ASSIGNMENT 5

BY SHREYASI REJA

Ticket Booking System

Control structure

Task 1: Conditional Statements

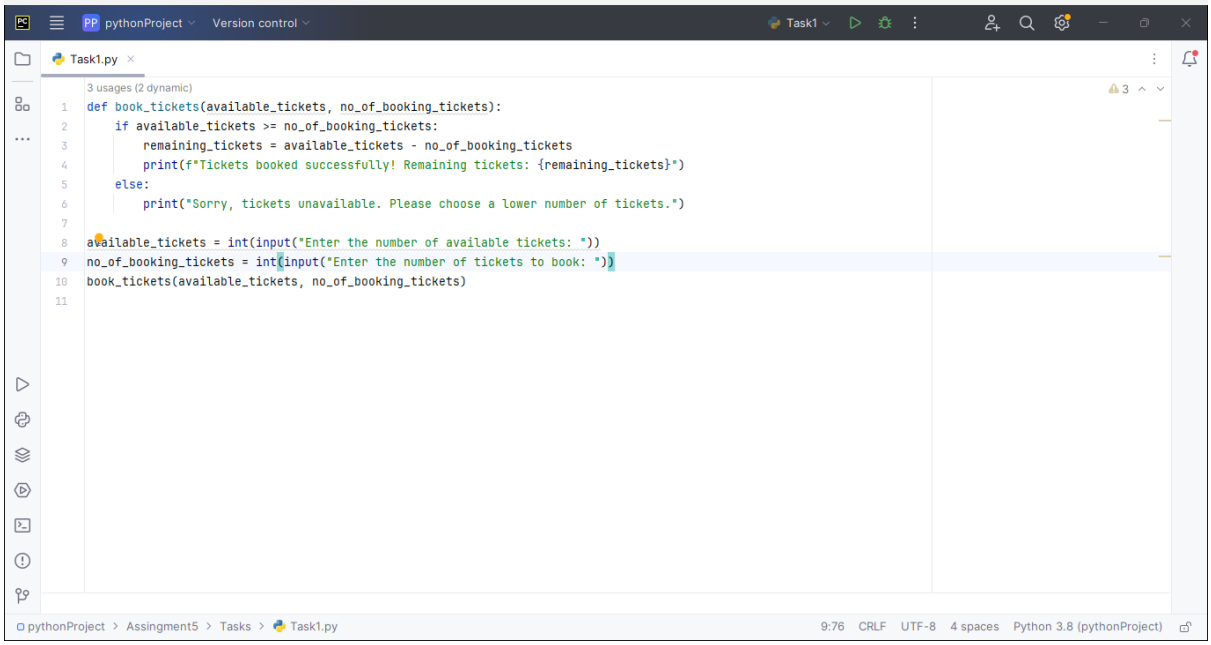
In a BookingSystem, you have been given the task is to create a program to book tickets. if available tickets more than noOfTicket to book then display the remaining tickets or ticket unavailable:

Tasks:

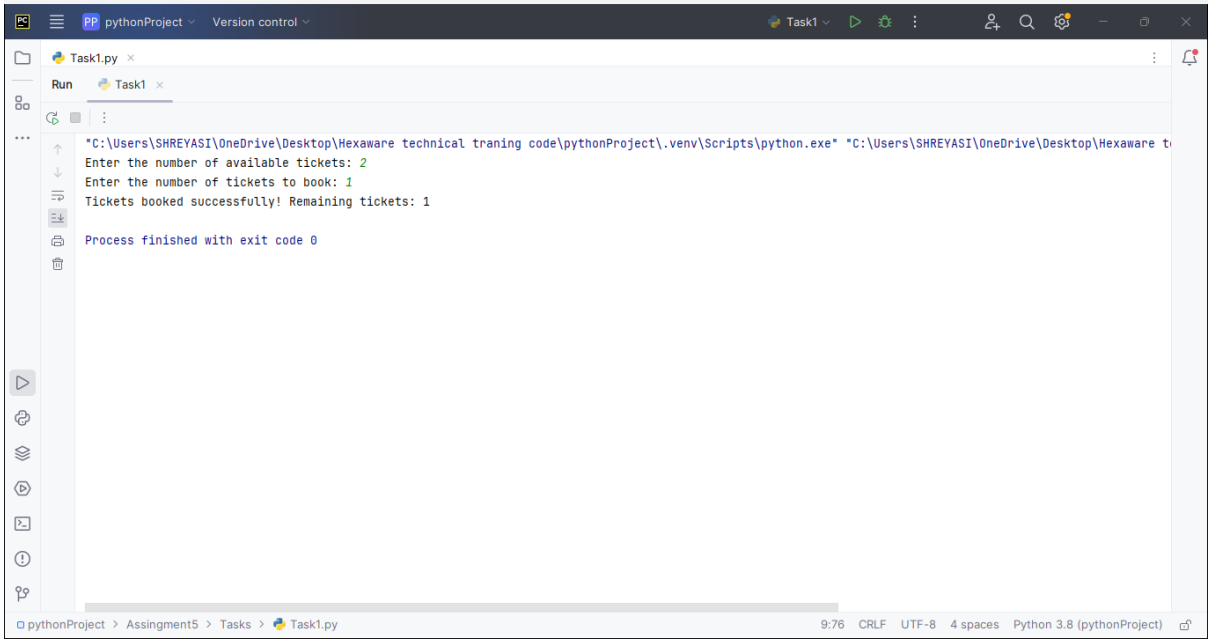
1. Write a program that takes the availableTicket and noOfBookingTicket as input.

2. Use conditional statements (if-else) to determine if the ticket is available or not.

3. Display an appropriate message based on ticket availability.



Output:-

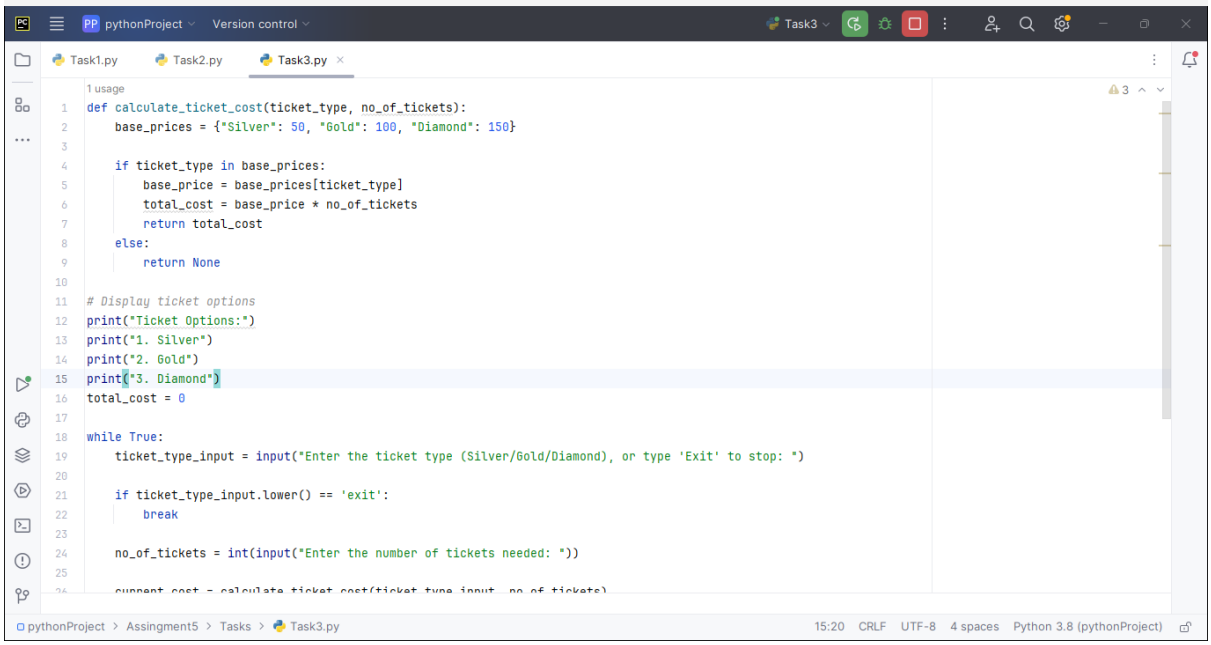


Task 2: Nested Conditional Statements

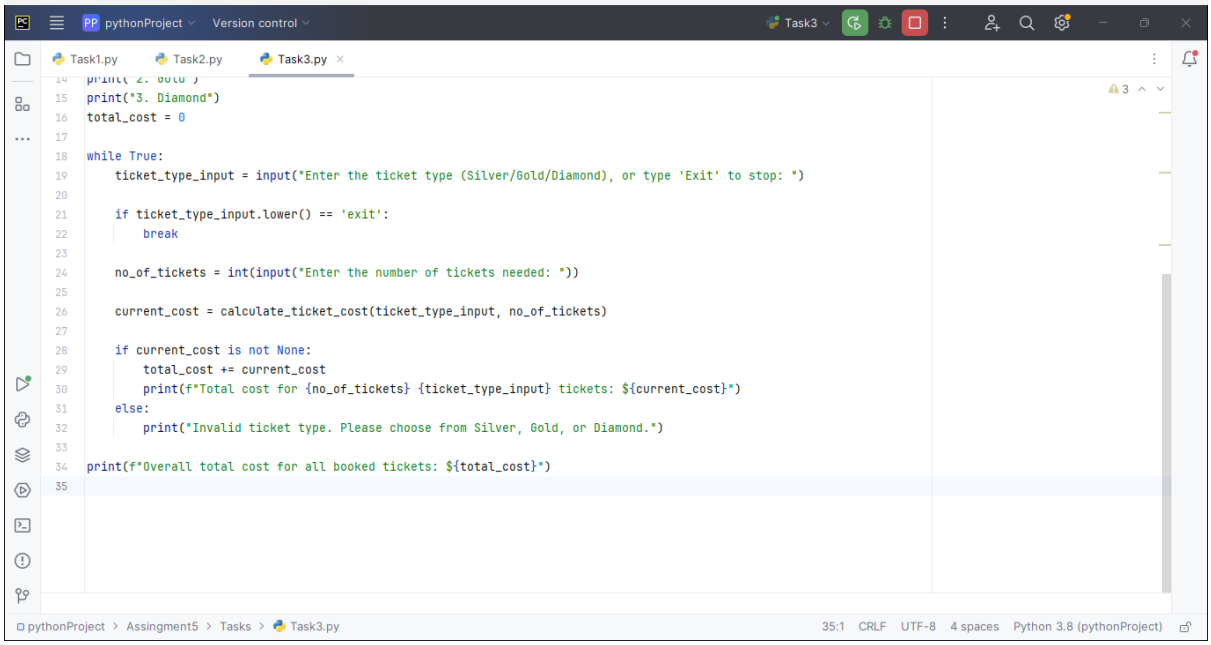
Create a program that simulates a Ticket booking and calculating cost of tickets. Display tickets options such as "Silver", "Gold", "Dimond". Based on ticket category fix the base ticket price and get the user input for ticket type and no of tickets need and calculate the total cost of tickets booked.

Task 3: Looping

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

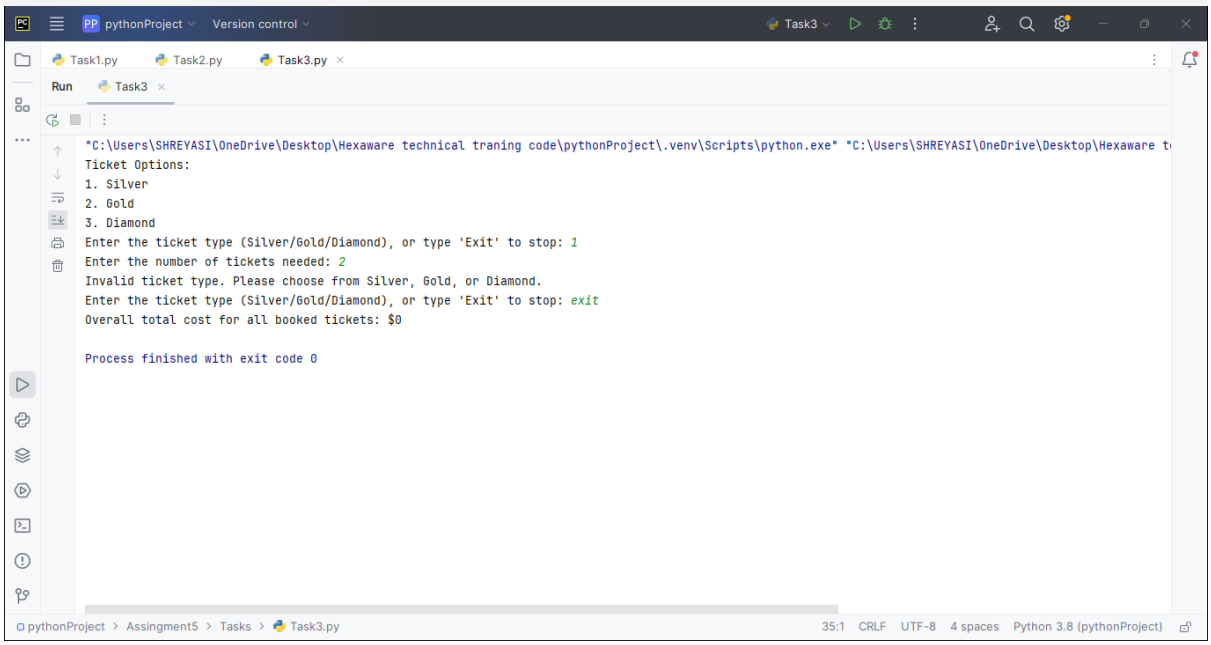


```
1 usage
def calculate_ticket_cost(ticket_type, no_of_tickets):
    base_prices = {"Silver": 50, "Gold": 100, "Diamond": 150}

    if ticket_type in base_prices:
        base_price = base_prices[ticket_type]
        total_cost = base_price * no_of_tickets
        return total_cost
    else:
        return None

# Display ticket options
print("Ticket Options:")
print("1. Silver")
print("2. Gold")
print("3. Diamond")
total_cost = 0

while True:
    ticket_type_input = input("Enter the ticket type (Silver/Gold/Diamond), or type 'Exit' to stop: ")

    if ticket_type_input.lower() == 'exit':
        break

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



```
print("3. Diamond")
total_cost = 0

while True:
    ticket_type_input = input("Enter the ticket type (Silver/Gold/Diamond), or type 'Exit' to stop: ")

    if ticket_type_input.lower() == 'exit':
        break

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

    current_cost = calculate_ticket_cost(ticket_type_input, no_of_tickets)

    if current_cost is not None:
        total_cost += current_cost
        print(f"Total cost for {no_of_tickets} {ticket_type_input} tickets: ${current_cost}")
    else:
        print("Invalid ticket type. Please choose from Silver, Gold, or Diamond.")

print(f"Overall total cost for all booked tickets: ${total_cost}")
```

Output:-



Task 4: Class & Object

Create a Following classes with the following attributes and methods:

1. Event Class:

Attributes:

o event_name,

o event_date DATE,

o event_time TIME,

o venue_name,

o total_seats,

o available_seats,

o ticket_price DECIMAL,

o event_type ENUM('Movie', 'Sports', 'Concert')

Methods and Constuctors:

o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

o calculate_total_revenue(): Calculate and return the total revenue based on the number of tickets sold.

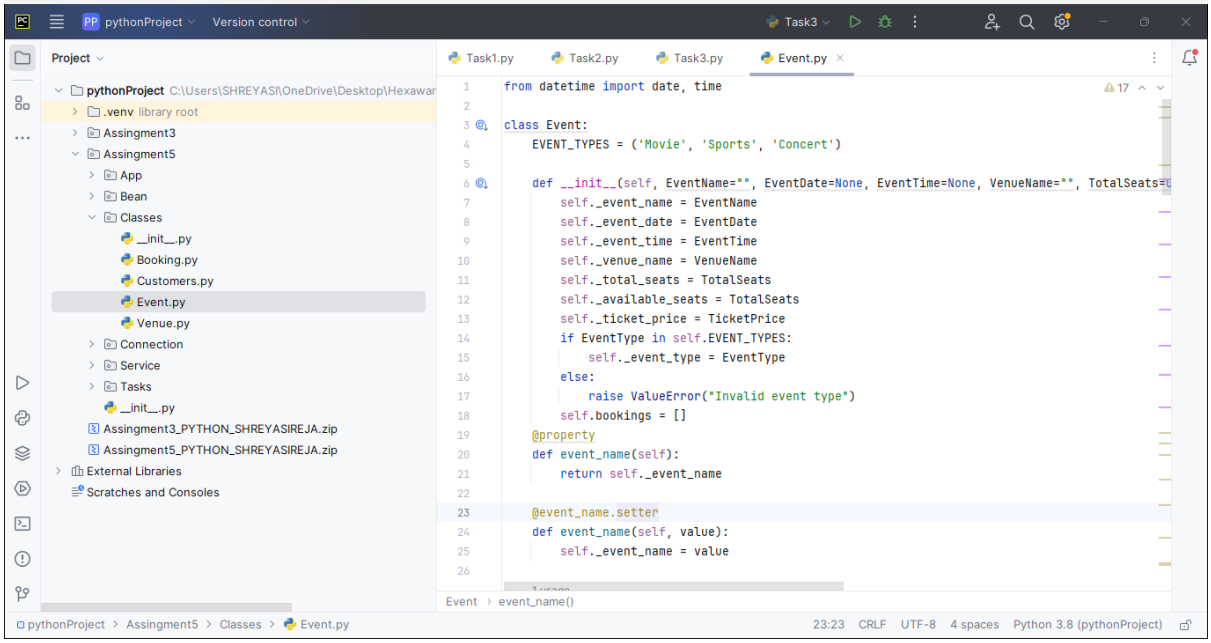
o getBookedNoOfTickets(): return the total booked tickets

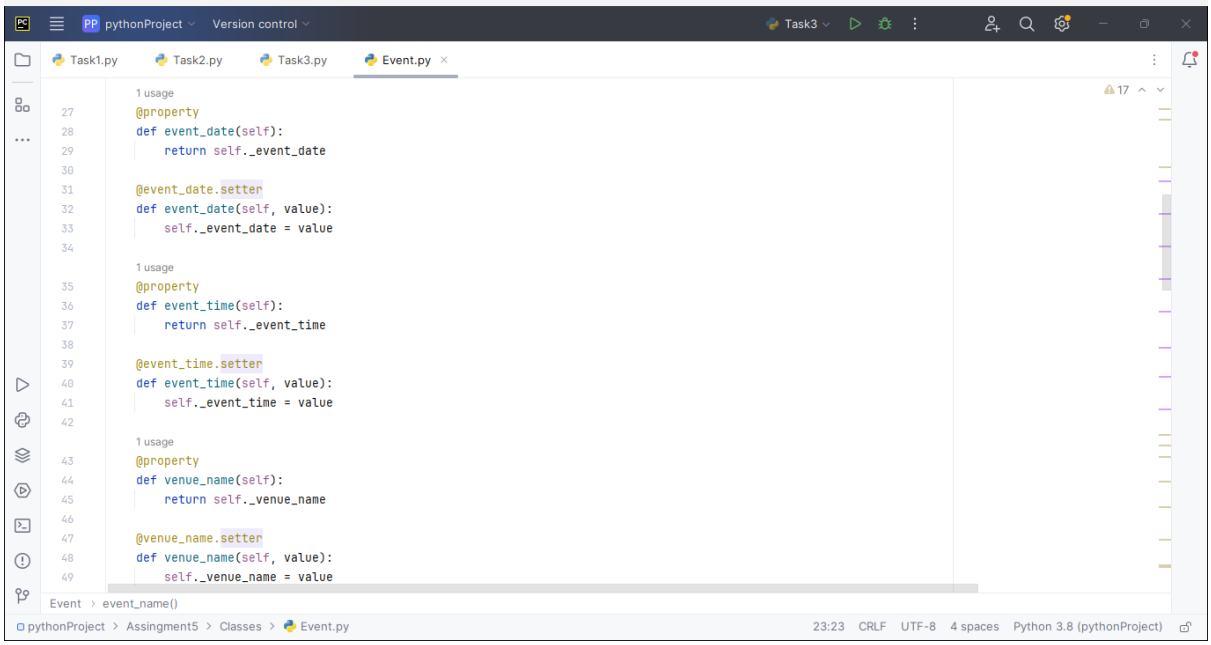
o book_tickets(num_tickets): Book a specified number of tickets for an event. Initially available seats are equal to the total seats when tickets are booked available seats number should be reduced.

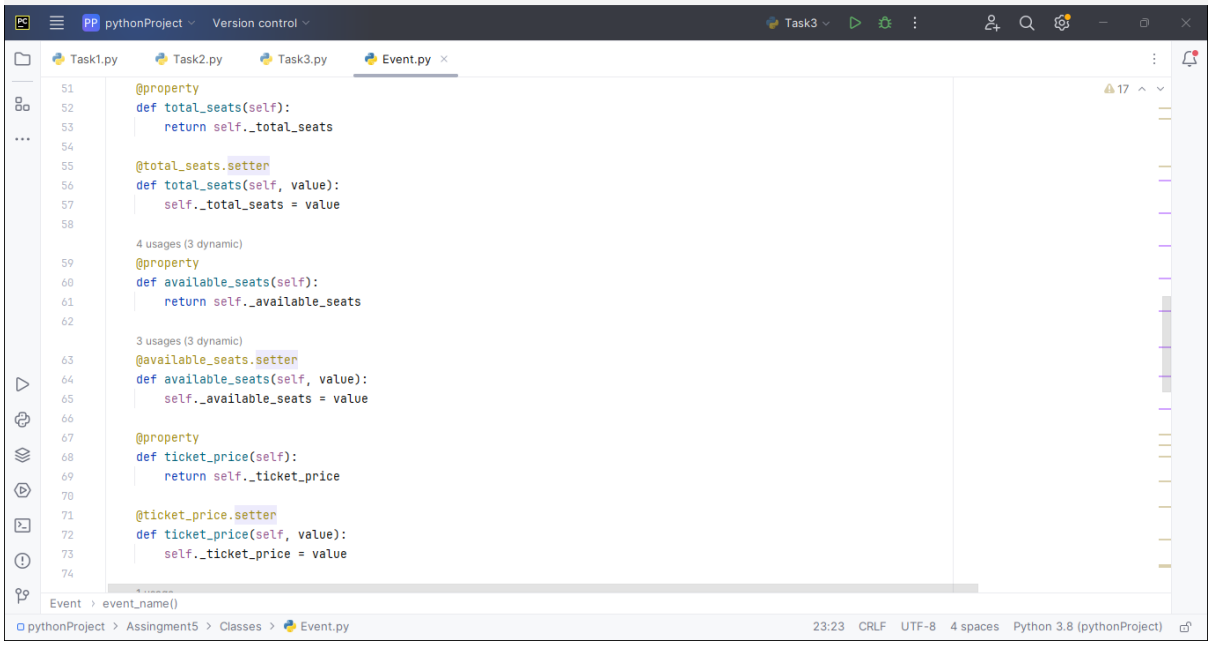
o cancel_booking(num_tickets): Cancel the booking and update the available seats.

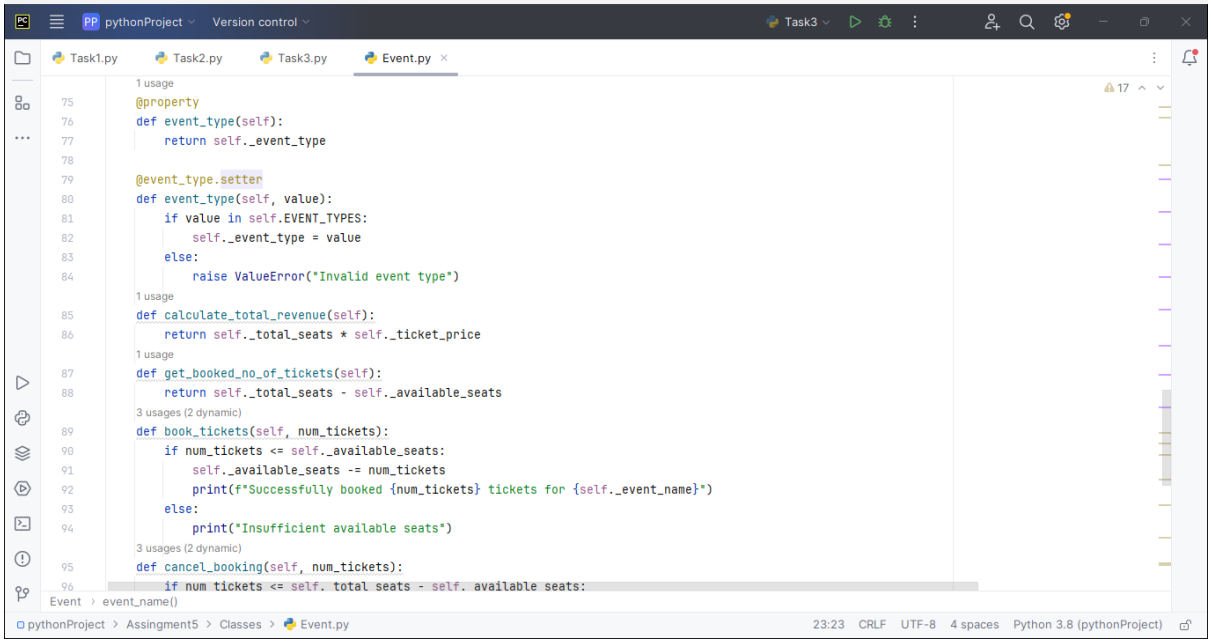
o display_event_details(): Display event details, including event name, date time seat availability.

EVENT:-

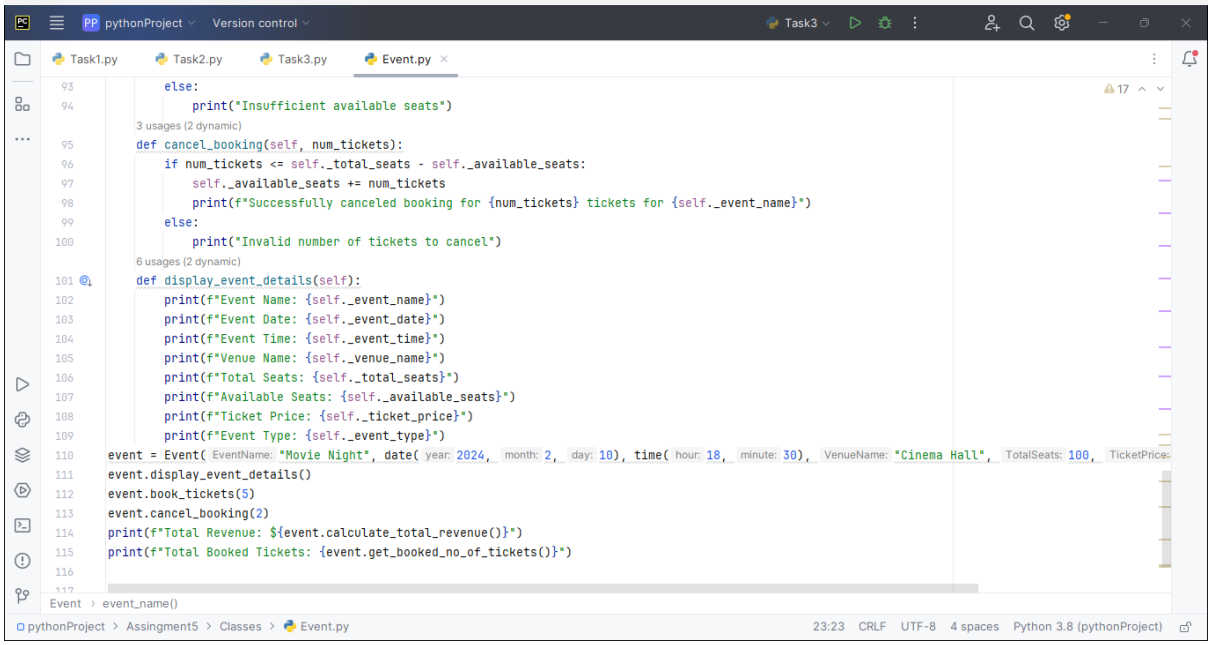



pythonProject
Version control
Task3
Task1.py
Task2.py
Task3.py
Event.py
17
1 usage
@property
def event_date(self):
return self._event_date
@event_date.setter
def event_date(self, value):
self._event_date = value
1 usage
@property
def event_time(self):
return self._event_time
@event_time.setter
def event_time(self, value):
self._event_time = value
1 usage
@property
def venue_name(self):
return self._venue_name
@venue_name.setter
def venue_name(self, value):
self._venue_name = value
Event › event_name()
pythonProject › Assignment5 › Classes › Event.py
23:23 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)


pythonProject
Version control
Task3
Task1.py
Task2.py
Task3.py
Event.py
17
@property
def total_seats(self):
return self._total_seats
@total_seats.setter
def total_seats(self, value):
self._total_seats = value
4 usages (3 dynamic)
@property
def available_seats(self):
return self._available_seats
3 usages (3 dynamic)
@available_seats.setter
def available_seats(self, value):
self._available_seats = value
@property
def ticket_price(self):
return self._ticket_price
@ticket_price.setter
def ticket_price(self, value):
self._ticket_price = value
Event › event_name()
pythonProject › Assignment5 › Classes › Event.py
23:23 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)



```
@property
def event_type(self):
    return self._event_type

@event_type.setter
def event_type(self, value):
    if value in self.EVENT_TYPES:
        self._event_type = value
    else:
        raise ValueError("Invalid event type")
def calculate_total_revenue(self):
    return self._total_seats * self._ticket_price
def get_booked_no_of_tickets(self):
    return self._total_seats - self._available_seats
def book_tickets(self, num_tickets):
    if num_tickets <= self._available_seats:
        self._available_seats -= num_tickets
        print(f"Successfully booked {num_tickets} tickets for {self._event_name}")
    else:
        print("Insufficient available seats")
def cancel_booking(self, num_tickets):
    if num_tickets <= self._total_seats - self._available_seats:
```



```
        else:
            print("Insufficient available seats")
def cancel_booking(self, num_tickets):
    if num_tickets <= self._total_seats - self._available_seats:
        self._available_seats += num_tickets
        print(f"Successfully canceled booking for {num_tickets} tickets for {self._event_name}")
    else:
        print("Invalid number of tickets to cancel")
def display_event_details(self):
    print(f"Event Name: {self._event_name}")
    print(f"Event Date: {self._event_date}")
    print(f"Event Time: {self._event_time}")
    print(f"Venue Name: {self._venue_name}")
    print(f"Total Seats: {self._total_seats}")
    print(f"Available Seats: {self._available_seats}")
    print(f"Ticket Price: {self._ticket_price}")
    print(f"Event Type: {self._event_type}")
event = Event( EventName: "Movie Night", date( year: 2024, month: 2, day: 10), time( hour: 18, minute: 30), VenueName: "Cinema Hall", TotalSeats: 100, TicketPrice:
event.display_event_details()
event.book_tickets(5)
event.cancel_booking(2)
print(f"Total Revenue: ${event.calculate_total_revenue()}")
print(f"Total Booked Tickets: {event.get_booked_no_of_tickets()}")
```

2. Venue Class

Attributes:

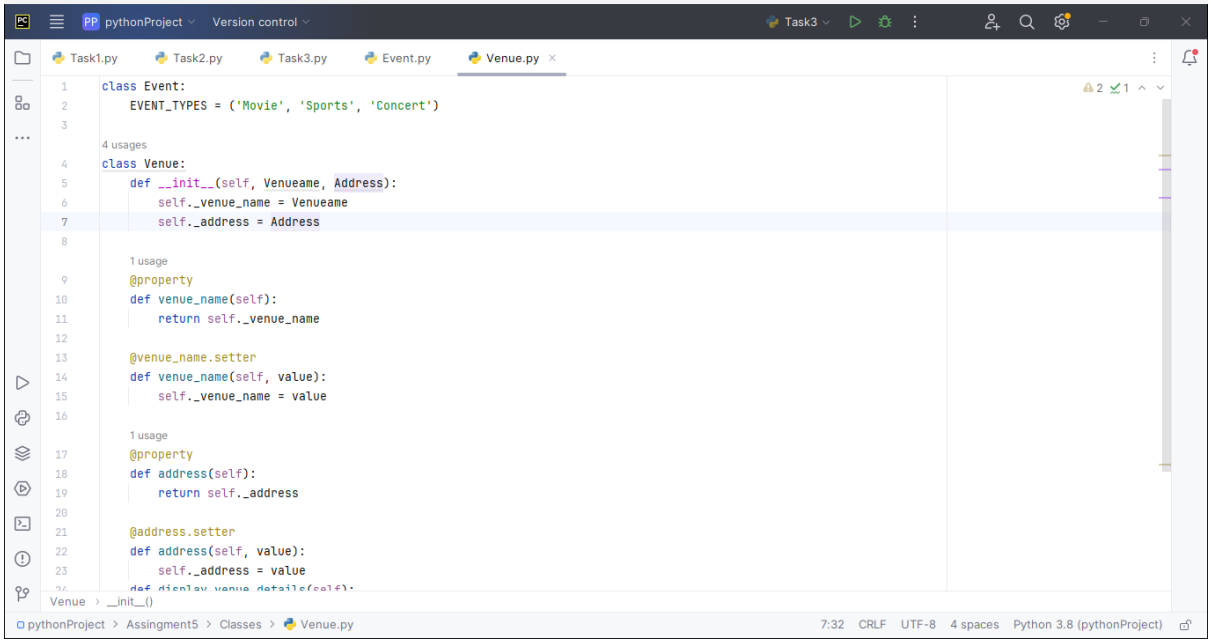
o venue_name,

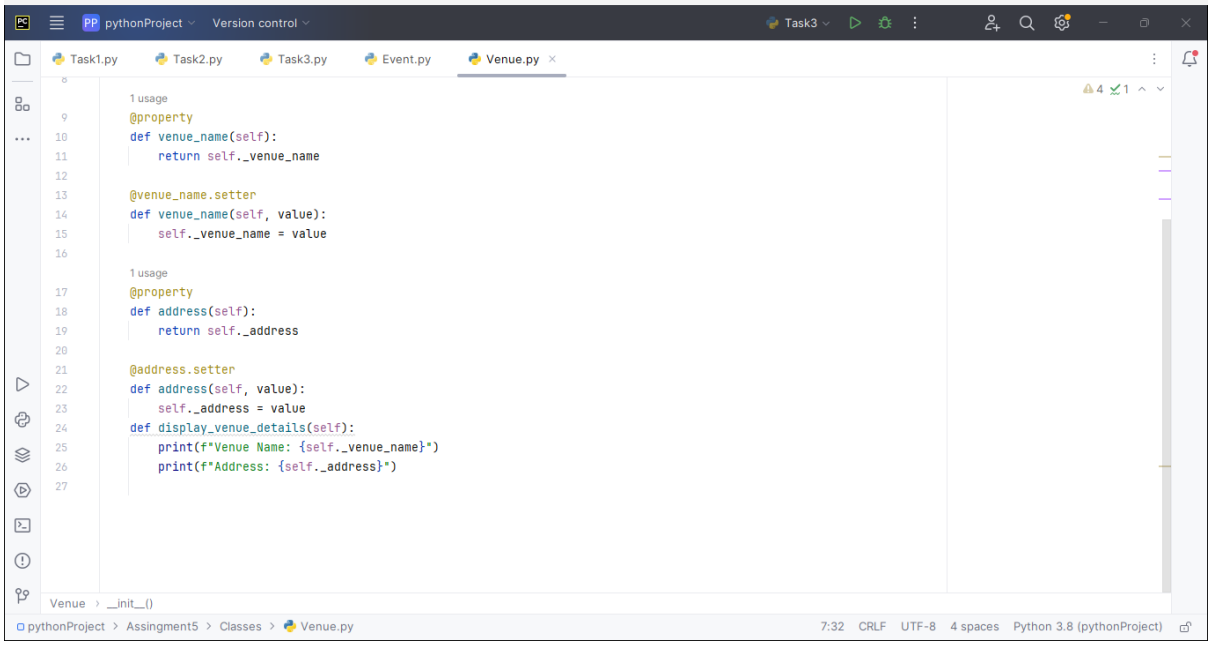
o address

Methods and Constuctors:

o display_venue_details(): Display venue details.

o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.





3. Customer Class

Attributes:

o customer_name,

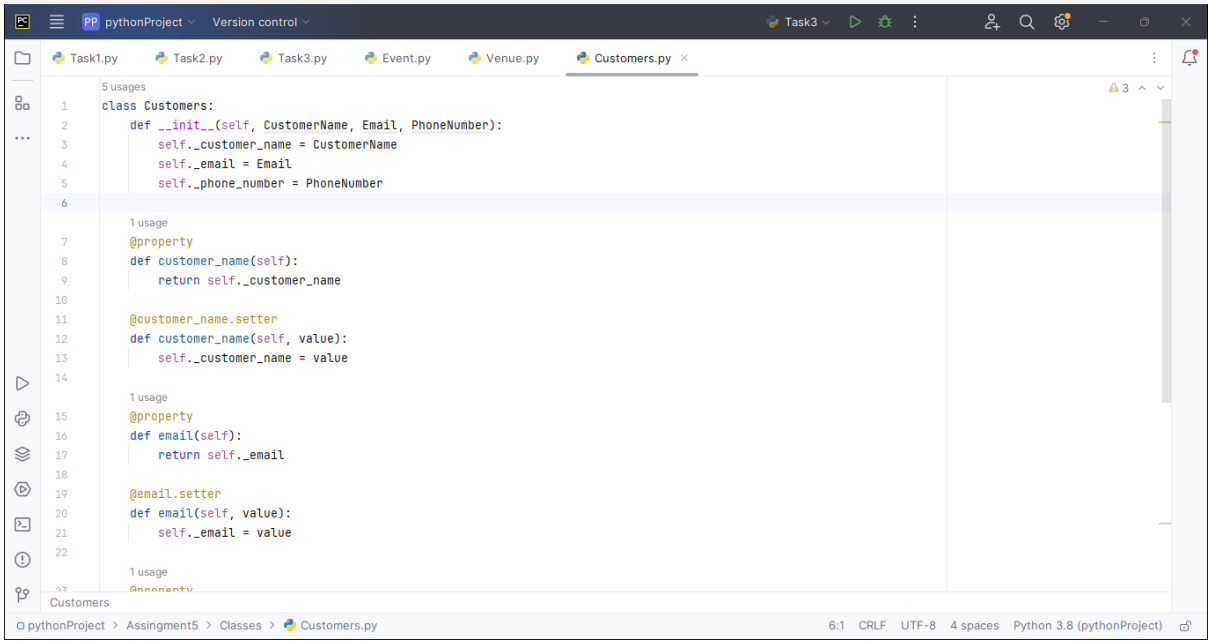
o email,

o phone_number,

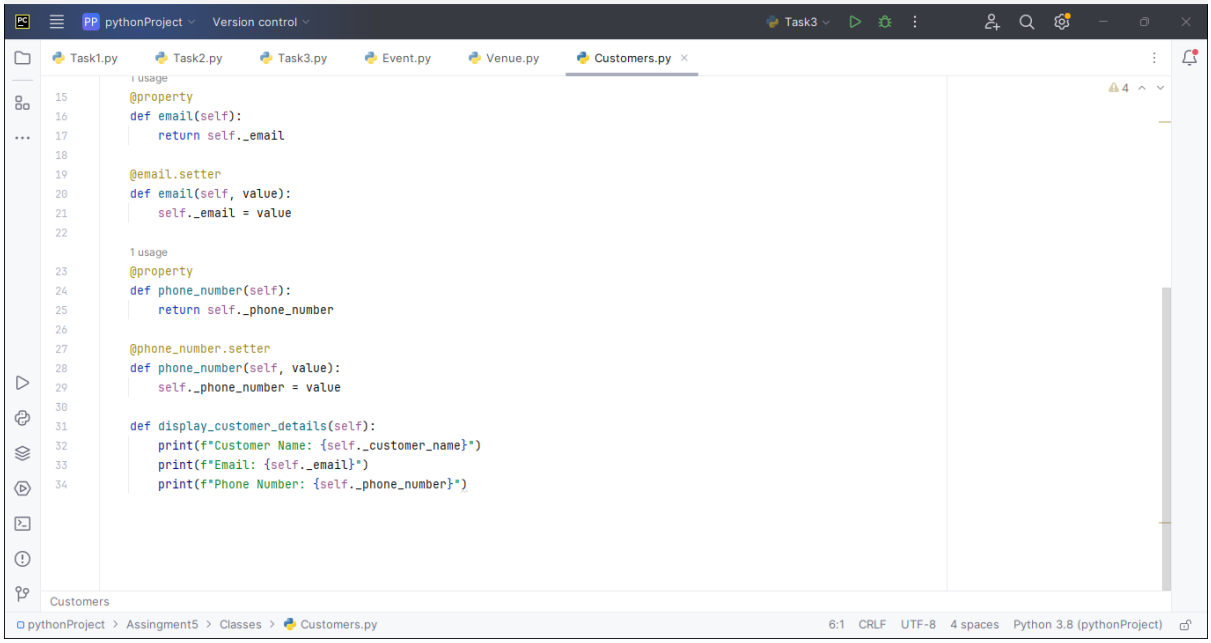
Methods and Constuctors:

o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

o display_customer_details(): Display customer details.



```
class Customers:
    def __init__(self, CustomerName, Email, PhoneNumber):
        self._customer_name = CustomerName
        self._email = Email
        self._phone_number = PhoneNumber

    @property
    def customer_name(self):
        return self._customer_name

    @customer_name.setter
    def customer_name(self, value):
        self._customer_name = value

    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, value):
        self._email = value
```



```
    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, value):
        self._email = value

    @property
    def phone_number(self):
        return self._phone_number

    @phone_number.setter
    def phone_number(self, value):
        self._phone_number = value

    def display_customer_details(self):
        print(f"Customer Name: {self._customer_name}")
        print(f"Email: {self._email}")
        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 Constuctors:

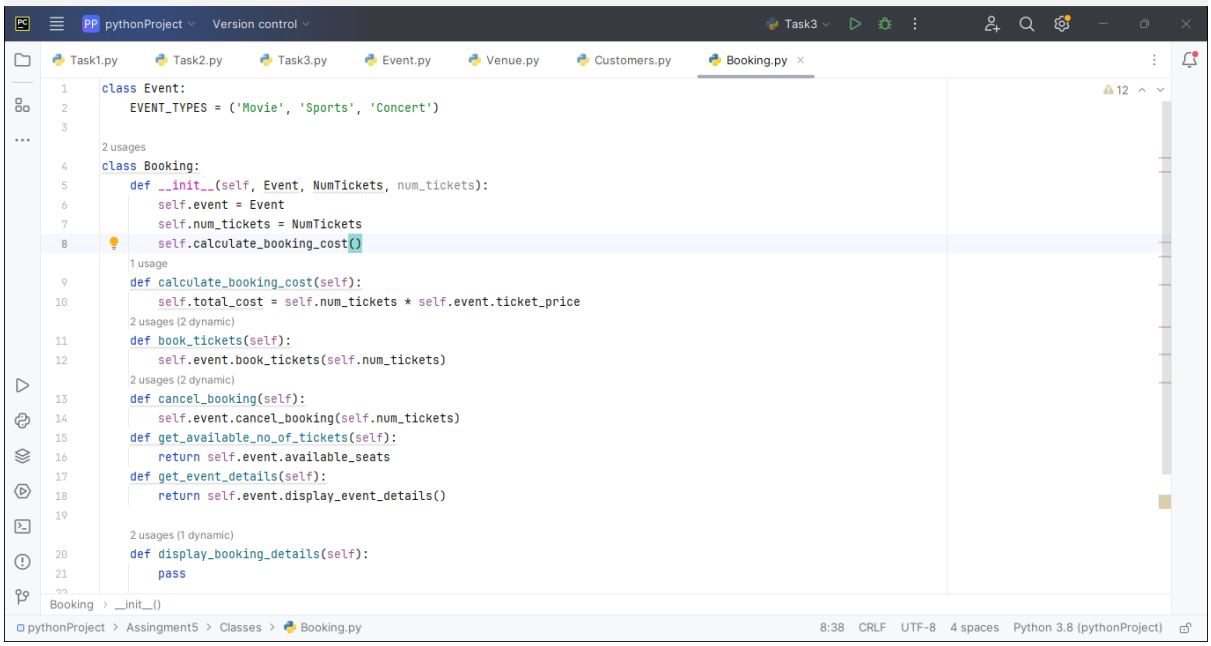
o calculate_booking_cost(num_tickets): Calculate and set the total cost of the booking.

o book_tickets(num_tickets): Book a specified number of tickets for an event.

o cancel_booking(num_tickets): Cancel the booking and update the available seats.

o getAvailableNoOfTickets(): return the total available tickets

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



Task 5: Inheritance and polymorphism

1. Inheritance

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

o Attributes:

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

2. ActorName

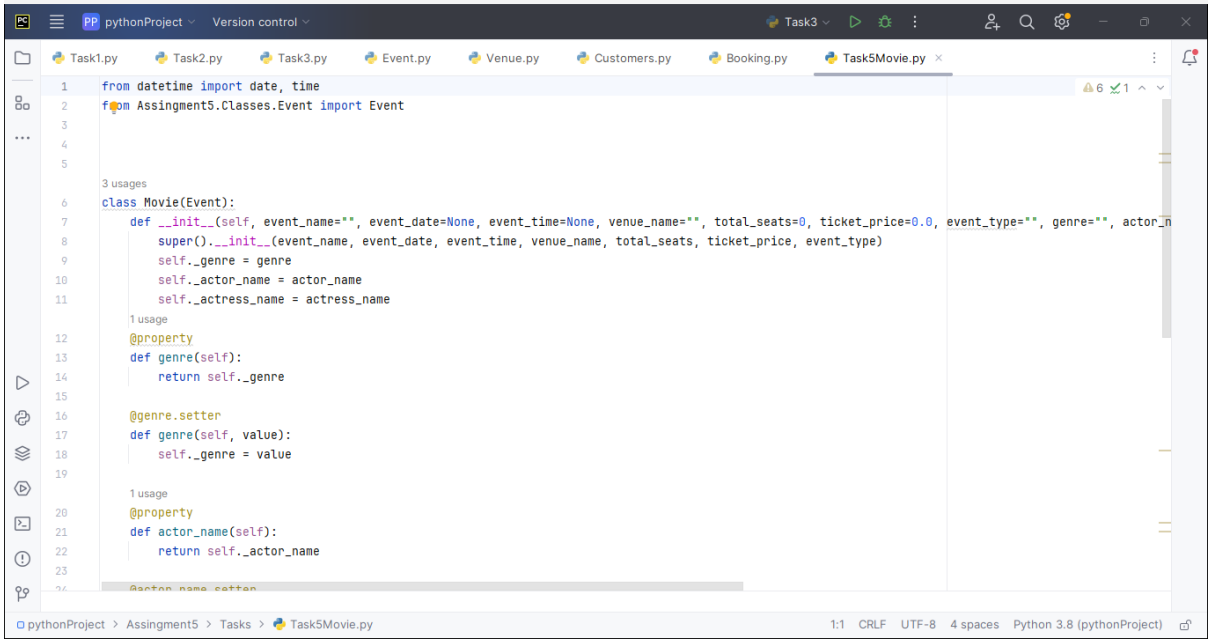
3. ActresName

o Methods:

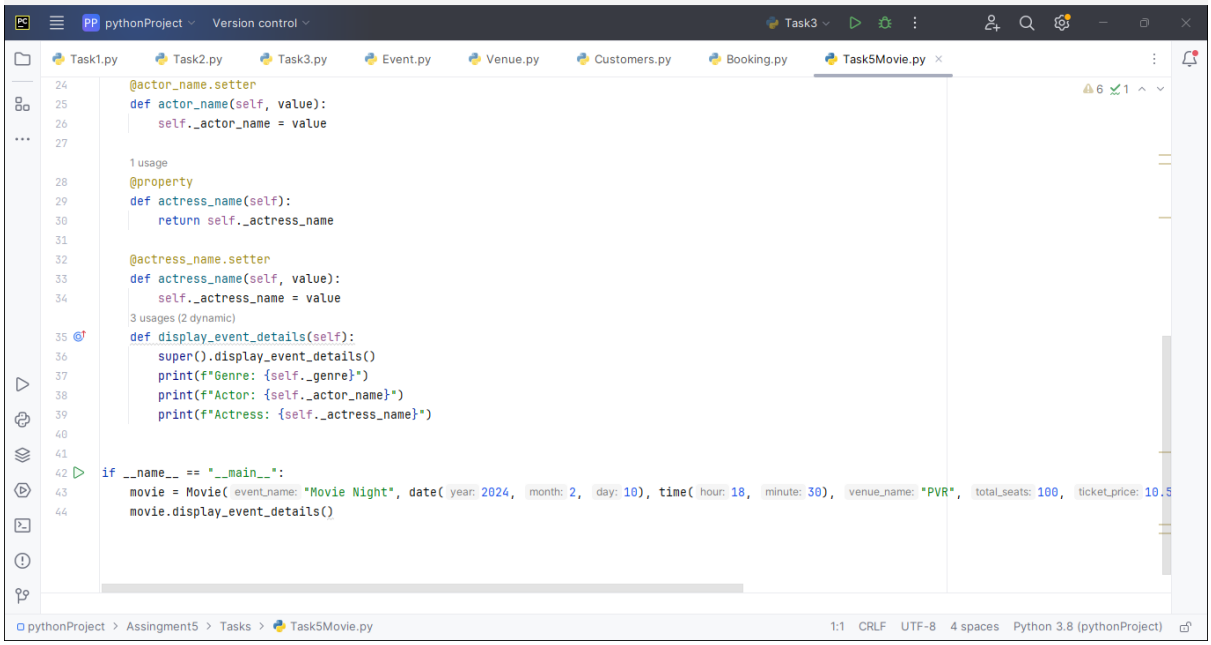
1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

2. display_event_details(): Display movie details, including genre.

MOVIE:-

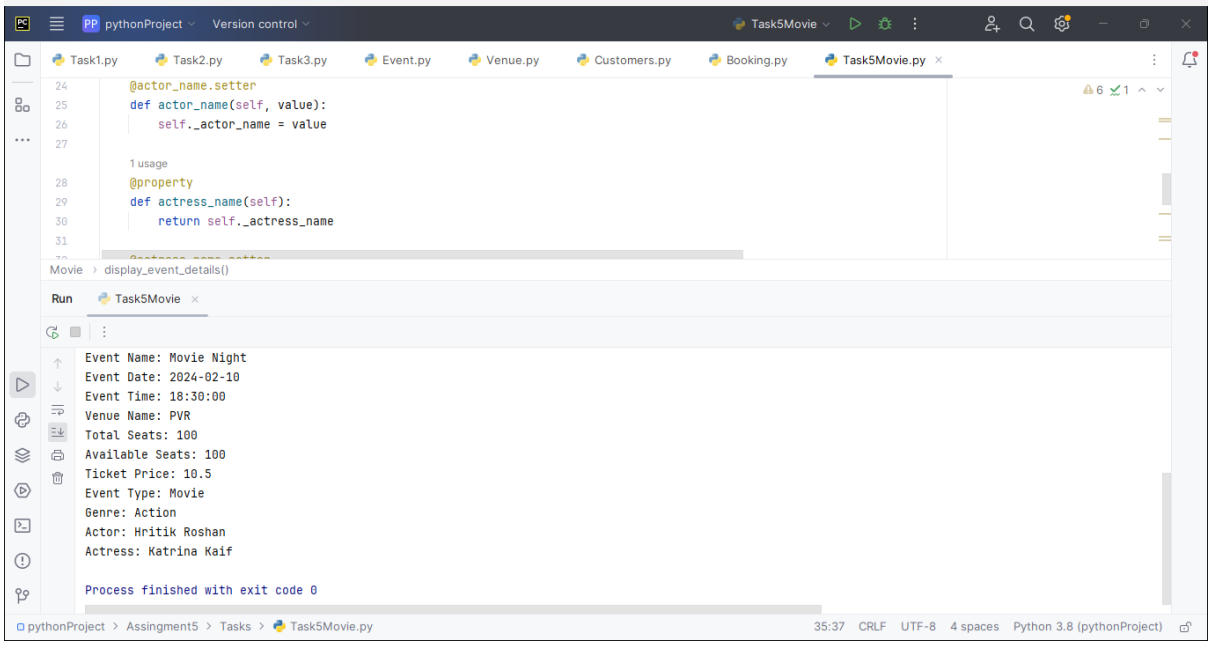


```python
from datetime import date, time
from Assignment5.Classes.Event import Event



3 usages
class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None, venue_name="", total_seats=0, ticket_price=0.0, event_type="", genre="", actor_n
        super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
        self._genre = genre
        self._actor_name = actor_name
        self._actress_name = actress_name
    1 usage
    @property
    def genre(self):
        return self._genre

    @genre.setter
    def genre(self, value):
        self._genre = value

    1 usage
    @property
    def actor_name(self):
        return self._actor_name
```



```python
    @actor_name.setter
    def actor_name(self, value):
        self._actor_name = value

    1 usage
    @property
    def actress_name(self):
        return self._actress_name

    @actress_name.setter
    def actress_name(self, value):
        self._actress_name = value
    3 usages (2 dynamic)
    def display_event_details(self):
        super().display_event_details()
        print(f"Genre: {self._genre}")
        print(f"Actor: {self._actor_name}")
        print(f"Actress: {self._actress_name}")


if __name__ == "__main__":
    movie = Movie( event_name: "Movie Night", date( year: 2024, month: 2, day: 10), time( hour: 18, minute: 30), venue_name: "PVR", total_seats: 100, ticket_price: 10.5
    movie.display_event_details()
```

Output:-



Create another subclass Concert that inherits from Event. Add the following attributes and methods:

o Attributes:

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

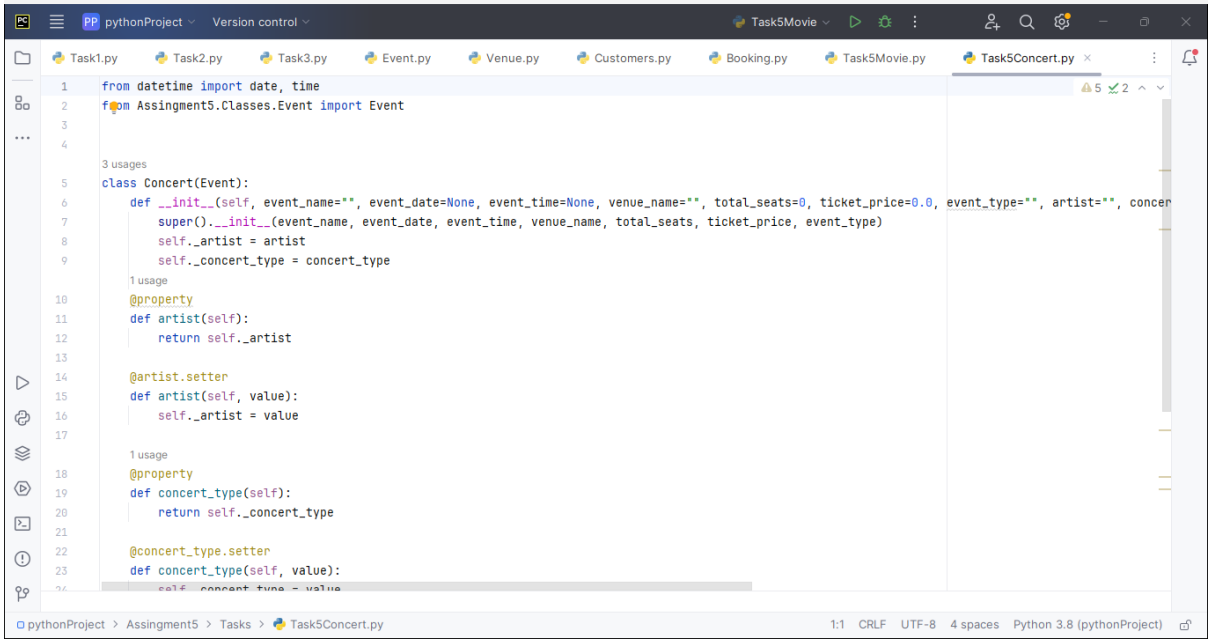
2. type: (Theatrical, Classical, Rock, Recital)

o Methods:

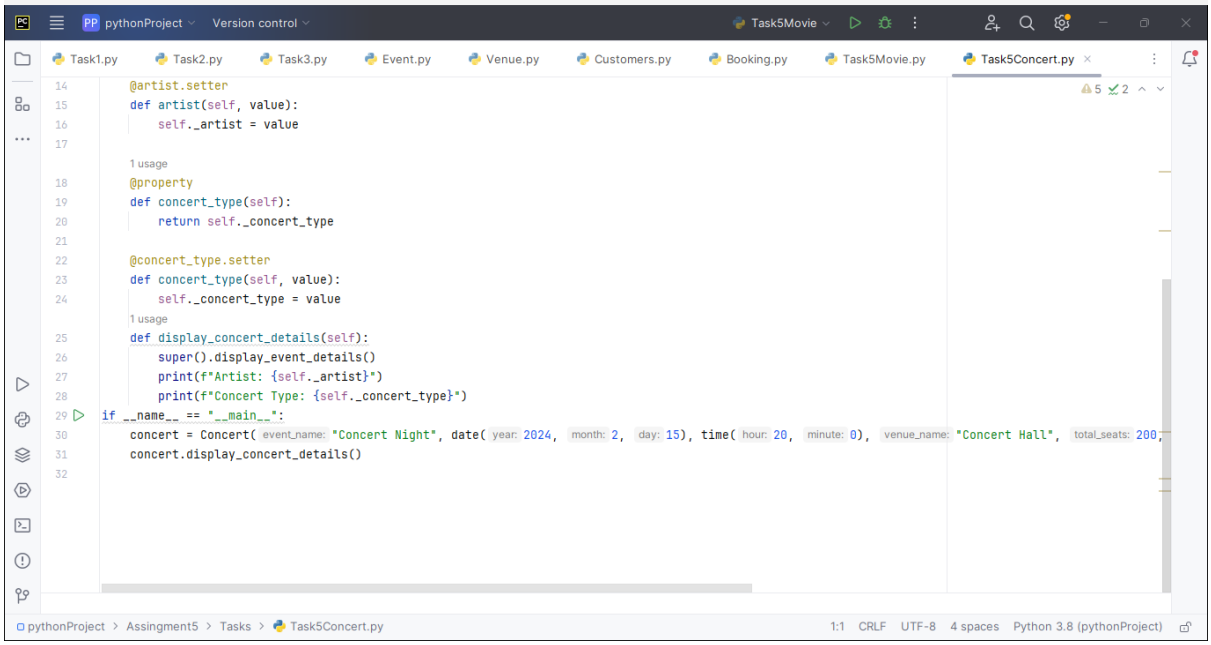
1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

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

CONCERT:-

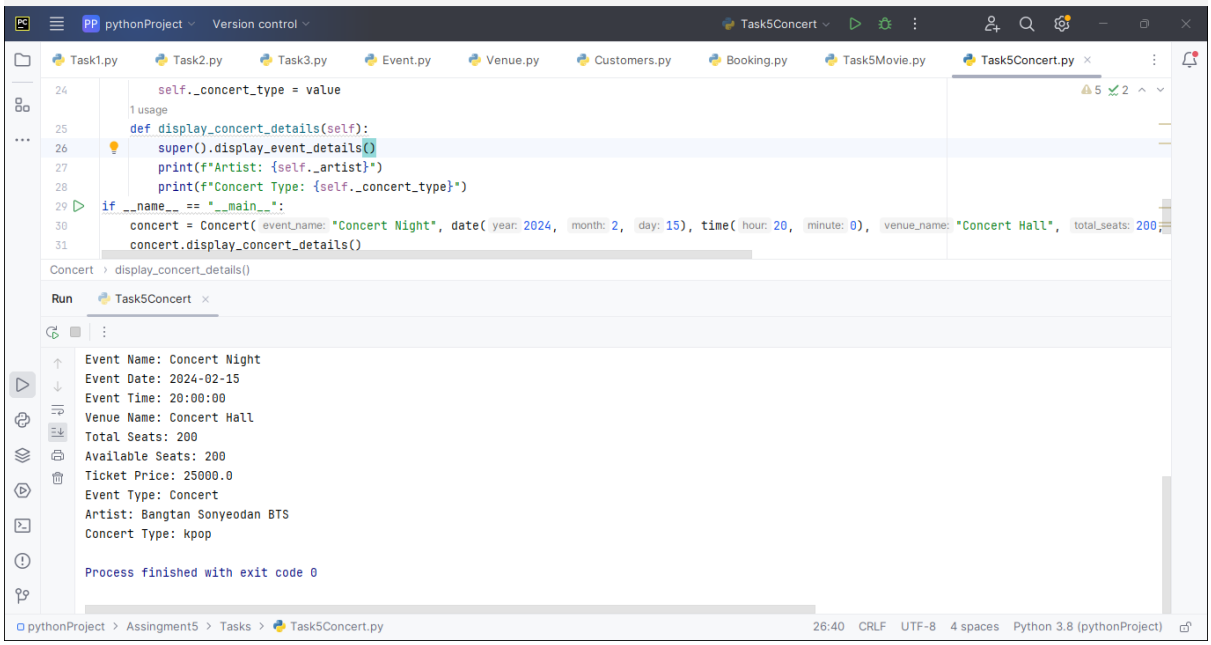


```
from datetime import date, time
from Assignment5.Classes.Event import Event


3 usages
class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None, venue_name="", total_seats=0, ticket_price=0.0, event_type="", artist="", concer
        super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
        self._artist = artist
        self._concert_type = concert_type
    1 usage
    @property
    def artist(self):
        return self._artist

    @artist.setter
    def artist(self, value):
        self._artist = value

    1 usage
    @property
    def concert_type(self):
        return self._concert_type

    @concert_type.setter
    def concert_type(self, value):
```



```
    @artist.setter
    def artist(self, value):
        self._artist = value

    1 usage
    @property
    def concert_type(self):
        return self._concert_type

    @concert_type.setter
    def concert_type(self, value):
        self._concert_type = value
    1 usage
    def display_concert_details(self):
        super().display_event_details()
        print(f"Artist: {self._artist}")
        print(f"Concert Type: {self._concert_type}")
if __name__ == "__main__":
    concert = Concert( event_name: "Concert Night", date( year: 2024, month: 2, day: 15), time( hour: 20, minute: 0), venue_name: "Concert Hall", total_seats: 200,
    concert.display_concert_details()
```

Output:



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

o Attributes:

1. sportName: Name of the game.

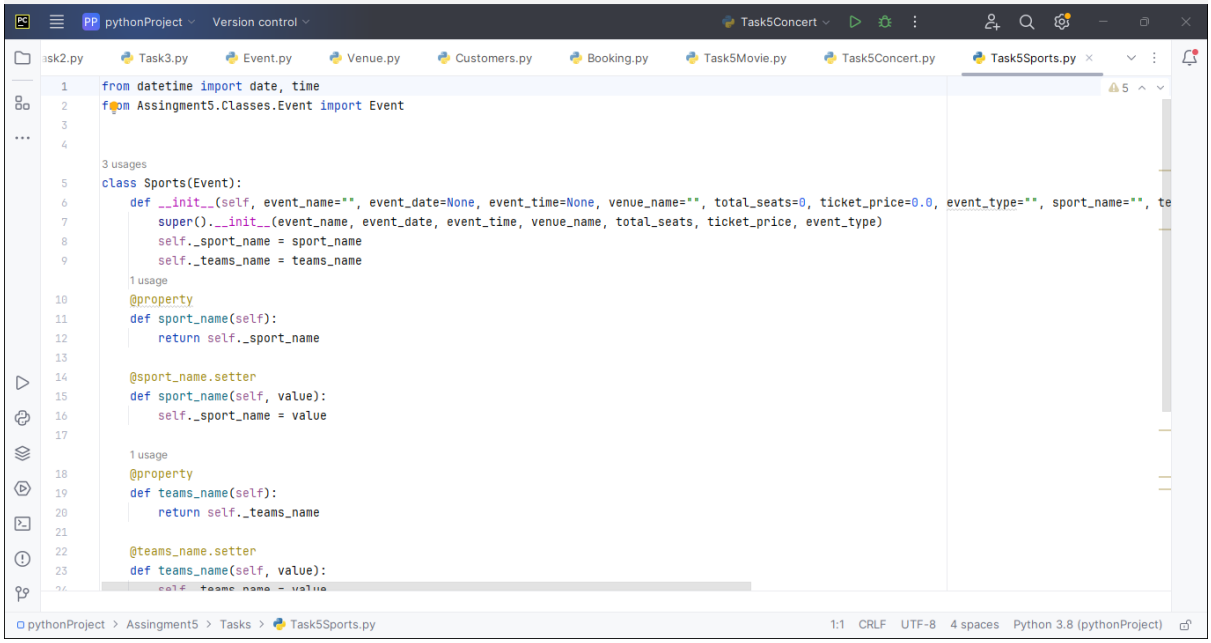
2. teamsName: (India vs Pakistan)

o Methods:

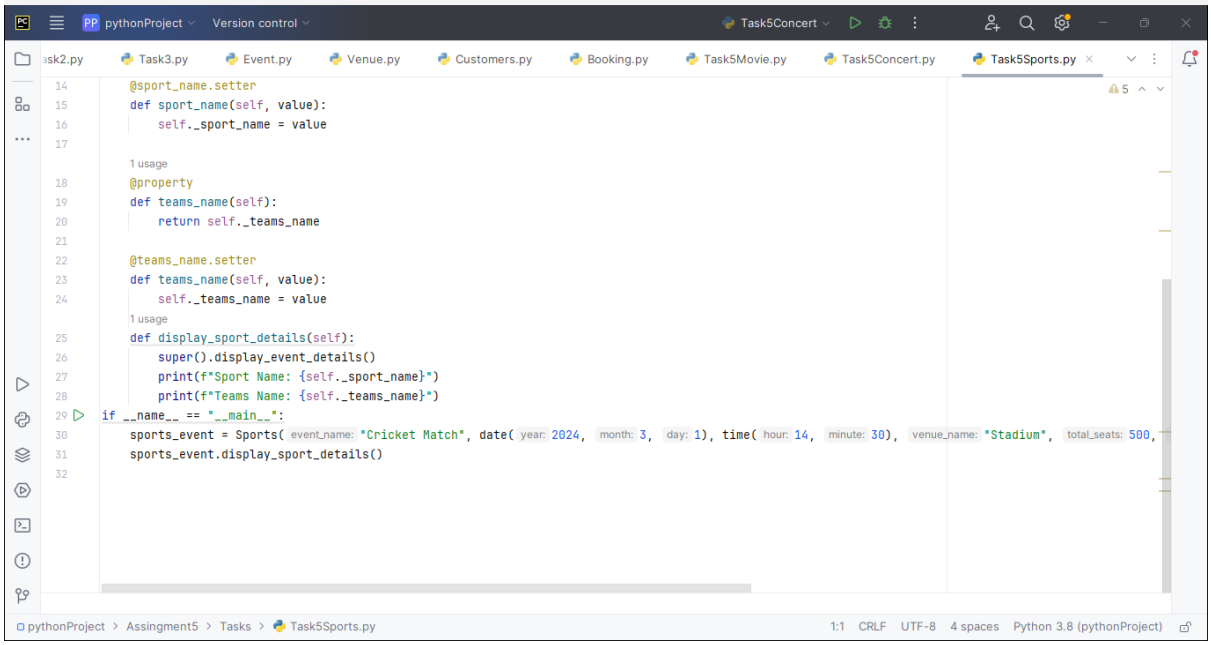
1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

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

SPORTS:-

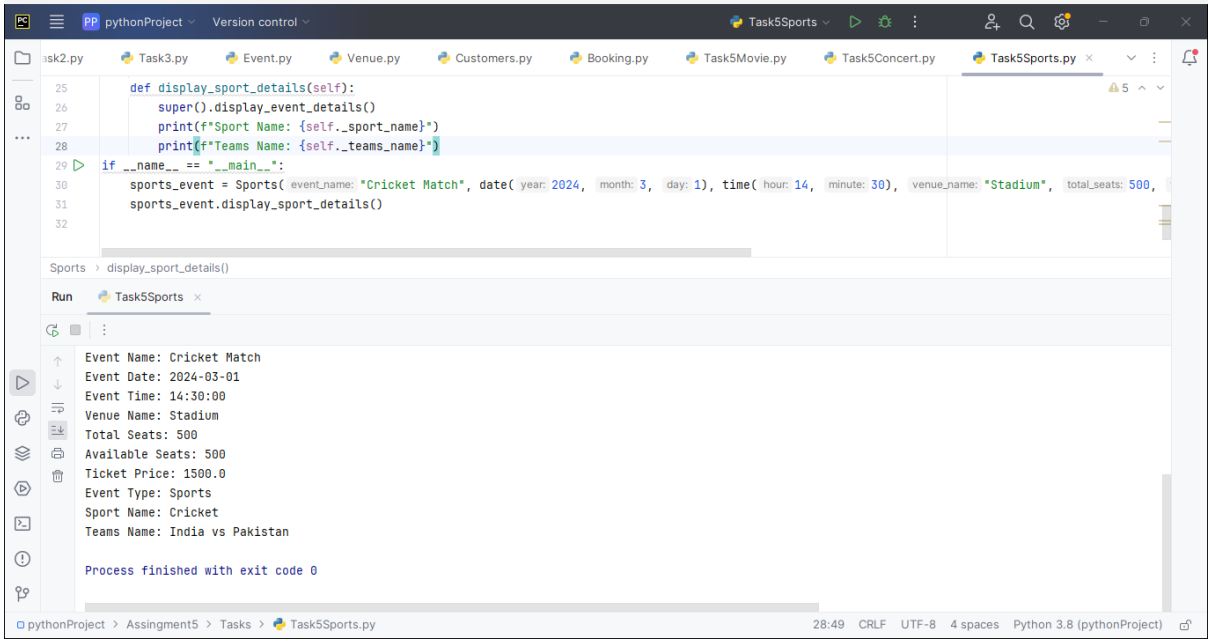


```python
from datetime import date, time
from Assingment5.Classes.Event import Event


3 usages
class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None, venue_name="", total_seats=0, ticket_price=0.0, event_type="", sport_name="", te
        super().__init__(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type)
        self._sport_name = sport_name
        self._teams_name = teams_name
    1 usage
    @property
    def sport_name(self):
        return self._sport_name

    @sport_name.setter
    def sport_name(self, value):
        self._sport_name = value

    1 usage
    @property
    def teams_name(self):
        return self._teams_name

    @teams_name.setter
    def teams_name(self, value):
```



```python
    @sport_name.setter
    def sport_name(self, value):
        self._sport_name = value

    1 usage
    @property
    def teams_name(self):
        return self._teams_name

    @teams_name.setter
    def teams_name(self, value):
        self._teams_name = value
    1 usage
    def display_sport_details(self):
        super().display_event_details()
        print(f"Sport Name: {self._sport_name}")
        print(f"Teams Name: {self._teams_name}")
if __name__ == "__main__":
    sports_event = Sports( event_name: "Cricket Match", date( year: 2024, month: 3, day: 1), time( hour: 14, minute: 30), venue_name: "Stadium", total_seats: 500,
    sports_event.display_sport_details()
```

Output:-



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.

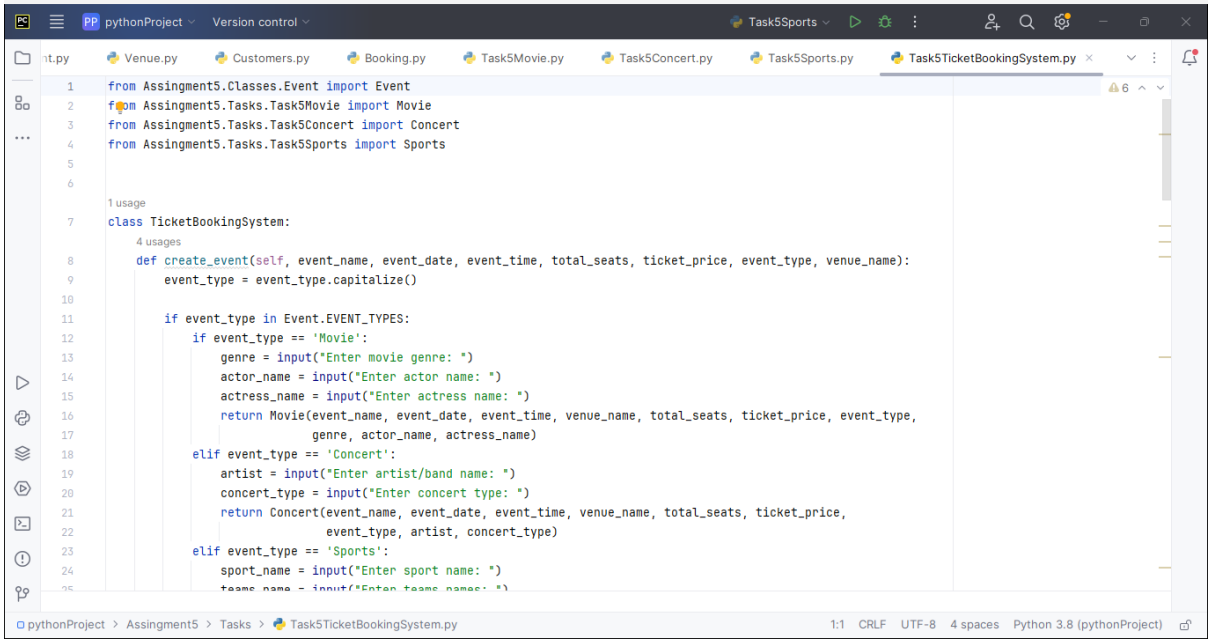
o main(): simulates the ticket booking system

1. User can book tickets and view the event details as per their choice in menu (movies, sports, concerts).

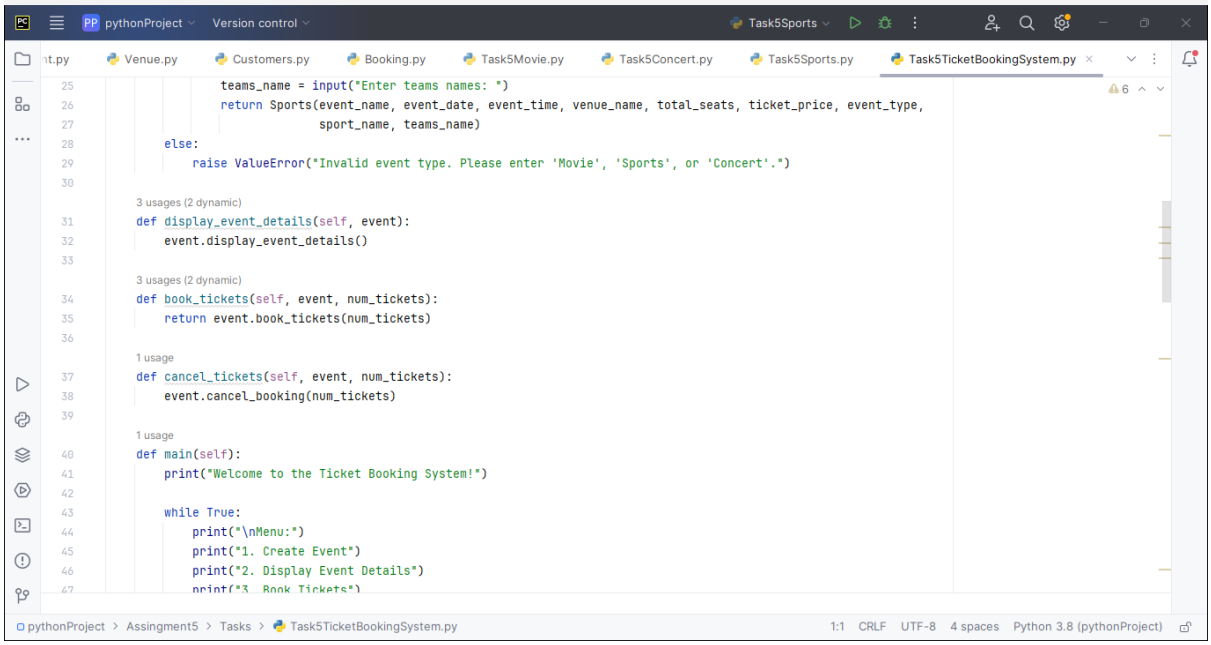
2. Display event details using the display_event_details() method without knowing the specific event type (demonstrate polymorphism).

3. Make bookings using the book_tickets() and cancel tickets cancel_tickets() method.

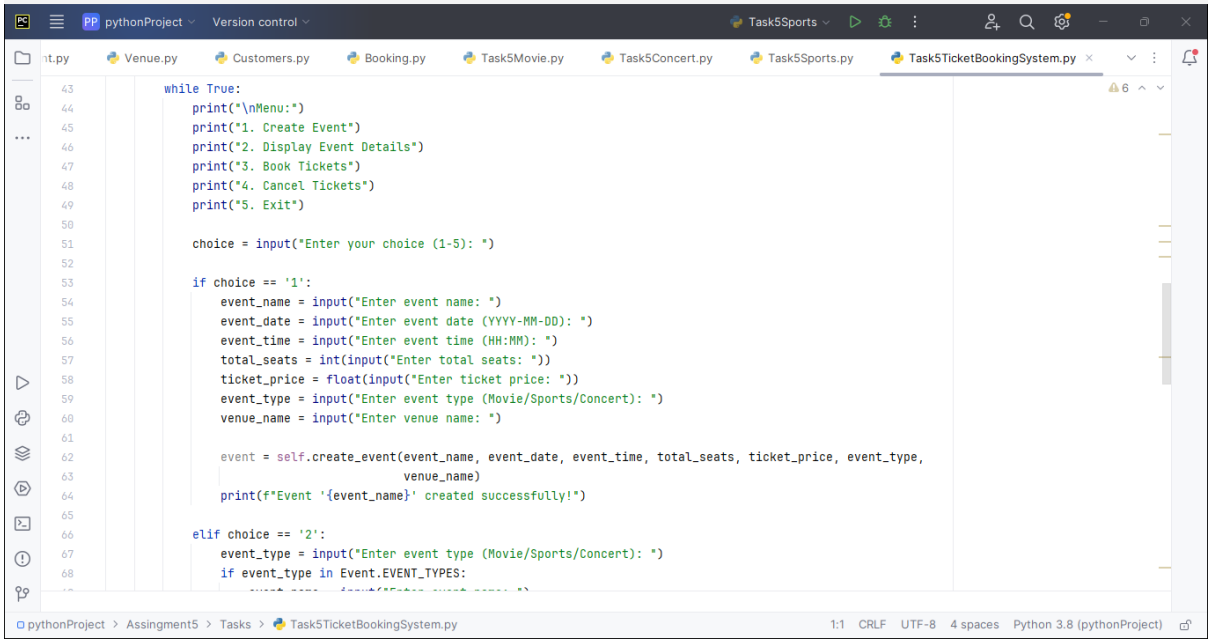
TicketBookingSystem:-



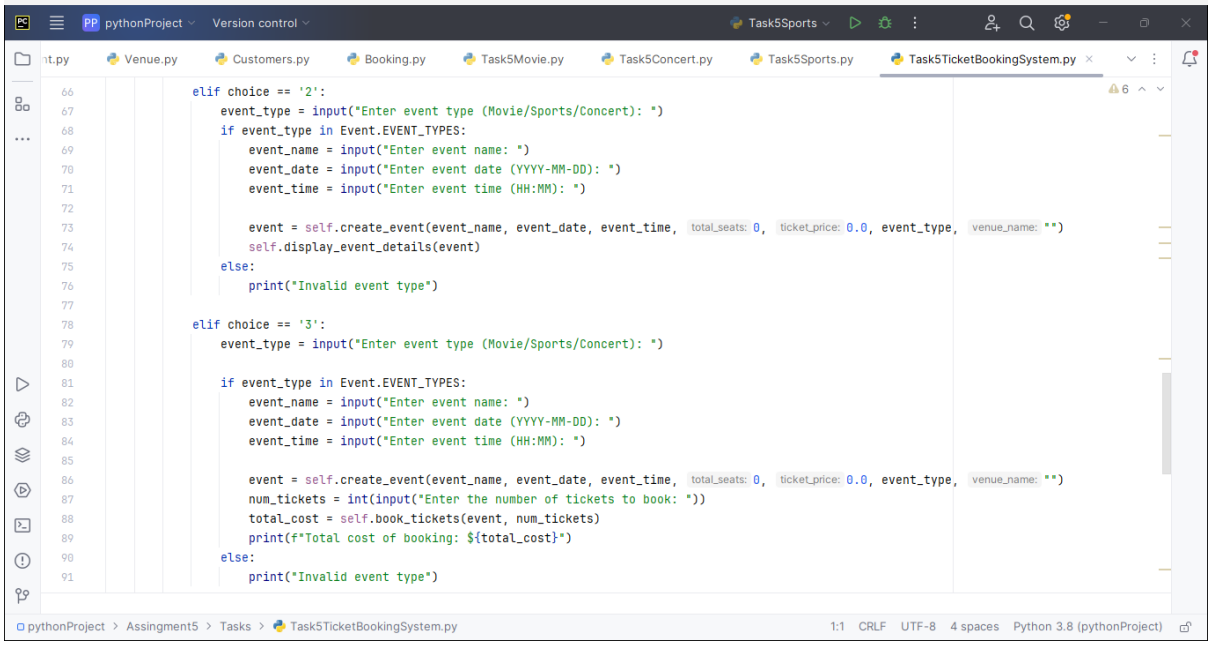
```
from Assingment5.Classes.Event import Event
from Assingment5.Tasks.Task5Movie import Movie
from Assingment5.Tasks.Task5Concert import Concert
from Assingment5.Tasks.Task5Sports import Sports


1 usage
class TicketBookingSystem:
    4 usages
    def create_event(self, event_name, event_date, event_time, total_seats, ticket_price, event_type, venue_name):
        event_type = event_type.capitalize()

        if event_type in Event.EVENT_TYPES:
            if event_type == 'Movie':
                genre = input("Enter movie genre: ")
                actor_name = input("Enter actor name: ")
                actress_name = input("Enter actress name: ")
                return Movie(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
                             genre, actor_name, actress_name)
            elif event_type == 'Concert':
                artist = input("Enter artist/band name: ")
                concert_type = input("Enter concert type: ")
                return Concert(event_name, event_date, event_time, venue_name, total_seats, ticket_price,
                               event_type, artist, concert_type)
            elif event_type == 'Sports':
                sport_name = input("Enter sport name: ")
```



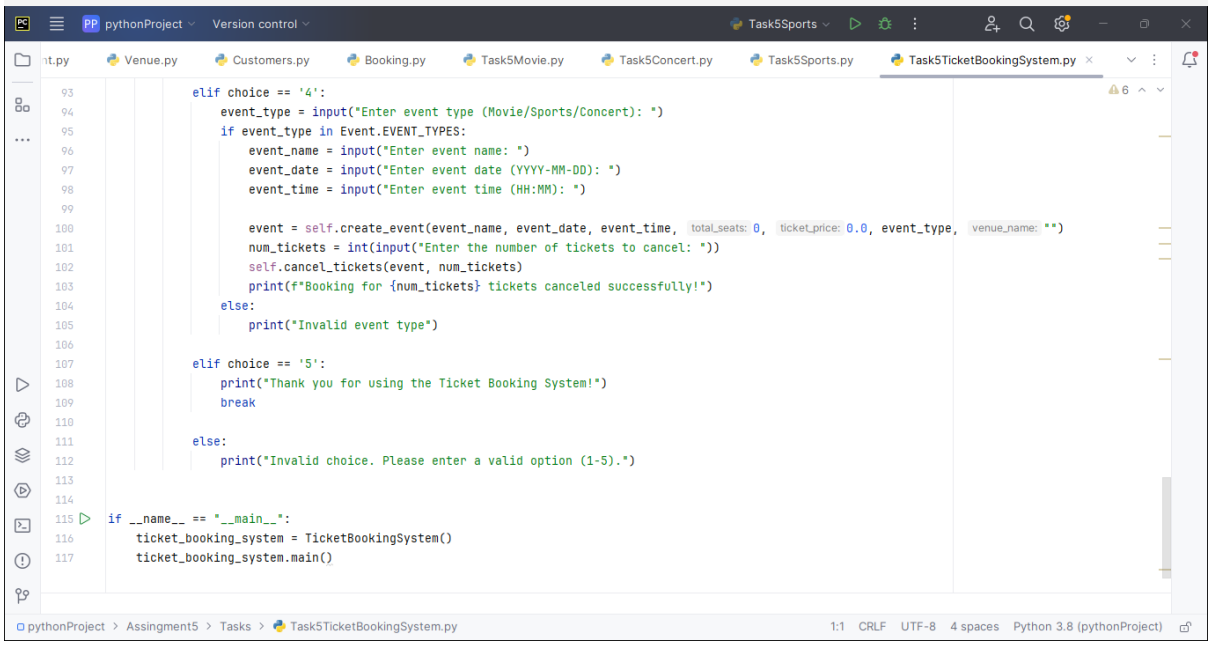
```
                teams_name = input("Enter teams names: ")
                return Sports(event_name, event_date, event_time, venue_name, total_seats, ticket_price, event_type,
                              sport_name, teams_name)
        else:
            raise ValueError("Invalid event type. Please enter 'Movie', 'Sports', or 'Concert'.")

    3 usages (2 dynamic)
    def display_event_details(self, event):
        event.display_event_details()

    3 usages (2 dynamic)
    def book_tickets(self, event, num_tickets):
        return event.book_tickets(num_tickets)

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

    1 usage
    def main(self):
        print("Welcome to the Ticket Booking System!")

        while True:
            print("\nMenu:")
            print("1. Create Event")
            print("2. Display Event Details")
```



```
while True:
    print("\nMenu:")
    print("1. Create Event")
    print("2. Display Event Details")
    print("3. Book Tickets")
    print("4. Cancel Tickets")
    print("5. Exit")

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

    if choice == '1':
        event_name = input("Enter event name: ")
        event_date = input("Enter event date (YYYY-MM-DD): ")
        event_time = input("Enter event time (HH:MM): ")
        total_seats = int(input("Enter total seats: "))
        ticket_price = float(input("Enter ticket price: "))
        event_type = input("Enter event type (Movie/Sports/Concert): ")
        venue_name = input("Enter venue name: ")

        event = self.create_event(event_name, event_date, event_time, total_seats, ticket_price, event_type,
                                  venue_name)
        print(f"Event '{event_name}' created successfully!")

    elif choice == '2':
        event_type = input("Enter event type (Movie/Sports/Concert): ")
        if event_type in Event.EVENT_TYPES:
```

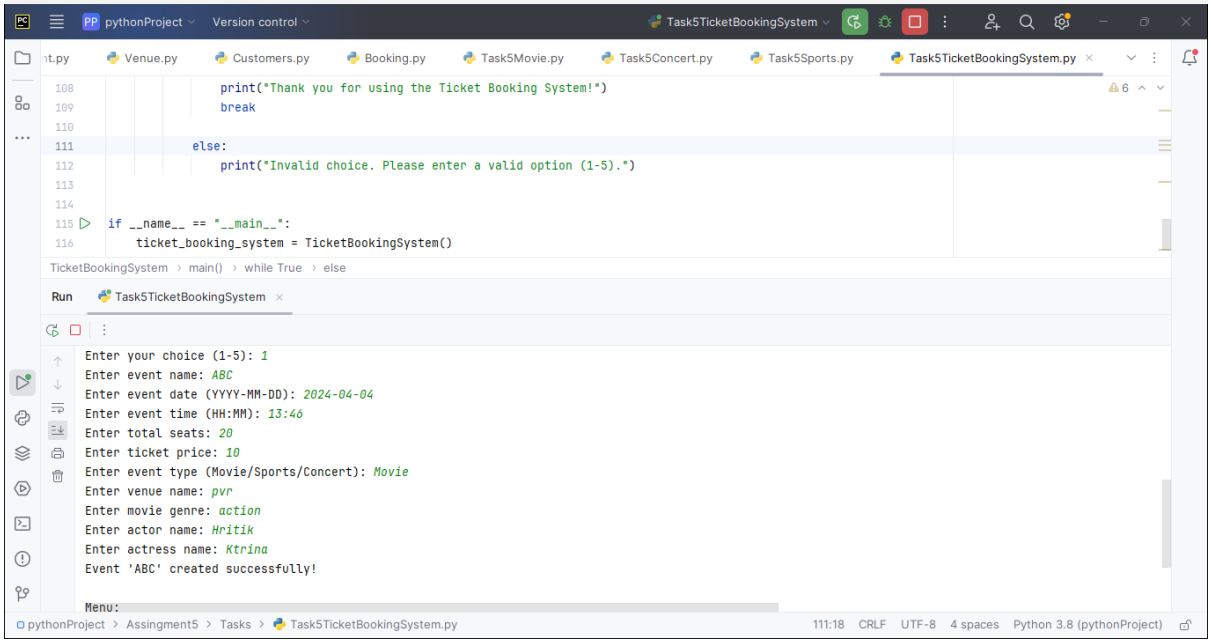


```
    elif choice == '2':
        event_type = input("Enter event type (Movie/Sports/Concert): ")
        if event_type in Event.EVENT_TYPES:
            event_name = input("Enter event name: ")
            event_date = input("Enter event date (YYYY-MM-DD): ")
            event_time = input("Enter event time (HH:MM): ")

            event = self.create_event(event_name, event_date, event_time, total_seats: 0, ticket_price: 0.0, event_type, venue_name: "")
            self.display_event_details(event)
        else:
            print("Invalid event type")

    elif choice == '3':
        event_type = input("Enter event type (Movie/Sports/Concert): ")

        if event_type in Event.EVENT_TYPES:
            event_name = input("Enter event name: ")
            event_date = input("Enter event date (YYYY-MM-DD): ")
            event_time = input("Enter event time (HH:MM): ")

            event = self.create_event(event_name, event_date, event_time, total_seats: 0, ticket_price: 0.0, event_type, venue_name: "")
            num_tickets = int(input("Enter the number of tickets to book: "))
            total_cost = self.book_tickets(event, num_tickets)
            print(f"Total cost of booking: ${total_cost}")
        else:
            print("Invalid event type")
```



```
            elif choice == '4':
                event_type = input("Enter event type (Movie/Sports/Concert): ")
                if event_type in Event.EVENT_TYPES:
                    event_name = input("Enter event name: ")
                    event_date = input("Enter event date (YYYY-MM-DD): ")
                    event_time = input("Enter event time (HH:MM): ")

                    event = self.create_event(event_name, event_date, event_time, total_seats: 0, ticket_price: 0.0, event_type, venue_name: "")
                    num_tickets = int(input("Enter the number of tickets to cancel: "))
                    self.cancel_tickets(event, num_tickets)
                    print(f"Booking for {num_tickets} tickets canceled successfully!")
                else:
                    print("Invalid event type")

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

            else:
                print("Invalid choice. Please enter a valid option (1-5).")


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

OUTPUT:-



```
            print("Thank you for using the Ticket Booking System!")
            break

        else:
            print("Invalid choice. Please enter a valid option (1-5).")


if __name__ == "__main__":
    ticket_booking_system = TicketBookingSystem()
```

```
Enter your choice (1-5): 1
Enter event name: ABC
Enter event date (YYYY-MM-DD): 2024-04-04
Enter event time (HH:MM): 13:46
Enter total seats: 20
Enter ticket price: 10
Enter event type (Movie/Sports/Concert): Movie
Enter venue name: pvr
Enter movie genre: action
Enter actor name: Hritik
Enter actress name: Ktrina
Event 'ABC' created successfully!

Menu:
```

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:

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.

ANS:-USE ABSTRACTION IN TASK 4 EVENT CLASS AND TASK 5 SUBCLASS MOVIE,CONCERT AND SPORT.

3. BookingSystem Abstraction:

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

4. Concrete TicketBookingSystem Class:

Create a concrete class TicketBookingSystem that inherits from BookingSystem:

TicketBookingSystem: Implement the abstract methods to create events, book tickets, and retrieve available seats. Maintain an array of events in this class.

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

ANS:-USE ABSTRACTION IN TASK 5 TICKETBOOKINGSYSTEM.

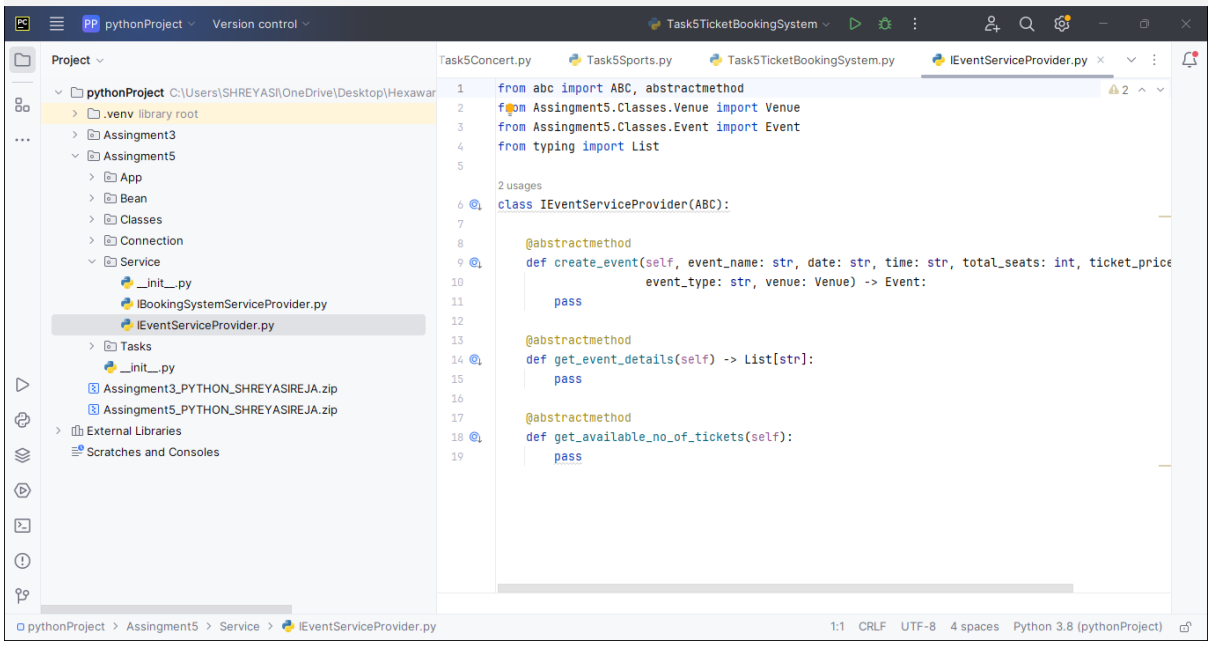
Task 8: Interface/abstract class, and Single Inheritance, static variable

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.



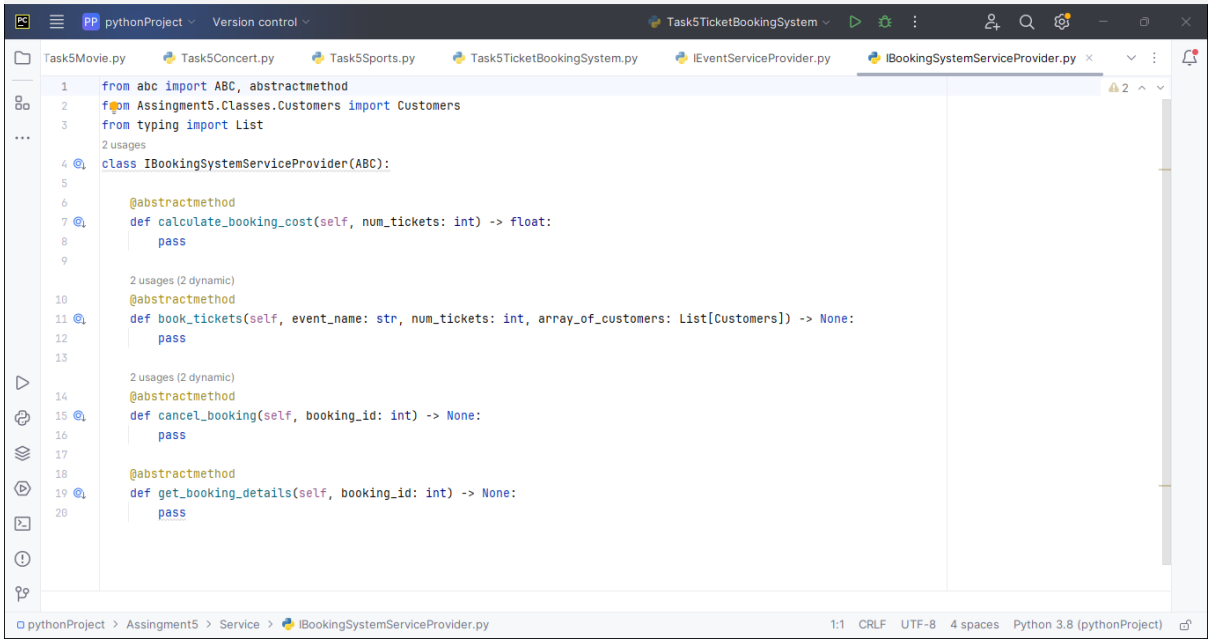
6. Create interface/abstract class IBookingSystemServiceProvider with following methods:

calculate_booking_cost(num_tickets): Calculate and set the total cost of the booking.

book_tickets(eventname:str, num_tickets, arrayOfCustomer): Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class.

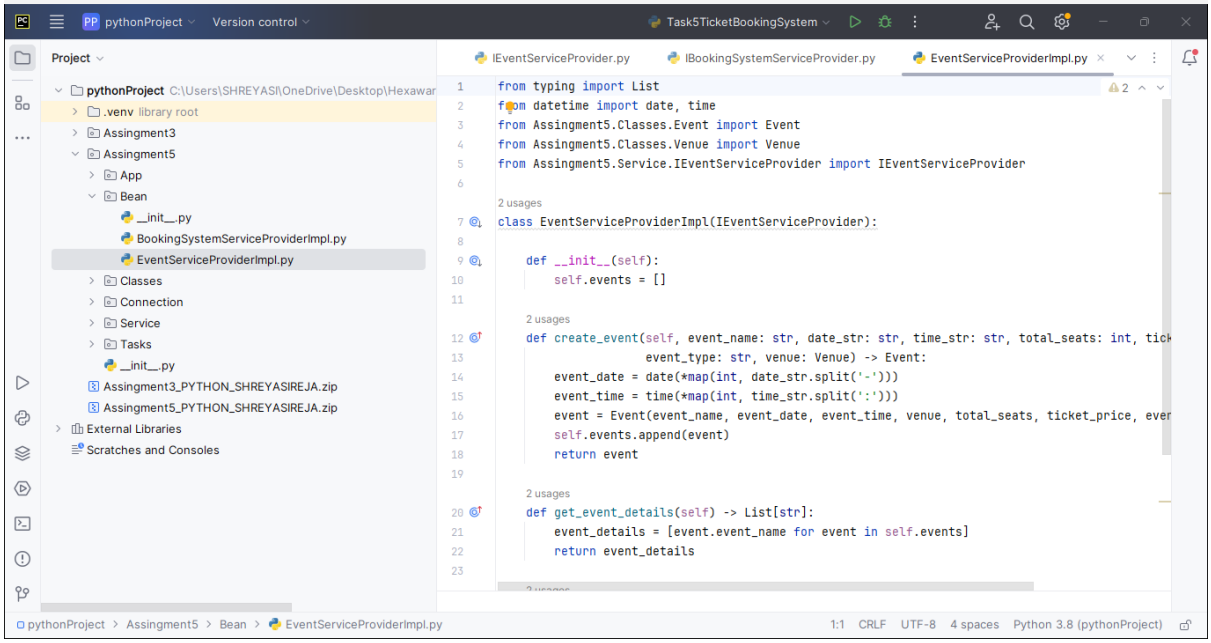
cancel_booking(booking_id): Cancel the booking and update the available seats.

get_booking_details(booking_id):get the booking details.

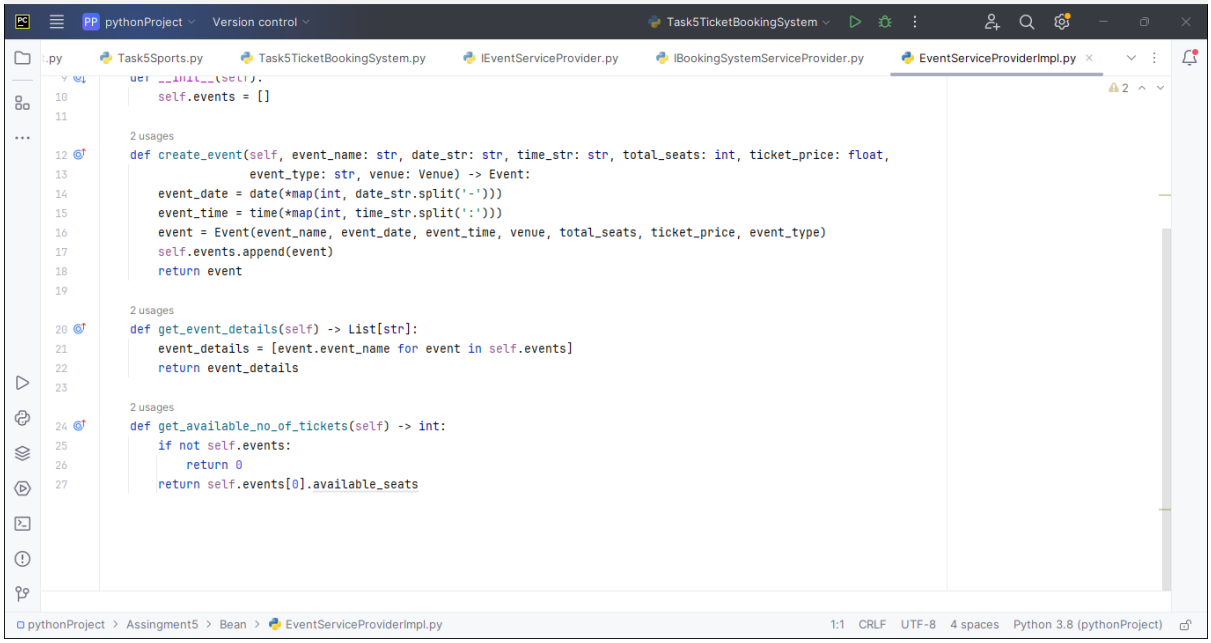


```python
from abc import ABC, abstractmethod
from Assingment5.Classes.Customers import Customers
from typing import List

class IBookingSystemServiceProvider(ABC):

    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int) -> float:
        pass

    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, array_of_customers: List[Customers]) -> None:
        pass

    @abstractmethod
    def cancel_booking(self, booking_id: int) -> None:
        pass

    @abstractmethod
    def get_booking_details(self, booking_id: int) -> None:
        pass
```

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



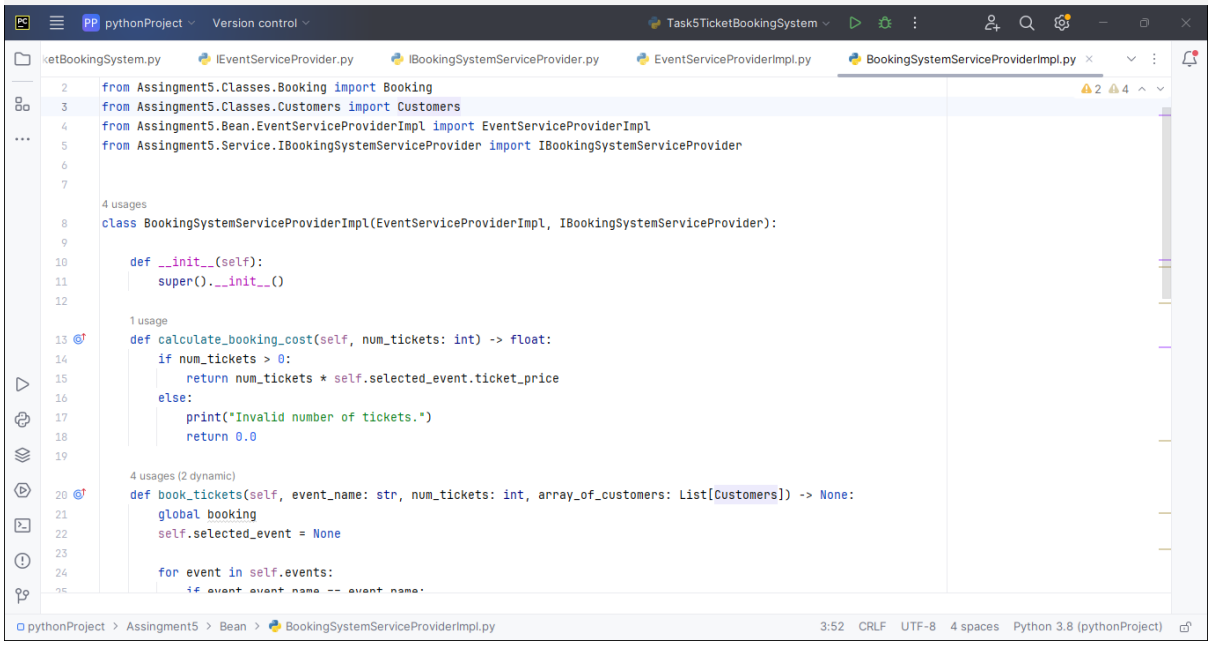
```python
from typing import List
from datetime import date, time
from Assingment5.Classes.Event import Event
from Assingment5.Classes.Venue import Venue
from Assingment5.Service.IEventServiceProvider import IEventServiceProvider

class EventServiceProviderImpl(IEventServiceProvider):

    def __init__(self):
        self.events = []

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

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

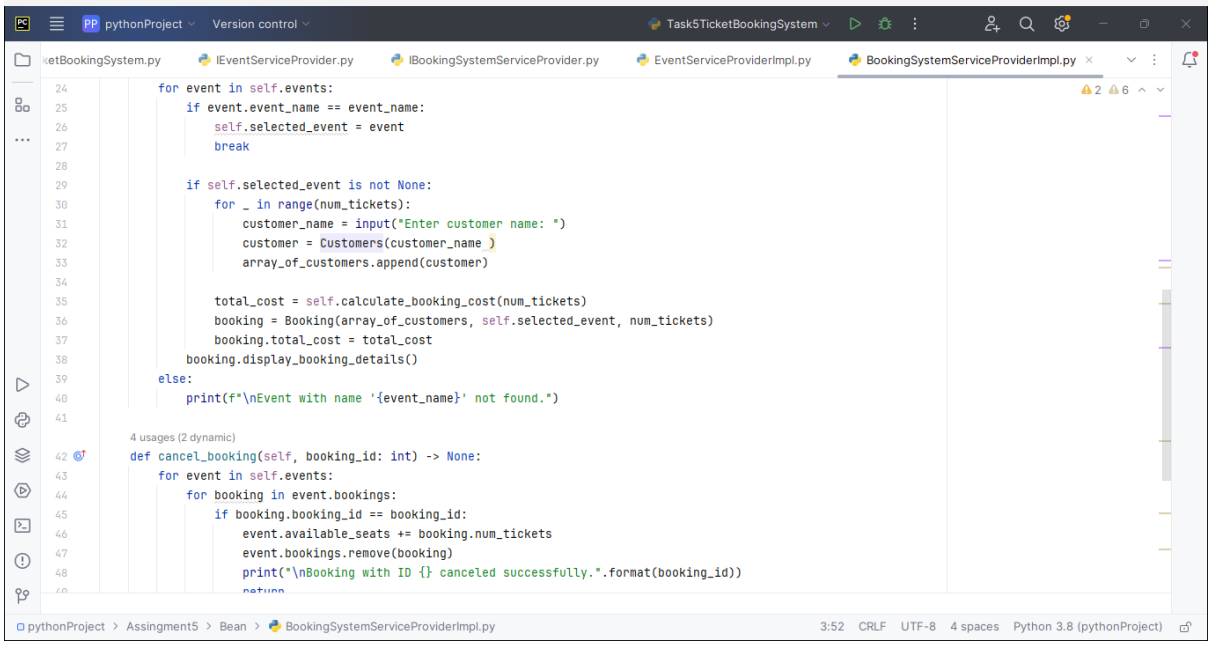


8. Create BookingSystemServiceProviderImpl class which implements IBookingSystemServiceProvider provide all implementation methods and inherits EventServiceProviderImpl class with following attributes.

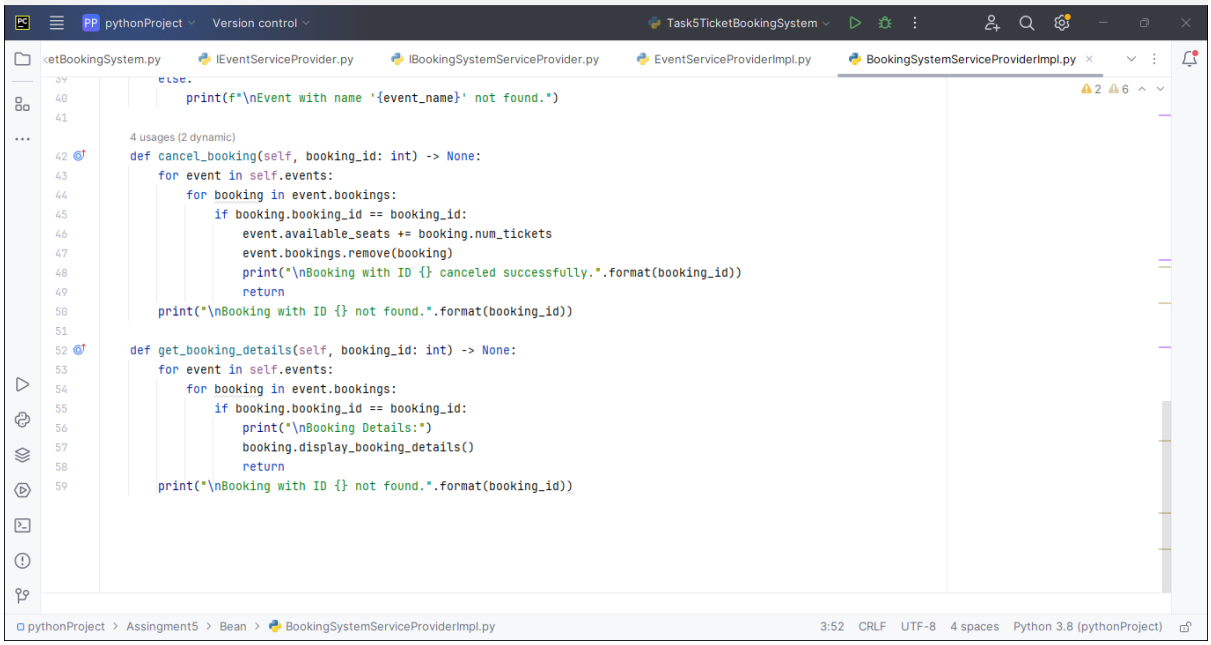
Attributes

o array of events





```
        for event in self.events:
            if event.event_name == event_name:
                self.selected_event = event
                break

            if self.selected_event is not None:
                for _ in range(num_tickets):
                    customer_name = input("Enter customer name: ")
                    customer = Customers(customer_name )
                    array_of_customers.append(customer)

                total_cost = self.calculate_booking_cost(num_tickets)
                booking = Booking(array_of_customers, self.selected_event, num_tickets)
                booking.total_cost = total_cost
            booking.display_booking_details()
        else:
            print(f"\nEvent with name '{event_name}' not found.")

    4 usages (2 dynamic)
    def cancel_booking(self, booking_id: int) -> None:
        for event in self.events:
            for booking in event.bookings:
                if booking.booking_id == booking_id:
                    event.available_seats += booking.num_tickets
                    event.bookings.remove(booking)
                    print("\nBooking with ID {} canceled successfully.".format(booking_id))
```

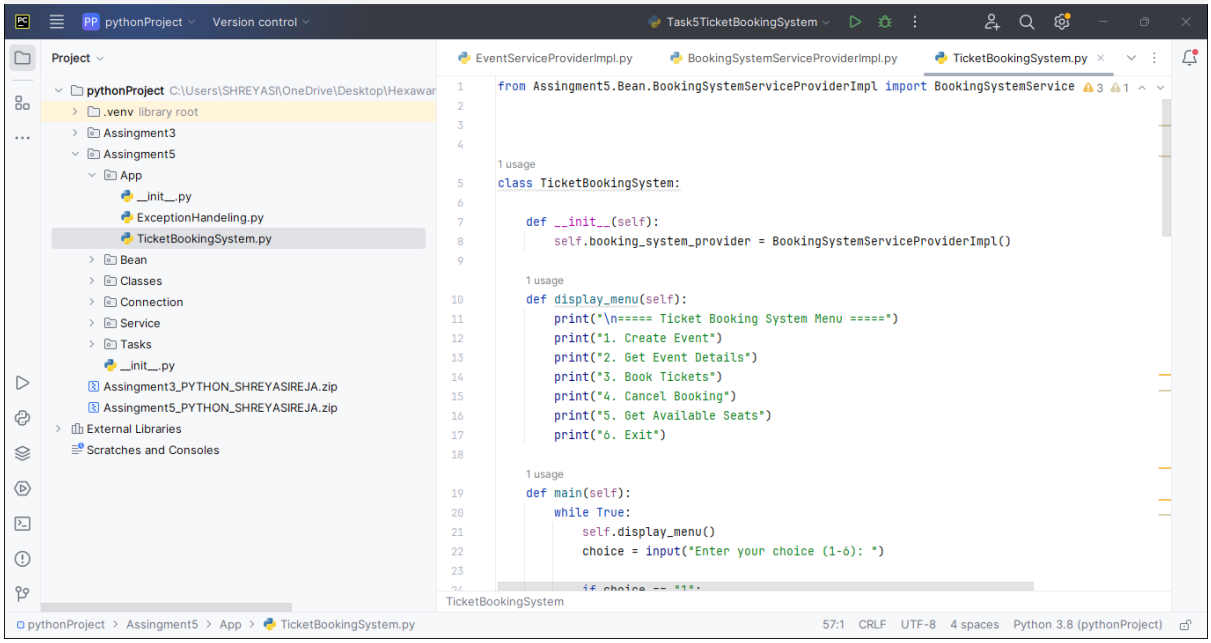


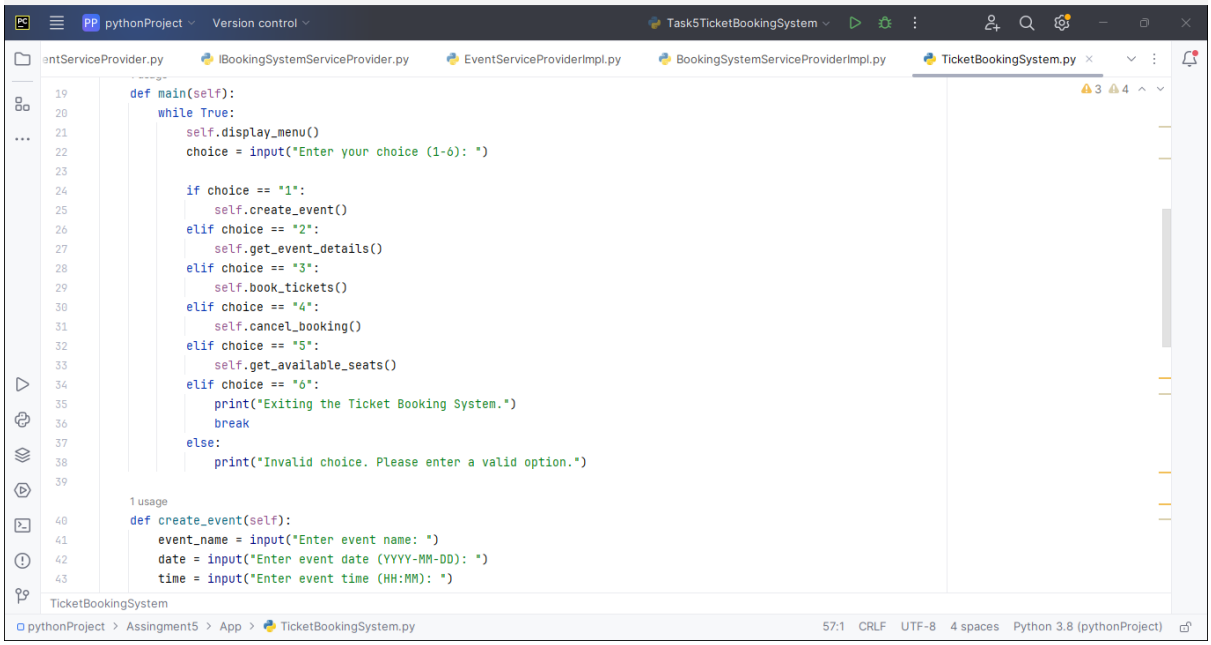
```
        else:
            print(f"\nEvent with name '{event_name}' not found.")

    4 usages (2 dynamic)
    def cancel_booking(self, booking_id: int) -> None:
        for event in self.events:
            for booking in event.bookings:
                if booking.booking_id == booking_id:
                    event.available_seats += booking.num_tickets
                    event.bookings.remove(booking)
                    print("\nBooking with ID {} canceled successfully.".format(booking_id))
                    return
        print("\nBooking with ID {} not found.".format(booking_id))

    def get_booking_details(self, booking_id: int) -> None:
        for event in self.events:
            for booking in event.bookings:
                if booking.booking_id == booking_id:
                    print("\nBooking Details:")
                    booking.display_booking_details()
                    return
        print("\nBooking with ID {} not found.".format(booking_id))
```

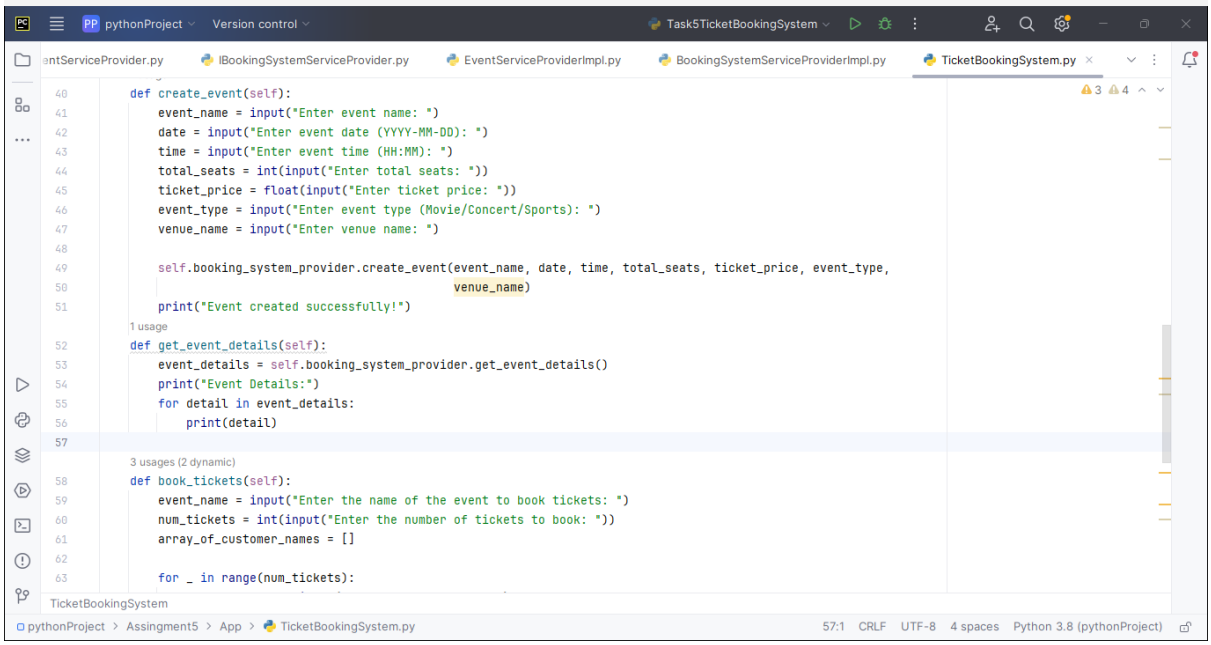
9. Create TicketBookingSystem class and perform following operations:

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."

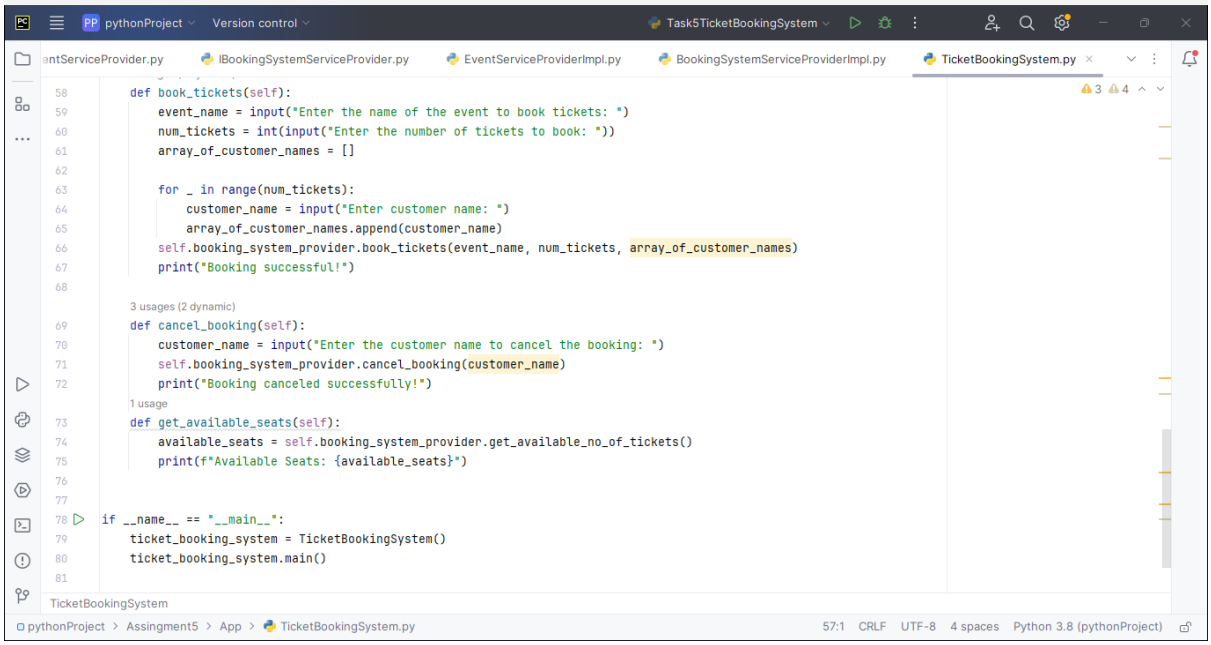
10. Place the interface/abstract class in service package and interface/abstract class implementation class, all concrete class in bean package and TicketBookingSystem class in app package.


pythonProject
Version control
Task5TicketBookingSystem
Project
pythonProject C:\Users\SHREYASI\OneDrive\Desktop\Hexawar
.venv library root
Assingment3
Assingment5
App
__init__.py
ExceptionHandeling.py
TicketBookingSystem.py
Bean
Classes
Connection
Service
Tasks
__init__.py
Assingment3_PYTHON_SHREYASIREJA.zip
Assingment5_PYTHON_SHREYASIREJA.zip
External Libraries
Scratches and Consoles
EventServiceProviderImpl.py
BookingSystemServiceProviderImpl.py
TicketBookingSystem.py
from Assingment5.Bean.BookingSystemServiceProviderImpl import BookingSystemService
1 usage
class TicketBookingSystem:
def __init__(self):
self.booking_system_provider = BookingSystemServiceProviderImpl()
1 usage
def display_menu(self):
print("\n===== Ticket Booking System Menu =====")
print("1. Create Event")
print("2. Get Event Details")
print("3. Book Tickets")
print("4. Cancel Booking")
print("5. Get Available Seats")
print("6. Exit")
1 usage
def main(self):
while True:
self.display_menu()
choice = input("Enter your choice (1-6): ")
TicketBookingSystem
pythonProject > Assingment5 > App > TicketBookingSystem.py
57:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)


pythonProject
Version control
Task5TicketBookingSystem
entServiceProvider.py
IBookingSystemServiceProvider.py
EventServiceProviderImpl.py
BookingSystemServiceProviderImpl.py
TicketBookingSystem.py
def main(self):
while True:
self.display_menu()
choice = input("Enter your choice (1-6): ")
if choice == "1":
self.create_event()
elif choice == "2":
self.get_event_details()
elif choice == "3":
self.book_tickets()
elif choice == "4":
self.cancel_booking()
elif choice == "5":
self.get_available_seats()
elif choice == "6":
print("Exiting the Ticket Booking System.")
break
else:
print("Invalid choice. Please enter a valid option.")
1 usage
def create_event(self):
event_name = input("Enter event name: ")
date = input("Enter event date (YYYY-MM-DD): ")
time = input("Enter event time (HH:MM): ")
TicketBookingSystem
pythonProject > Assingment5 > App > TicketBookingSystem.py
57:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

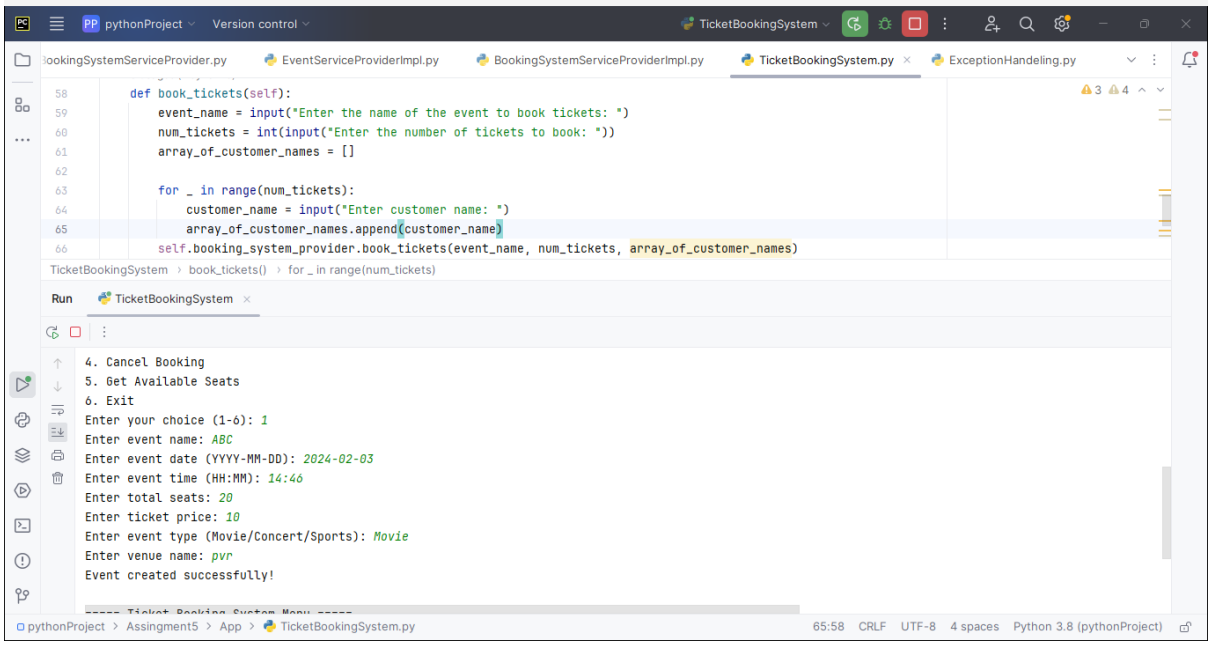


```
    def create_event(self):
        event_name = input("Enter event name: ")
        date = input("Enter event date (YYYY-MM-DD): ")
        time = input("Enter event time (HH:MM): ")
        total_seats = int(input("Enter total seats: "))
        ticket_price = float(input("Enter ticket price: "))
        event_type = input("Enter event type (Movie/Concert/Sports): ")
        venue_name = input("Enter venue name: ")

        self.booking_system_provider.create_event(event_name, date, time, total_seats, ticket_price, event_type,
                                                  venue_name)
        print("Event created successfully!")
    1 usage
    def get_event_details(self):
        event_details = self.booking_system_provider.get_event_details()
        print("Event Details:")
        for detail in event_details:
            print(detail)

    3 usages (2 dynamic)
    def book_tickets(self):
        event_name = input("Enter the name of the event to book tickets: ")
        num_tickets = int(input("Enter the number of tickets to book: "))
        array_of_customer_names = []

        for _ in range(num_tickets):
```



```
    def book_tickets(self):
        event_name = input("Enter the name of the event to book tickets: ")
        num_tickets = int(input("Enter the number of tickets to book: "))
        array_of_customer_names = []

        for _ in range(num_tickets):
            customer_name = input("Enter customer name: ")
            array_of_customer_names.append(customer_name)
        self.booking_system_provider.book_tickets(event_name, num_tickets, array_of_customer_names)
        print("Booking successful!")

    3 usages (2 dynamic)
    def cancel_booking(self):
        customer_name = input("Enter the customer name to cancel the booking: ")
        self.booking_system_provider.cancel_booking(customer_name)
        print("Booking canceled successfully!")
    1 usage
    def get_available_seats(self):
        available_seats = self.booking_system_provider.get_available_no_of_tickets()
        print(f"Available Seats: {available_seats}")


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

Output:-



Task 9: Exception Handling

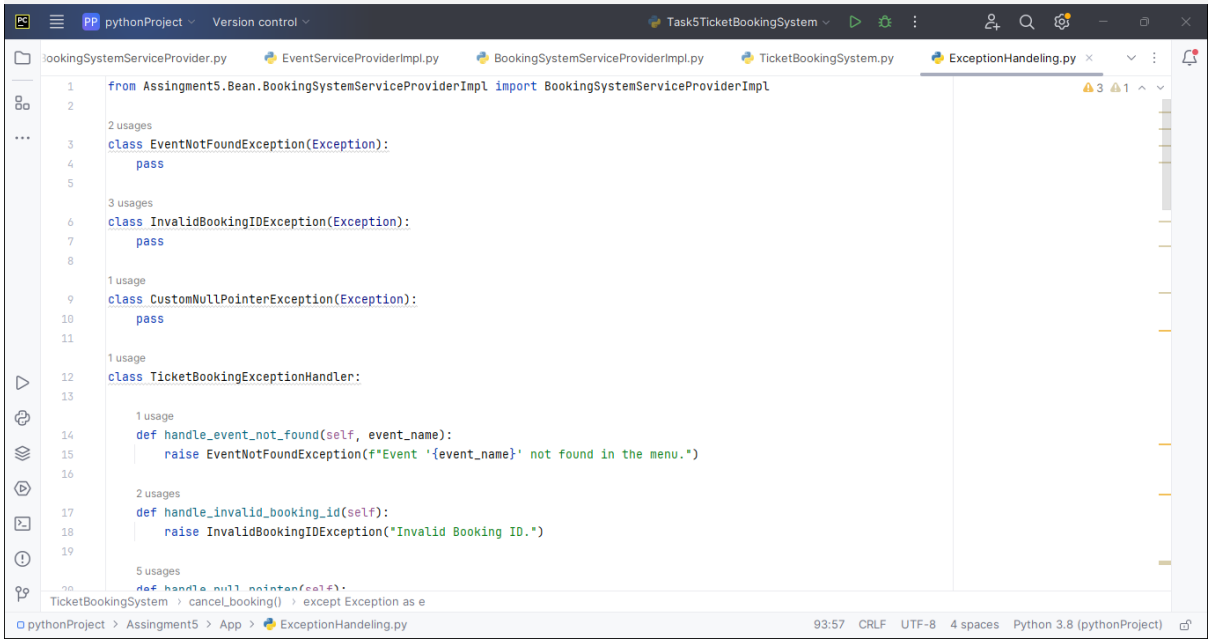
throw the exception whenever needed and Handle in main method,

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

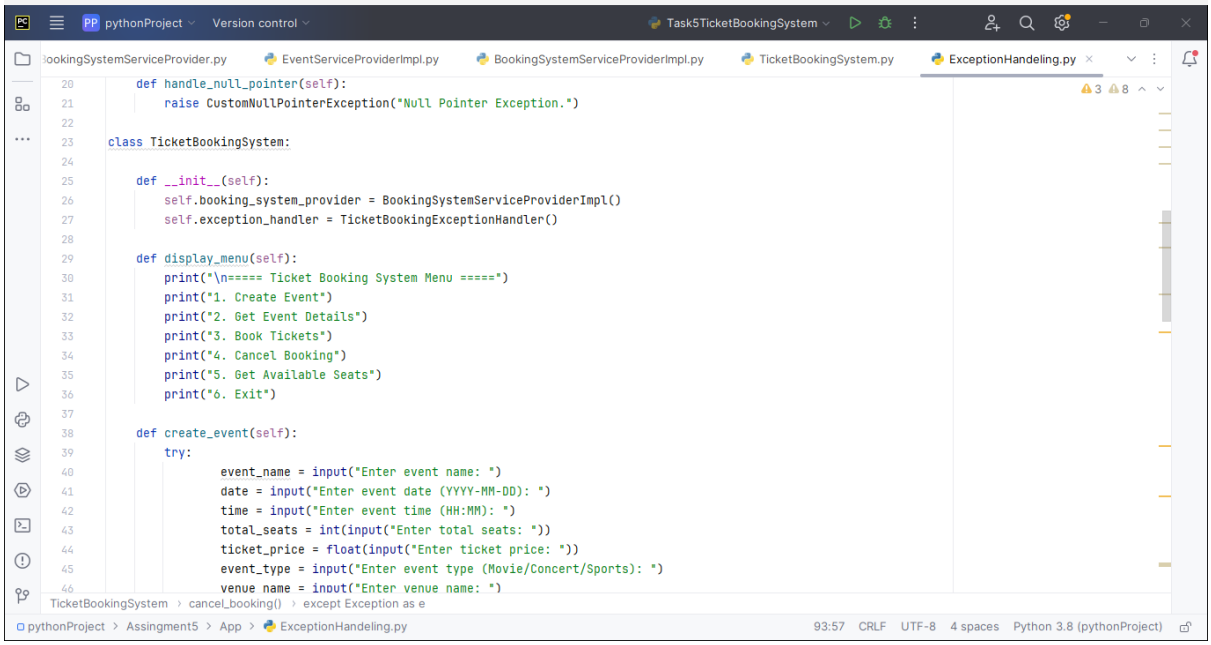
2. InvalidBookingIDException throw this exception when user entered the invalid bookingId when he tries to view the booking or cancel the booking.

3. NullPointerException handle in main method.

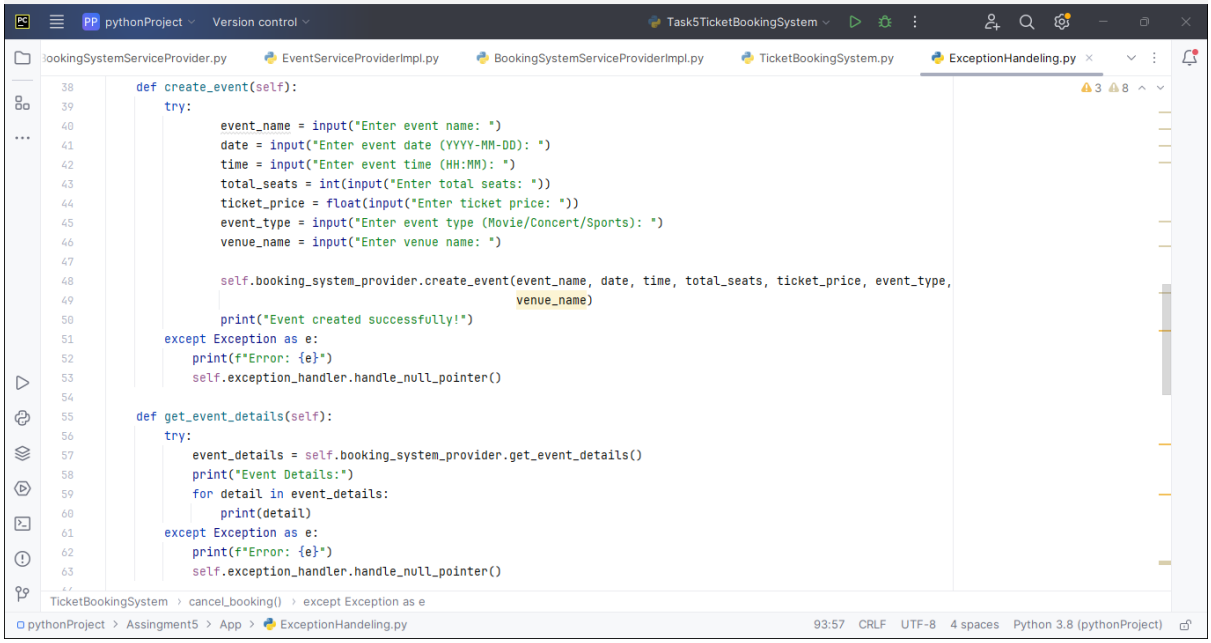
Throw these exceptions from the methods in TicketBookingSystem class. Make necessary changes to accommodate exception in the source code. Handle all these exceptions from the main program.



```
from Assignment5.Bean.BookingSystemServiceProviderImpl import BookingSystemServiceProviderImpl

2 usages
class EventNotFoundException(Exception):
    pass

3 usages
class InvalidBookingIDException(Exception):
    pass

1 usage
class CustomNullPointerException(Exception):
    pass

1 usage
class TicketBookingExceptionHandler:

    1 usage
    def handle_event_not_found(self, event_name):
        raise EventNotFoundException(f"Event '{event_name}' not found in the menu.")

    2 usages
    def handle_invalid_booking_id(self):
        raise InvalidBookingIDException("Invalid Booking ID.")

    5 usages
```



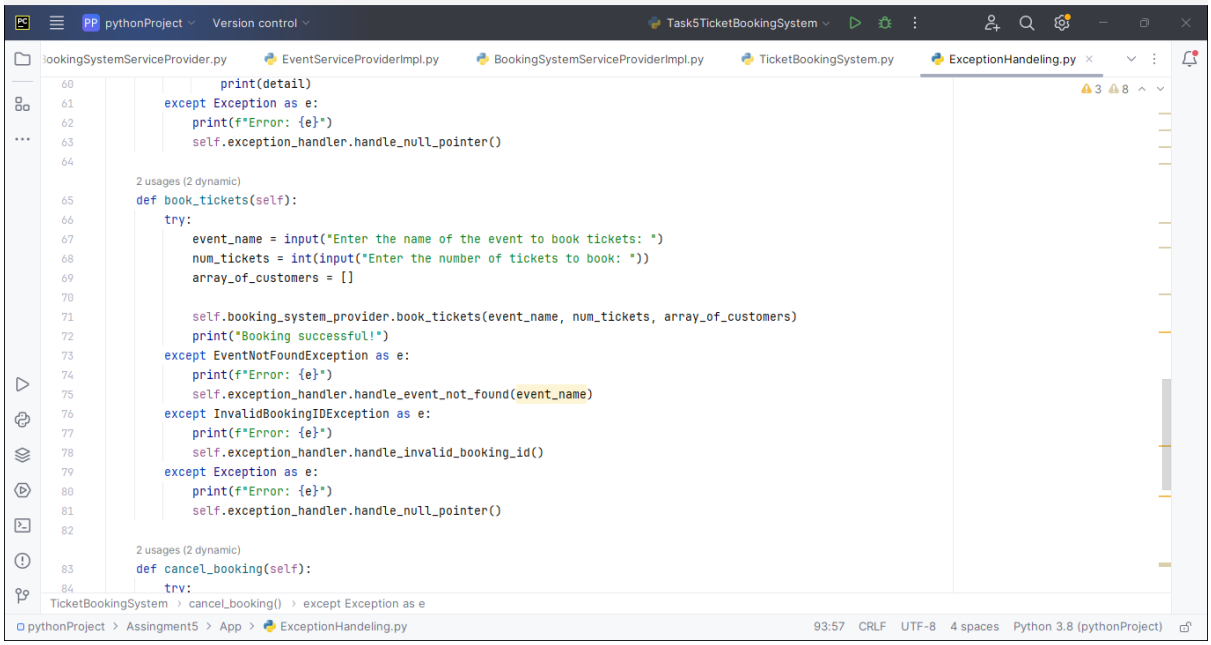
```
    def handle_null_pointer(self):
        raise CustomNullPointerException("Null Pointer Exception.")

class TicketBookingSystem:

    def __init__(self):
        self.booking_system_provider = BookingSystemServiceProviderImpl()
        self.exception_handler = TicketBookingExceptionHandler()

    def display_menu(self):
        print("\n===== Ticket Booking System Menu =====")
        print("1. Create Event")
        print("2. Get Event Details")
        print("3. Book Tickets")
        print("4. Cancel Booking")
        print("5. Get Available Seats")
        print("6. Exit")

    def create_event(self):
        try:
                event_name = input("Enter event name: ")
                date = input("Enter event date (YYYY-MM-DD): ")
                time = input("Enter event time (HH:MM): ")
                total_seats = int(input("Enter total seats: "))
                ticket_price = float(input("Enter ticket price: "))
                event_type = input("Enter event type (Movie/Concert/Sports): ")
                venue_name = input("Enter venue name: ")
```



```
    def create_event(self):
        try:
                event_name = input("Enter event name: ")
                date = input("Enter event date (YYYY-MM-DD): ")
                time = input("Enter event time (HH:MM): ")
                total_seats = int(input("Enter total seats: "))
                ticket_price = float(input("Enter ticket price: "))
                event_type = input("Enter event type (Movie/Concert/Sports): ")
                venue_name = input("Enter venue name: ")

                self.booking_system_provider.create_event(event_name, date, time, total_seats, ticket_price, event_type,
                                                          venue_name)
                print("Event created successfully!")
        except Exception as e:
            print(f"Error: {e}")
            self.exception_handler.handle_null_pointer()

    def get_event_details(self):
        try:
            event_details = self.booking_system_provider.get_event_details()
            print("Event Details:")
            for detail in event_details:
                print(detail)
        except Exception as e:
            print(f"Error: {e}")
            self.exception_handler.handle_null_pointer()
```

TicketBookingSystem › cancel_booking() › except Exception as e

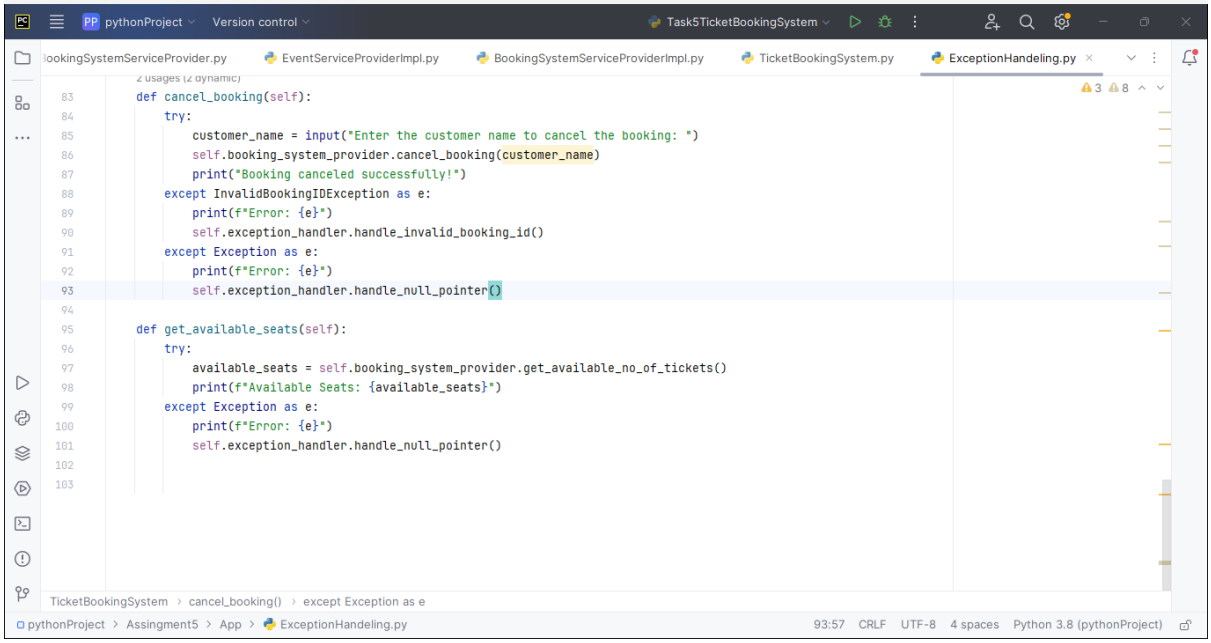
pythonProject › Assignment5 › App › ExceptionHandeling.py 93:57 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)



```
                print(detail)
        except Exception as e:
            print(f"Error: {e}")
            self.exception_handler.handle_null_pointer()

    2 usages (2 dynamic)
    def book_tickets(self):
        try:
            event_name = input("Enter the name of the event to book tickets: ")
            num_tickets = int(input("Enter the number of tickets to book: "))
            array_of_customers = []

            self.booking_system_provider.book_tickets(event_name, num_tickets, array_of_customers)
            print("Booking successful!")
        except EventNotFoundException as e:
            print(f"Error: {e}")
            self.exception_handler.handle_event_not_found(event_name)
        except InvalidBookingIDException as e:
            print(f"Error: {e}")
            self.exception_handler.handle_invalid_booking_id()
        except Exception as e:
            print(f"Error: {e}")
            self.exception_handler.handle_null_pointer()

    2 usages (2 dynamic)
    def cancel_booking(self):
        try:
```

TicketBookingSystem › cancel_booking() › except Exception as e

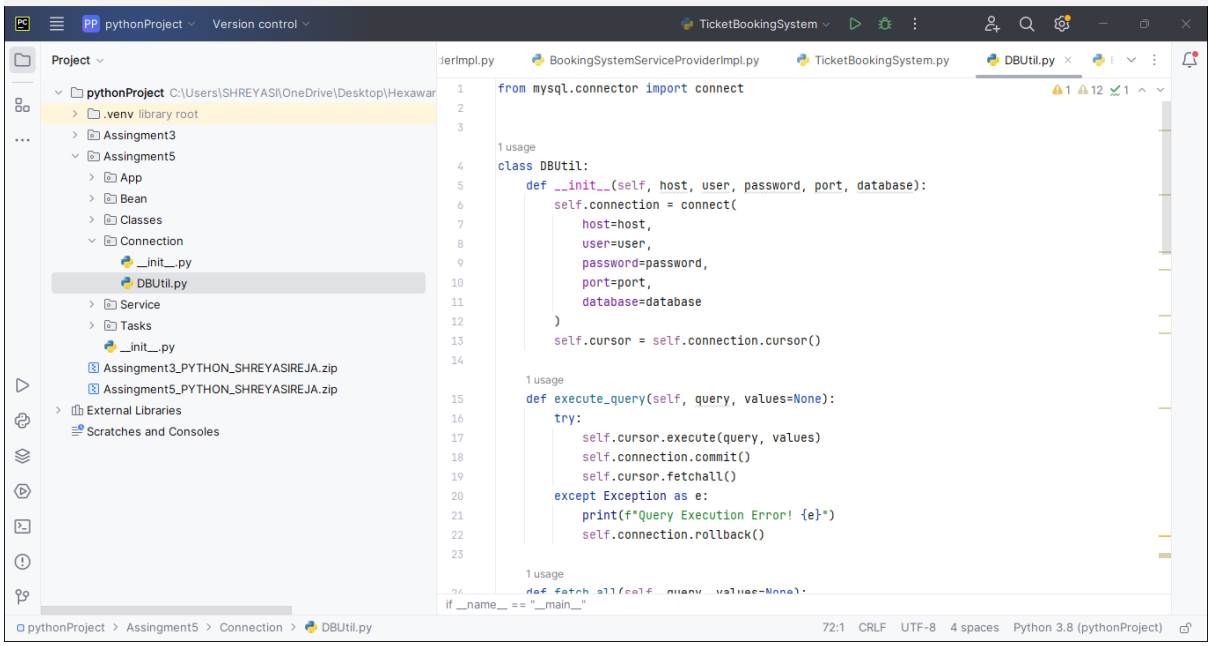
pythonProject › Assignment5 › App › ExceptionHandeling.py 93:57 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

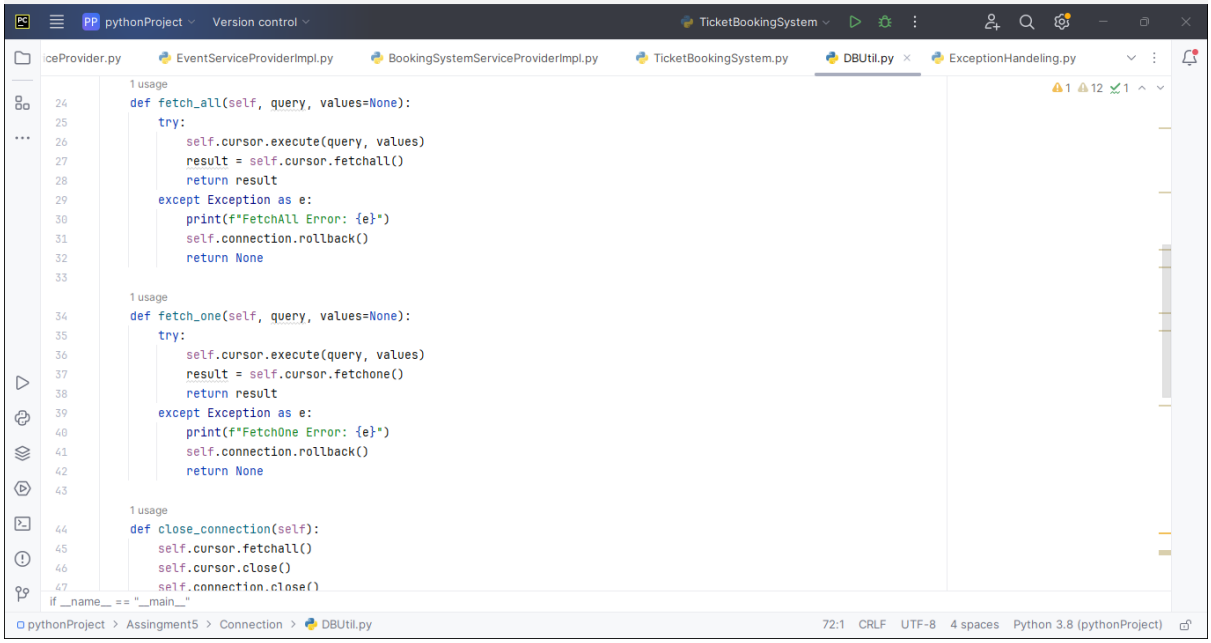


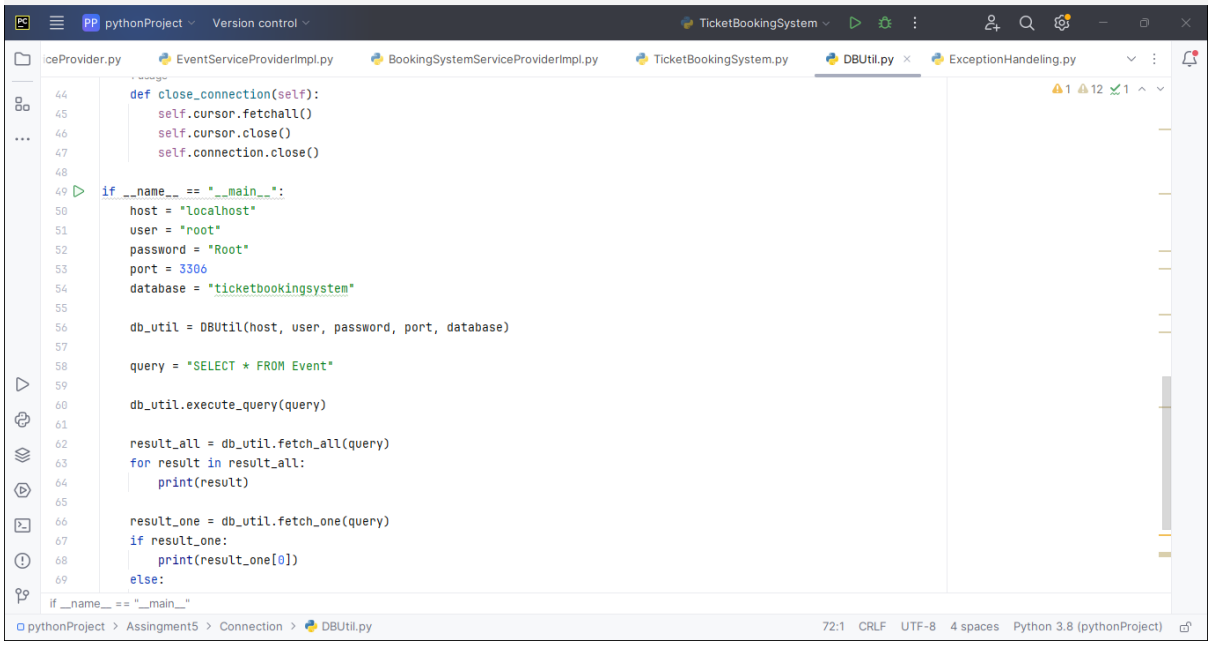
Task 11: Database Connectivity.

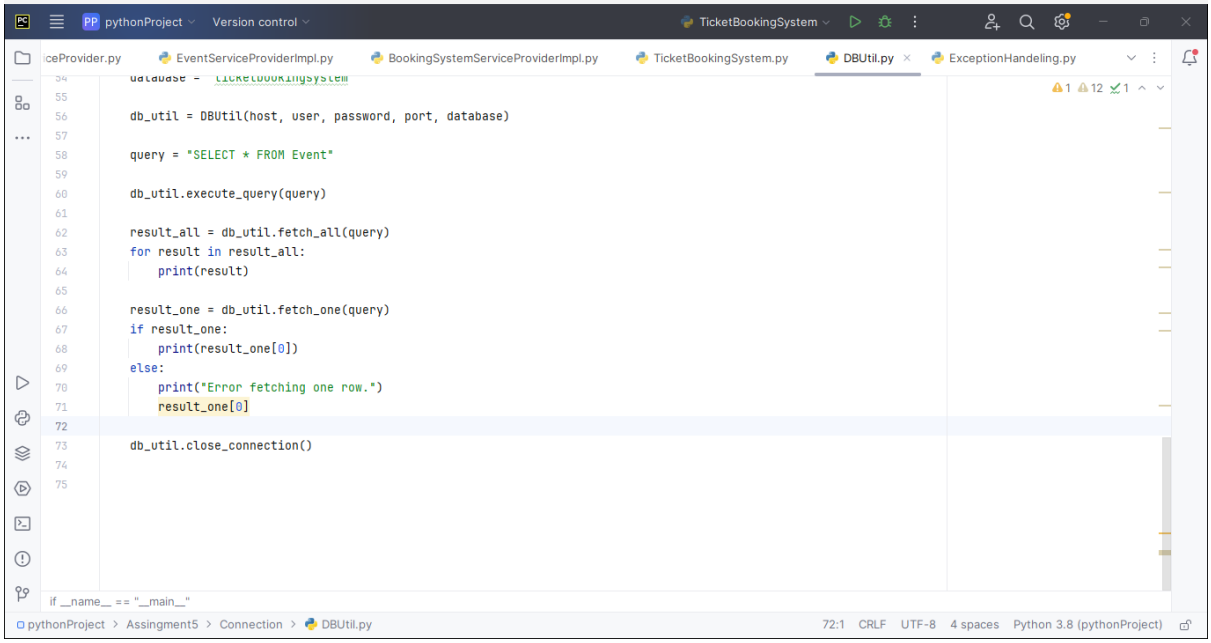
6. Create DBUtil class and add the following method.

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



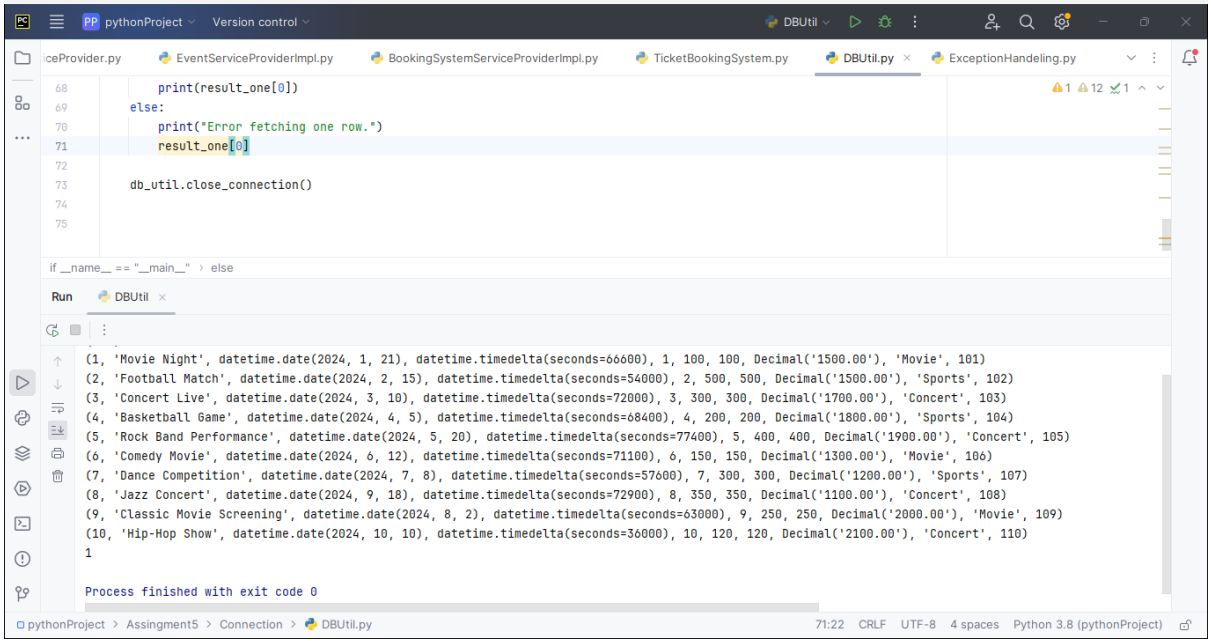

pythonProject
Version control
TicketBookingSystem
ceProvider.py
EventServiceProviderImpl.py
BookingSystemServiceProviderImpl.py
TicketBookingSystem.py
DBUtil.py
ExceptionHandeling.py
1 usage
def fetch_all(self, query, values=None):
try:
self.cursor.execute(query, values)
result = self.cursor.fetchall()
return result
except Exception as e:
print(f"FetchAll Error: {e}")
self.connection.rollback()
return None
1 usage
def fetch_one(self, query, values=None):
try:
self.cursor.execute(query, values)
result = self.cursor.fetchone()
return result
except Exception as e:
print(f"FetchOne Error: {e}")
self.connection.rollback()
return None
1 usage
def close_connection(self):
self.cursor.fetchall()
self.cursor.close()
self.connection.close()
if __name__ == "__main__"
pythonProject > Assignment5 > Connection > DBUtil.py
72:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)


pythonProject
Version control
TicketBookingSystem
ceProvider.py
EventServiceProviderImpl.py
BookingSystemServiceProviderImpl.py
TicketBookingSystem.py
DBUtil.py
ExceptionHandeling.py
def close_connection(self):
self.cursor.fetchall()
self.cursor.close()
self.connection.close()
if __name__ == "__main__":
host = "localhost"
user = "root"
password = "Root"
port = 3306
database = "ticketbookingsystem"
db_util = DBUtil(host, user, password, port, database)
query = "SELECT * FROM Event"
db_util.execute_query(query)
result_all = db_util.fetch_all(query)
for result in result_all:
print(result)
result_one = db_util.fetch_one(query)
if result_one:
print(result_one[0])
else:
if __name__ == "__main__"
pythonProject > Assignment5 > Connection > DBUtil.py
72:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)



```
database = ticketbookingsystem

db_util = DBUtil(host, user, password, port, database)

query = "SELECT * FROM Event"

db_util.execute_query(query)

result_all = db_util.fetch_all(query)
for result in result_all:
    print(result)

result_one = db_util.fetch_one(query)
if result_one:
    print(result_one[0])
else:
    print("Error fetching one row.")
    result_one[0]

db_util.close_connection()
```

Connect with database:-



```
    print(result_one[0])
else:
    print("Error fetching one row.")
    result_one[0]

db_util.close_connection()
```

```
(1, 'Movie Night', datetime.date(2024, 1, 21), datetime.timedelta(seconds=66600), 1, 100, 100, Decimal('1500.00'), 'Movie', 101)
(2, 'Football Match', datetime.date(2024, 2, 15), datetime.timedelta(seconds=54000), 2, 500, 500, Decimal('1500.00'), 'Sports', 102)
(3, 'Concert Live', datetime.date(2024, 3, 10), datetime.timedelta(seconds=72000), 3, 300, 300, Decimal('1700.00'), 'Concert', 103)
(4, 'Basketball Game', datetime.date(2024, 4, 5), datetime.timedelta(seconds=68400), 4, 200, 200, Decimal('1800.00'), 'Sports', 104)
(5, 'Rock Band Performance', datetime.date(2024, 5, 20), datetime.timedelta(seconds=77400), 5, 400, 400, Decimal('1900.00'), 'Concert', 105)
(6, 'Comedy Movie', datetime.date(2024, 6, 12), datetime.timedelta(seconds=71100), 6, 150, 150, Decimal('1300.00'), 'Movie', 106)
(7, 'Dance Competition', datetime.date(2024, 7, 8), datetime.timedelta(seconds=57600), 7, 300, 300, Decimal('1200.00'), 'Sports', 107)
(8, 'Jazz Concert', datetime.date(2024, 9, 18), datetime.timedelta(seconds=72900), 8, 350, 350, Decimal('1100.00'), 'Concert', 108)
(9, 'Classic Movie Screening', datetime.date(2024, 8, 2), datetime.timedelta(seconds=63000), 9, 250, 250, Decimal('2000.00'), 'Movie', 109)
(10, 'Hip-Hop Show', datetime.date(2024, 10, 10), datetime.timedelta(seconds=36000), 10, 120, 120, Decimal('2100.00'), 'Concert', 110)
1

Process finished with exit code 0
```