail.com, Phone: 134235
Customer: fdf, Email: fdf6@gmail.com, Phone: 1453673
Customer: , Email: , Phone:
Customer: , Email: , Phone:
Customer: , Email: , Phone:

Enter command (create_event, book_tickets, cancel_booking, get_available_seats, get_event_details, exit): get_available_seats
hit → Available Tickets: 295

Enter command (create_event, book_tickets, cancel_booking, get_available_seats, get_event_details, exit): cancel_tickets
Invalid command. Try again.

Enter command (create_event, book_tickets, cancel_booking, get_available_seats, get_event_details, exit): cancel_booking
Enter Booking ID to cancel: 1
 Booking ID 1 cancelled successfully.

Enter command (create_event, book_tickets, cancel_booking, get_available_seats, get_event_details, exit): get_available_seats
hit → Available Tickets: 300

Enter command (create_event, book_tickets, cancel_booking, get_available_seats, get_event_details, exit): exit
Exiting the booking system.
PS C:\Users\HP>
```

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

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

```
class Venue:
    def __init__(self, name, address):
        self.name = name
        self.address = address

    def display(self):
        print(f"Venue: {self.name}, Address: {self.address}")
```

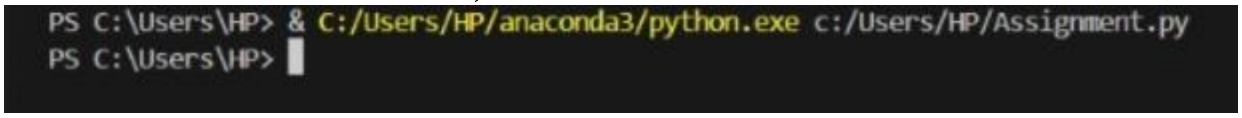


```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

2. Event Class:
   - Attributes:
     - event_name,

  - event_date DATE,
  - event_time TIME,
  - venue (reference of class Venu),
  - total_seats,
  - available_seats,
  - ticket_price DECIMAL,
  - event_type ENUM('Movie', 'Sports', 'Concert')
- Methods and Constuctors:
  - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

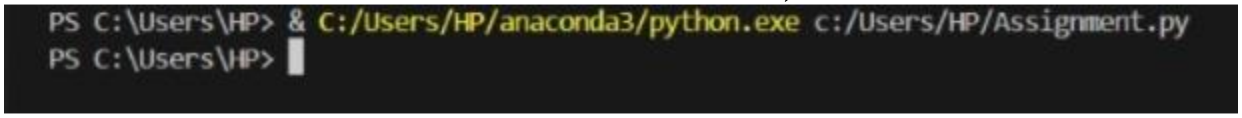
```
class EventType(Enum):
    Movie = 'Movie'
    Sports = 'Sports'
    Concert = 'Concert'

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

    def __str__(self):
        return f"Event(Name: {self.event_name}, Date: {self.event_date},
Time: {self.event_time}, Type: {self.event_type.value}, Price:
{self.ticket_price}, Venue: {self.venue}, Available:
{self.available_seats})"
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

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

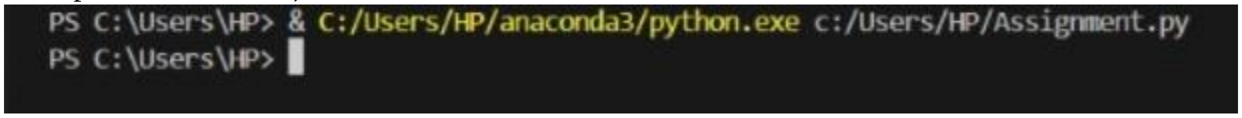
```
class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price,
genre, actor, actress):
        super().__init__(event_name, event_date, event_time, venue, total_seats,
ticket_price, "Movie")
        self.genre = genre
        self.actor = actor
        self.actress = actress

    def display_event_details(self):
        print(f"Movie: {self.event_name} ({self.genre}), Starring: {self.actor},
{self.actress}")
        print(f"Date: {self.event_date}, Time: {self.event_time}, Venue:
{self.venue.venue_name}, Available: {self.available_seats}")

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

    def display_event_details(self):
        print(f"Concert: {self.event_name} by {self.artist} ({self.ctype})")
        print(f"Date: {self.event_date}, Time: {self.event_time}, Venue:
{self.venue.venue_name}, Available: {self.available_seats}")

class Sport(Event):
    def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price,
sport_name, teams):
        super().__init__(event_name, event_date, event_time, venue, total_seats,
ticket_price, "Sports")
        self.sport_name = sport_name
        self.teams = teams

    def display_event_details(self):
        print(f"Sport: {self.sport_name} - Match: {self.teams}")
        print(f"Date: {self.event_date}, Time: {self.event_time}, Venue:
{self.venue.venue_name}, Available: {self.available_seats}")
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

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

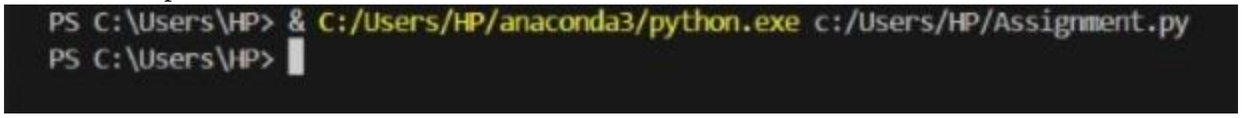
```
class Customer:
    def __init__(self, customer_name="", email="", phone_number=""):
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number

    def display_customer_details(self):
        print(f"Customer: {self.customer_name}, Email: {self.email}, Phone:
{self.phone_number}")
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

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

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

    @abstractmethod
    def getEventDetails(self):
        pass

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

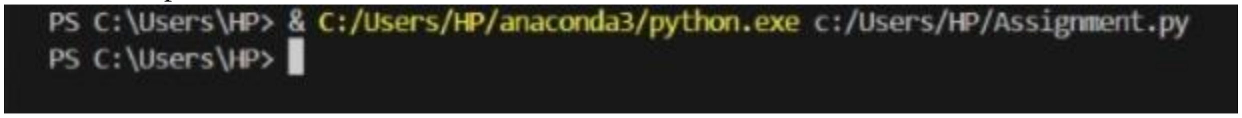


```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

6. Create interface/abstract class IBookingSystemServiceProvider with following methods:
    - calculate_booking_cost(num_tickets): Calculate and set the total cost of the booking.

- book_tickets(eventname:str, num_tickets, arrayOfCustomer): Book a specified number of t ickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class.
- cancel_booking(booking_id): Cancel the booking and update the available seats.
- get_booking_details(booking_id):get the booking details.

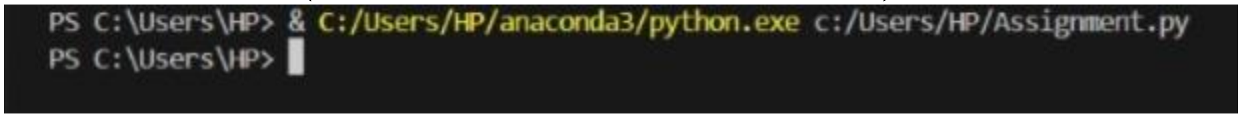
```
class IBookingSystemServiceProvider(ABC):
    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        pass

    @abstractmethod
    def book_tickets(self, event_name, num_tickets, array_of_customers):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

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



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

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

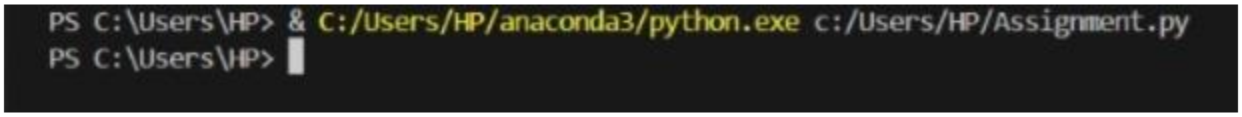
```
class EventServiceProviderImpl(IEventServiceProvider):
    def __init__(self):
        self.events = []

    def create_event(self, event_name, date, time, total_seats, ticket_price,
event_type, venue):
        e_type = EventType[event_type]
        event = Event(event_name, date, time, venue, total_seats, total_seats,
ticket_price, e_type)
        self.events.append(event)
        return event

    def getEventDetails(self):
        return self.events
    def getAvailableNoOfTickets(self):
        return sum(event.available_seats for event in self.events)
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> █
```

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

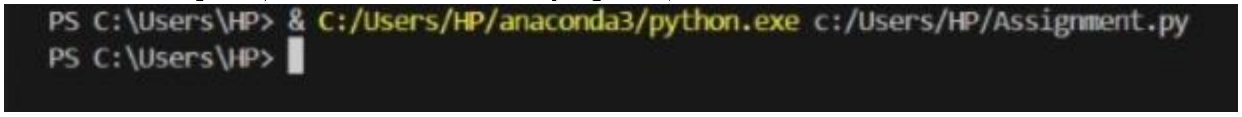
```
class BookingSystemServiceProviderImpl(EventServiceProviderImpl,
IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__()
        self.bookings = {}

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

    def book_tickets(self, event_name, num_tickets, array_of_customers):
        for event in self.events:
            if event.event_name == event_name:
                if event.available_seats >= num_tickets:
                    booking = Booking(event, array_of_customers)
                    event.available_seats -= num_tickets
                    self.bookings[booking.booking_id] = booking
                    return booking
                else:
                    return "Not enough tickets available."
        return "Event not found."

    def cancel_booking(self, booking_id):
        booking = self.bookings.get(booking_id)
        if booking:
            booking.event.available_seats += booking.num_tickets
            del self.bookings[booking_id]
            return f"Booking {booking_id} cancelled."
        return "Booking ID not found."

    def get_booking_details(self, booking_id):
        return self.bookings.get(booking_id, "Booking not found.")
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> █
```

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."

```
def main():
    print("Welcome to Ticket Booking System")
    system = BookingSystemServiceProviderImpl()

    while True:
        cmd = input("\nEnter command (create_event, book_tickets, cancel_tickets,
get_available_seats, get_event_details, exit): ").strip()

        if cmd == "create_event":
            name = input("Event name: ")
            date = input("Event date (YYYY-MM-DD): ")
            time = input("Event time (HH:MM): ")
            total = int(input("Total seats: "))
            price = float(input("Ticket price: "))
            etype = input("Event type (movie/sports/concert): ")
            vname = input("Venue name: ")
            vaddr = input("Venue address: ")
            venue = Venue(vname, vaddr)
            system.create_event(name, date, time, total, price, etype, venue)

        elif cmd == "book_tickets":
            ename = input("Event name: ")
            num = int(input("Number of tickets: "))
            customers = []
            for i in range(num):
                cname = input(f"Customer {i+1} name: ")
                email = input("Email: ")
                phone = input("Phone: ")
                customers.append(Customer(cname, email, phone))
            system.book_tickets(ename, num, customers)

        elif cmd == "cancel_tickets":
            bid = int(input("Booking ID to cancel: "))
            system.cancel_booking(bid)

        elif cmd == "get_available_seats":
            system.getAvailableNoOfTickets()

        elif cmd == "get_event_details":
            events = system.getEventDetails()
            for e in events:
                e.display_event_details()

        elif cmd == "exit":
            print("Thank you for using the system!")
            break

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

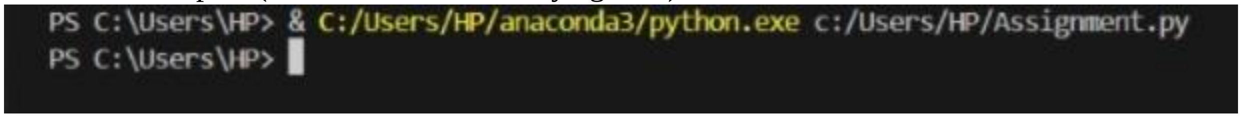


```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

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

```
def main():
    print("Welcome to Ticket Booking System")
    system = BookingSystemServiceProviderImpl()

    while True:
        cmd = input("\nEnter command (create_event, book_tickets, cancel_tickets,
get_available_seats, get_event_details, exit): ").strip()

        if cmd == "create_event":
            name = input("Event name: ")
            date = input("Event date (YYYY-MM-DD): ")
            time = input("Event time (HH:MM): ")
            total = int(input("Total seats: "))
            price = float(input("Ticket price: "))
            etype = input("Event type (movie/sports/concert): ")
            vname = input("Venue name: ")
            vaddr = input("Venue address: ")
            venue = Venue(vname, vaddr)
            system.create_event(name, date, time, total, price, etype, venue)
```

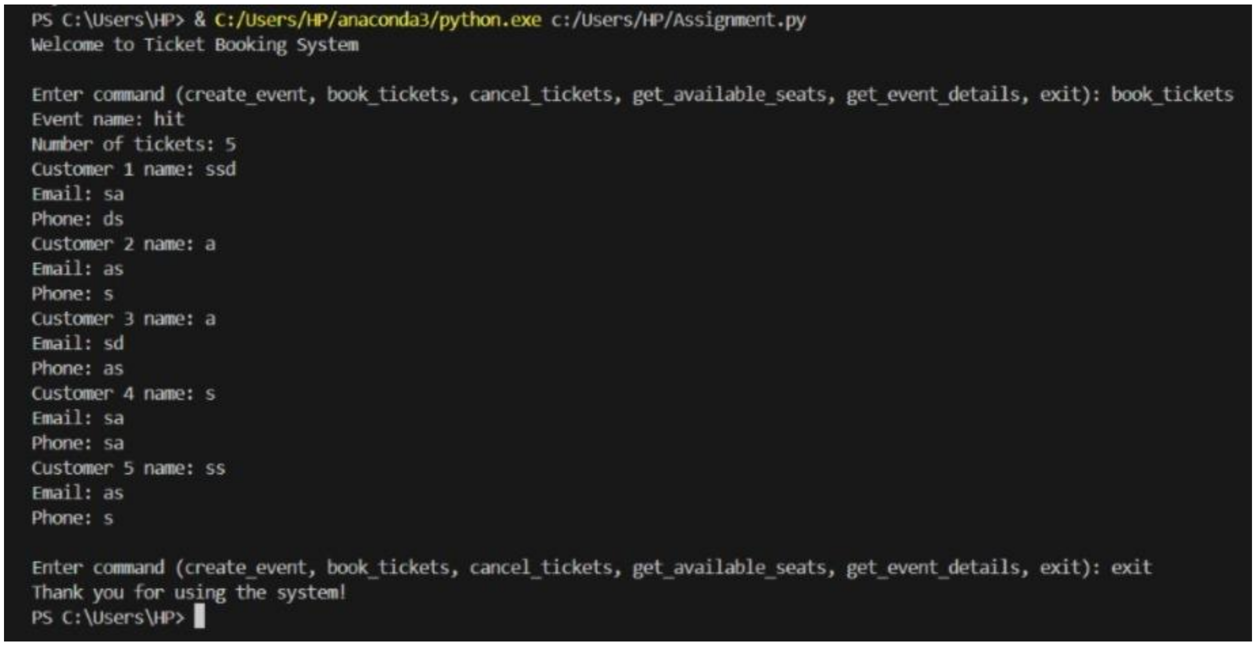
```
elif cmd == "book_tickets":
    ename = input("Event name: ")
    num = int(input("Number of tickets: "))
    customers = []
    for i in range(num):
        cname = input(f"Customer {i+1} name: ")
        email = input("Email: ")
        phone = input("Phone: ")
        customers.append(Customer(cname, email, phone))
    system.book_tickets(ename, num, customers)

elif cmd == "cancel_tickets":
    bid = int(input("Booking ID to cancel: "))
    system.cancel_booking(bid)

elif cmd == "get_available_seats":
    system.getAvailableNoOfTickets()

elif cmd == "get_event_details":
    events = system.getEventDetails()
    for e in events:
        e.display_event_details()

elif cmd == "exit":
    print("Thank you for using the system!")
    break

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



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

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

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



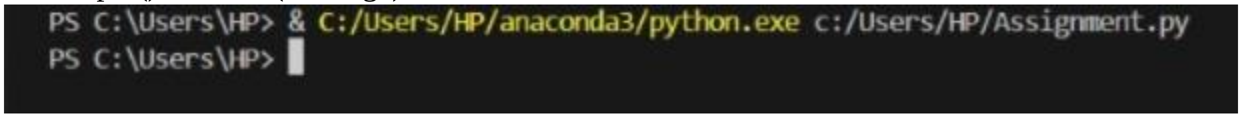
```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
Welcome to Ticket Booking System

Enter command (create_event, book_tickets, cancel_tickets, get_available_seats, get_event_details, exit): book_tickets
Event name: hit
Number of tickets: 5
Customer 1 name: ssd
Email: sa
Phone: ds
Customer 2 name: a
Email: as
Phone: s
Customer 3 name: a
Email: sd
Phone: as
Customer 4 name: s
Email: sa
Phone: sa
Customer 5 name: ss
Email: as
Phone: s

Enter command (create_event, book_tickets, cancel_tickets, get_available_seats, get_event_details, exit): exit
Thank you for using the system!
PS C:\Users\HP>
```

Task 9: Exception Handling

throw the exception whenever needed and Handle in main method,

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

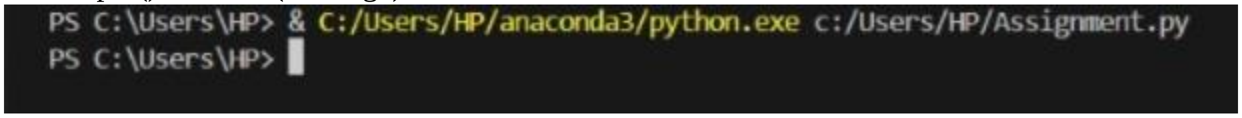
```
class EventNotFoundException(Exception):
    def __init__(self, message="Event not found!"):
        super().__init__(message)
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

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

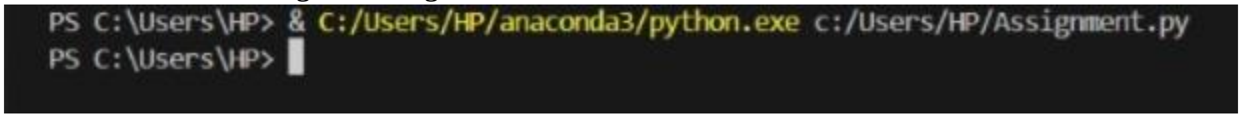
```
class InvalidBookingIDException(Exception):
    def __init__(self, message="Invalid Booking ID!"):
        super().__init__(message)
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```

3. NullPointerException handle in main method.

```
class Event:
    def __init__(self, event_id, name, total_seats, ticket_price):
        self.event_id = event_id
        self.name = name
        self.total_seats = total_seats
        self.available_seats = total_seats
        self.ticket_price = ticket_price

    def book_tickets(self, num):
        if self.available_seats >= num:
            self.available_seats -= num
            return num * self.ticket_price
        else:
            return -1

    def display_event_details(self):
        print(f"Event ID: {self.event_id}, Name: {self.name}, Available Seats:
{self.available_seats}")

class Booking:
    def __init__(self, booking_id, event, num_tickets, total_cost):
        self.booking_id = booking_id
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = total_cost

    def display_booking_details(self):
        print(f"Booking ID: {self.booking_id}, Event: {self.event.name}, Tickets:
{self.num_tickets}, Cost: {self.total_cost}")

class TicketBookingSystem:
    def __init__(self):
        self.events = {}
        self.bookings = {}
        self.booking_counter = 1

    def add_event(self, event: Event):
        self.events[event.event_id] = event

    def book_tickets(self, event_id, num_tickets):
        if event_id not in self.events:
            raise EventNotFoundException(f"Event with ID {event_id} does not exist.")
        event = self.events[event_id]
        cost = event.book_tickets(num_tickets)
        if cost == -1:
            print("Not enough seats available.")
            return None
        booking = Booking(self.booking_counter, event, num_tickets, cost)
        self.bookings[self.booking_counter] = booking
        self.booking_counter += 1
        return booking

    def cancel_booking(self, booking_id):
        if booking_id not in self.bookings:
            raise InvalidBookingIDException(f"Booking ID {booking_id} is invalid.")
        booking = self.bookings.pop(booking_id)
        booking.event.available_seats += booking.num_tickets
        print("Booking canceled successfully.")

    def view_booking(self, booking_id):
        if booking_id not in self.bookings:
            raise InvalidBookingIDException(f"Booking ID {booking_id} is invalid.")
        return self.bookings[booking_id]
```



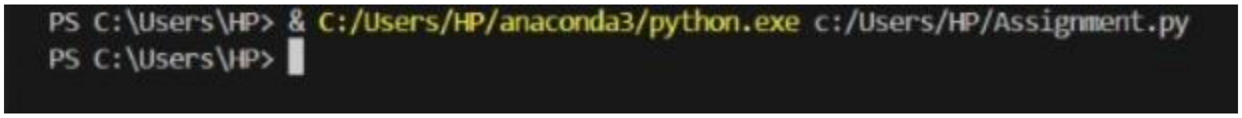
```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

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.

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.

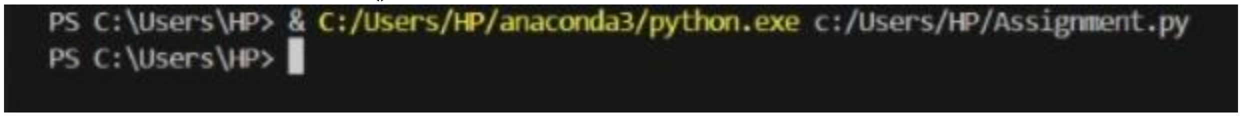
```
class Booking:
    def __init__(self, booking_id, customers: list, event, num_tickets, total_cost,
booking_date):
        self.booking_id = booking_id
        self.customers = customers  # List of Customer objects
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date

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



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

2. From the previous task change all list type of attribute to type Set in Booking and BookingSystem class and perform the same operation.
   - Avoid adding duplicate Account object to the set.
   - Create Comparator object to sort the event based on event name and location in alphabetical order.

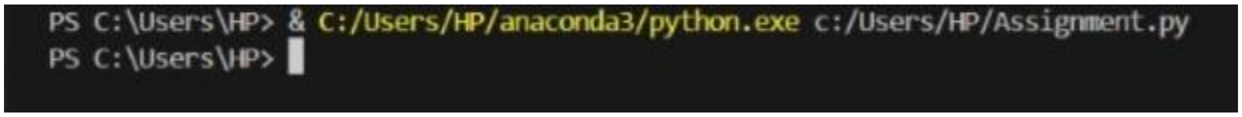
```
class Booking:
    def __init__(self, booking_id, customers: set, event, num_tickets, total_cost,
booking_date):
        self.customers = set(customers)  # Set to avoid duplicates

class TicketBookingSystem:
    def __init__(self):
        self.events = set()
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

3. From the previous task change all list type of attribute to type Map object in Booking and BookingSystem class and perform the same operation.

```
class Booking:
    def __init__(self):
        self.customers = {}  # Dictionary: {customer_id: Customer}

class TicketBookingSystem:
    def __init__(self):
        self.events = {}
```



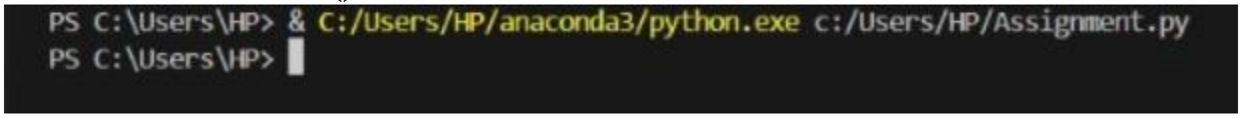
```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP> 
```

Task 11: Database Connectivity.

1. Create Venue, Event, Customer and Booking class as mentioned above Task 5.
2. Create Event sub classes as mentioned in above Task 4.
3. Create interface/abstract class IEventServiceProvider, IBookingSystemServiceProvider and its implementation classes as mentioned in above Task 5.

4. Create IBookingSystemRepository interface/abstract class which include following methods to interact with database.
    - create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu: Venu): Create a new event with the specified details and event type (movie, sport or concert) and return event object and should store in database.
    - getEventDetails(): return array of event details from the database.
    - getAvailableNoOfTickets(): return the total available tickets from the database.
    - calculate_booking_cost(num_tickets): Calculate and set the total cost of the booking.
    - book_tickets(eventname:str, num_tickets, listOfCustomer): Book a specified number of t ickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class and stored in database.
    - cancel_booking(booking_id): Cancel the booking and update the available seats and stored in database.
    - get_booking_details(booking_id): get the booking details from database.
5. Create BookingSystemRepositoryImpl interface/abstract class which implements IBookingSystemRepository interface/abstract class and provide implementation of all methods and perform the database operations.
6. Create DBUtil class and add the following method.
    - static getDBConn():Connection Establish a connection to the database and return Connection reference
7. Place the interface/abstract class in service package and interface implementation class, concrete class in bean package and TicketBookingSystemRepository class in app package.
8. Should throw appropriate exception as mentioned in above task along with handle SQLException.
9. Create TicketBookingSystem class and perform following operations:
    - Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats,", "get_event_details," and "exit."

```
import mysql.connector
from mysql.connector import Error

#  1. Connect to the existing database
def get_connection():
    try:
        conn = mysql.connector.connect(
            host='localhost',
            user='root',
            password='password',
            port=3306,
            database='ticket_booking'
        )
        return conn
    except Error as e:
        print("Error:", e)
        return None

#  2. Insert an event (data will go into your pre-created `event` table)
def insert_event():
    conn = get_connection()
    if conn:
        try:
            cursor = conn.cursor()
            sql = """
            INSERT INTO event
            (name, date, time, total_seats, available_seats, ticket_price, event_type,
venue_name, venue_address)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
            """
            values = (
                'Comedy Night', '2025-07-01', '18:30:00', 100, 100,
                350.00, 'Concert', 'Phoenix Arena', 'Main Road, Chennai'
            )
            cursor.execute(sql, values)
            conn.commit()
            print(" Event inserted successfully")
        except Error as e:
            print(" Failed to insert event:", e)
        finally:
            cursor.close()
            conn.close()

#  3. Read all events
def fetch_events():
    conn = get_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM event")
            rows = cursor.fetchall()
            print("Event Details:")
            for row in rows:
                print(row)
        except Error as e:
            print(" Failed to fetch events:", e)
        finally:
            cursor.close()
            conn.close()

#  4. Book tickets by inserting into booking and customer tables
def book_tickets(event_id, num_tickets, customers):
    conn = get_connection()
    if conn:
        try:
            cursor = conn.cursor()
            # Fetch ticket price
            cursor.execute("SELECT ticket_price, available_seats FROM event
WHERE id = %s", (event_id,))
            result = cursor.fetchone()
            if not result:
                print(" Event not found.")
                return
            price, available = result
            if available < num_tickets:
                print(" Not enough seats available.")
                return

            total_cost = price * num_tickets
            booking_id = "BK" + str(event_id) + str(num_tickets)

            # Insert into booking
            cursor.execute("INSERT INTO booking (id, event_id, num_tickets,
total_cost) VALUES (%s, %s, %s, %s)",
                           (booking_id, event_id, num_tickets, total_cost))

            # Insert each customer
            for cust in customers:
                cursor.execute("INSERT INTO customer (booking_id, name, email)
VALUES (%s, %s, %s)",
                               (booking_id, cust['name'], cust['email']))

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

            conn.commit()
            print(f" Booking successful! Booking ID: {booking_id}")
        except Error as e:
            print(" Booking failed:", e)
        finally:
            cursor.close()
            conn.close()

# 5. Cancel booking
def cancel_booking(booking_id):
    conn = get_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("SELECT event_id, num_tickets FROM booking WHERE
id = %s", (booking_id,))
            result = cursor.fetchone()
            if not result:
                print(" Booking not found.")
                return
            event_id, tickets = result

            # Delete from customer and booking
            cursor.execute("DELETE FROM customer WHERE booking_id = %s",
(booking_id,))
            cursor.execute("DELETE FROM booking WHERE id = %s",
(booking_id,))
            cursor.execute("UPDATE event SET available_seats = available_seats +
%s WHERE id = %s",
                    (tickets, event_id))

            conn.commit()
            print(" Booking cancelled successfully.")
        except Error as e:
            print(" Error cancelling booking:", e)
        finally:
            cursor.close()
            conn.close()
```



```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe c:/Users/HP/Assignment.py
PS C:\Users\HP>
```