Ticket Booking System

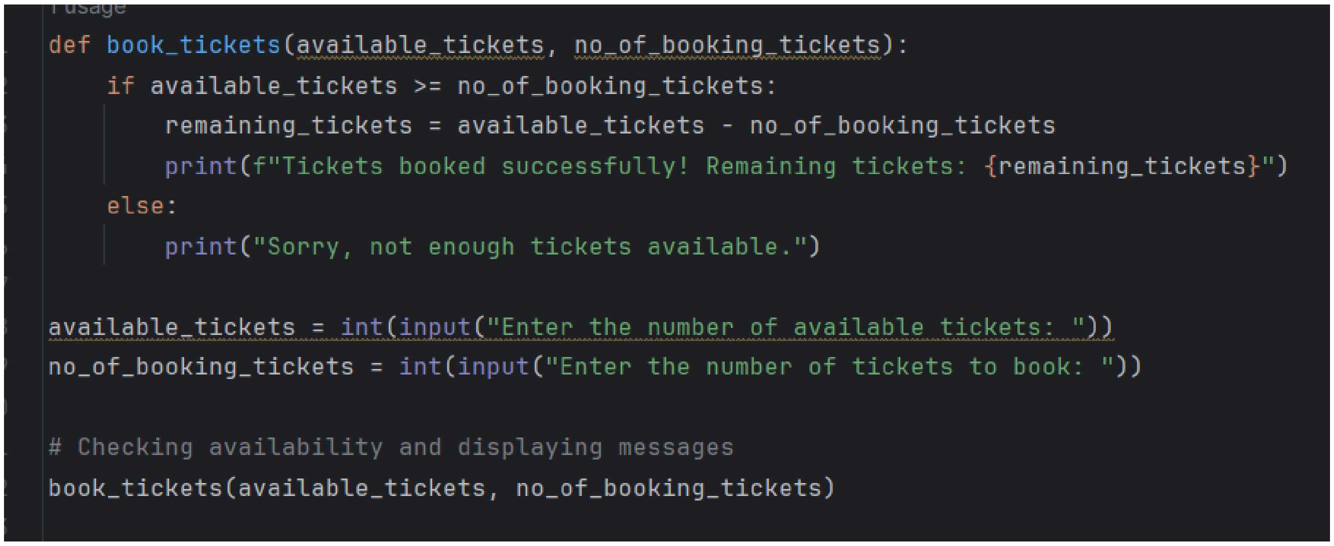
Control structure

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:

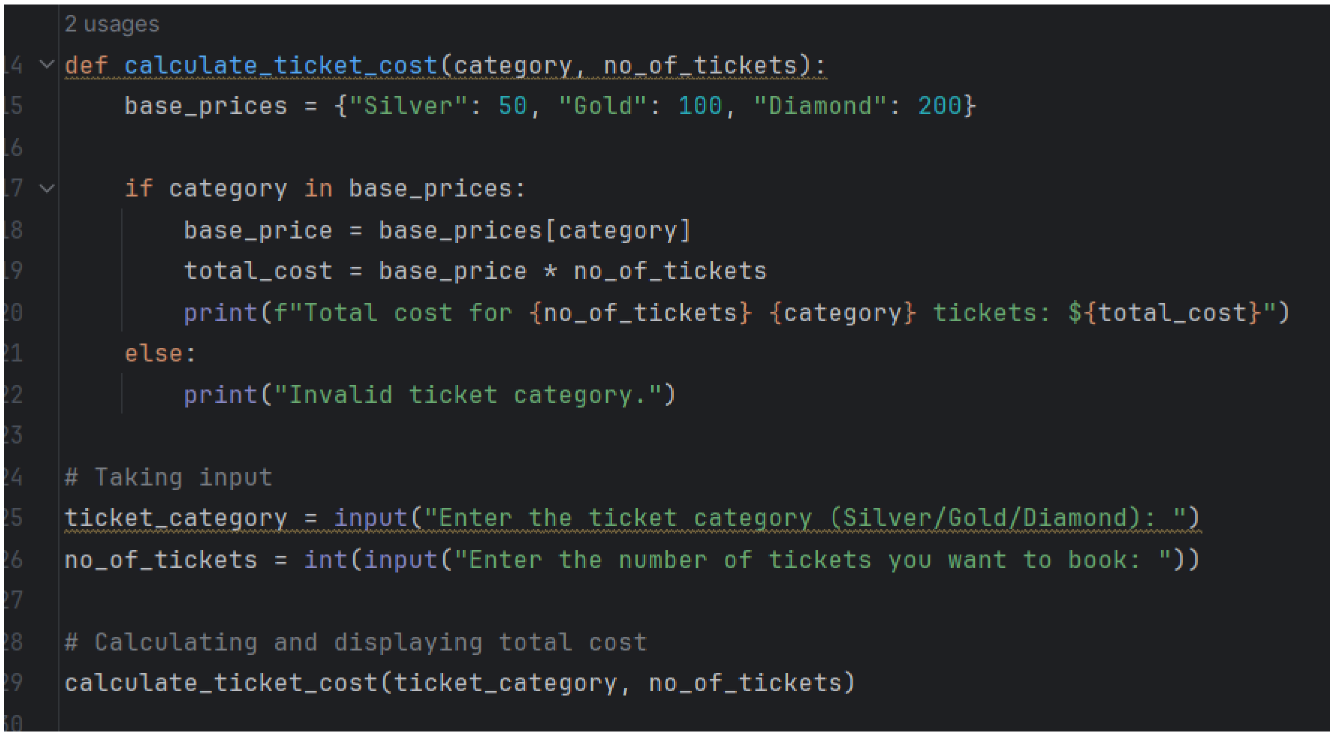
1. Write a program that takes the availableTicket and noOfBookingTicket as input.
2. Use conditional statements (if-else) to determine if the ticket is available or not.
3. Display an appropriate message based on ticket availability.



```
1 usage
def book_tickets(available_tickets, no_of_booking_tickets):
    if available_tickets >= no_of_booking_tickets:
        remaining_tickets = available_tickets - no_of_booking_tickets
        print(f"Tickets booked successfully! Remaining tickets: {remaining_tickets}")
    else:
        print("Sorry, not enough tickets available.")

available_tickets = int(input("Enter the number of available tickets: "))
no_of_booking_tickets = int(input("Enter the number of tickets to book: "))

# Checking availability and displaying messages
book_tickets(available_tickets, no_of_booking_tickets)
```

Task 2: Nested Conditional Statements

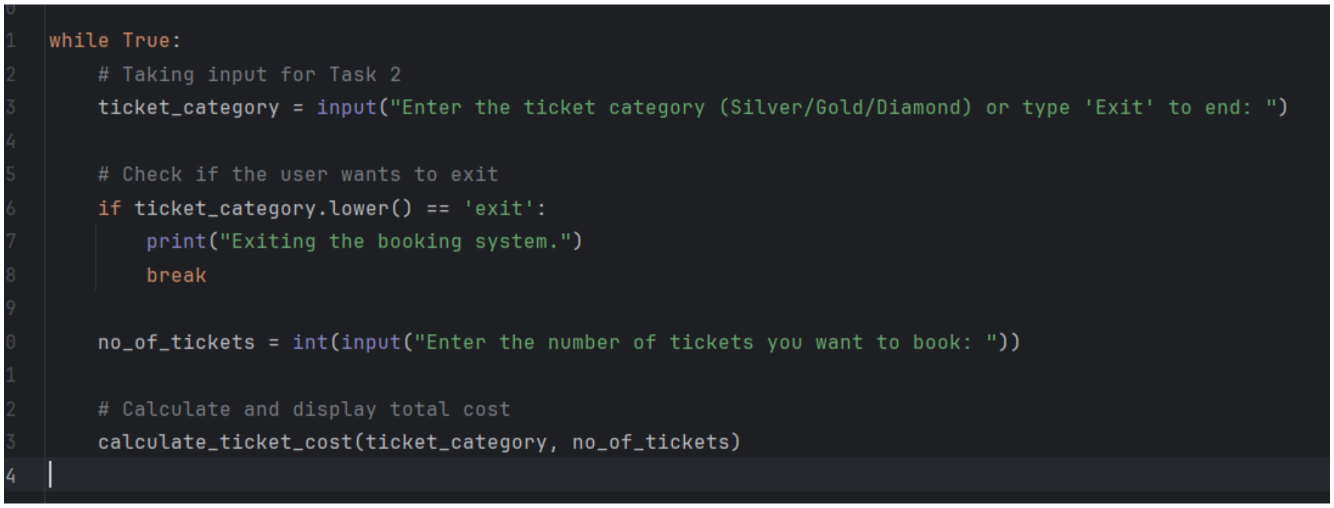
Create a program that simulates a Ticket booking and calculating cost of tickets. Display tickets options such as "Silver", "Gold", "Dimond". Based on ticket category fix the base ticket price and get the user input for ticket type and no of tickets need and calculate the total cost of tickets booked.



```
2 usages
def calculate_ticket_cost(category, no_of_tickets):
    base_prices = {"Silver": 50, "Gold": 100, "Diamond": 200}

    if category in base_prices:
        base_price = base_prices[category]
        total_cost = base_price * no_of_tickets
        print(f"Total cost for {no_of_tickets} {category} tickets: ${total_cost}")
    else:
        print("Invalid ticket category.")

# Taking input
ticket_category = input("Enter the ticket category (Silver/Gold/Diamond): ")
no_of_tickets = int(input("Enter the number of tickets you want to book: "))

# Calculating and displaying total cost
calculate_ticket_cost(ticket_category, no_of_tickets)
```

Task 3: Looping

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



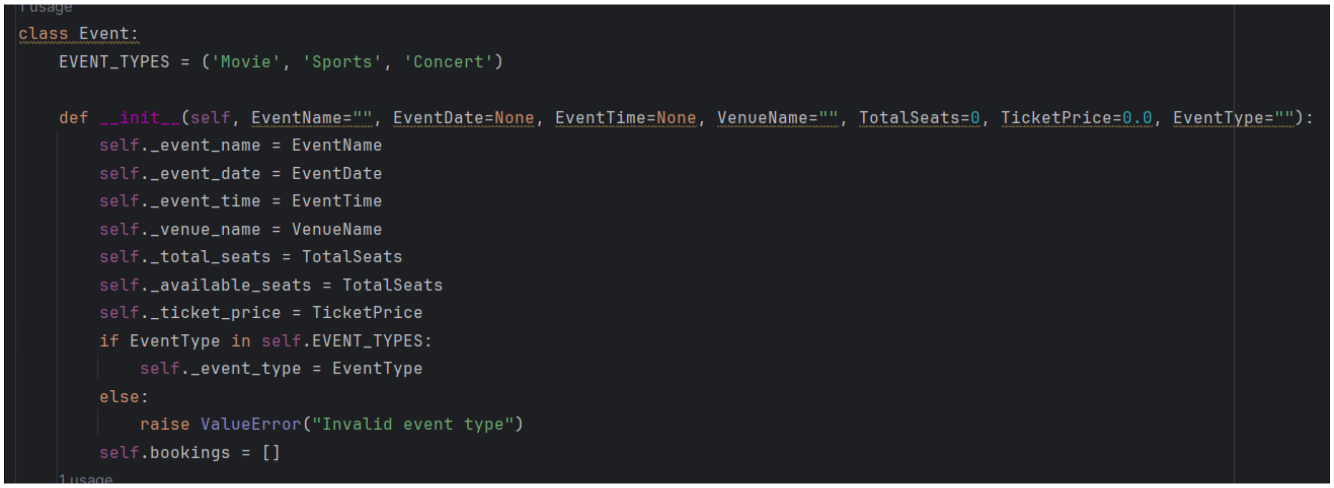
```
while True:
    # Taking input for Task 2
    ticket_category = input("Enter the ticket category (Silver/Gold/Diamond) or type 'Exit' to end: ")

    # Check if the user wants to exit
    if ticket_category.lower() == 'exit':
        print("Exiting the booking system.")
        break

    no_of_tickets = int(input("Enter the number of tickets you want to book: "))

    # Calculate and display total cost
    calculate_ticket_cost(ticket_category, no_of_tickets)
```

Task 4: Class & Object

Create a Following classes with the following attributes and methods:

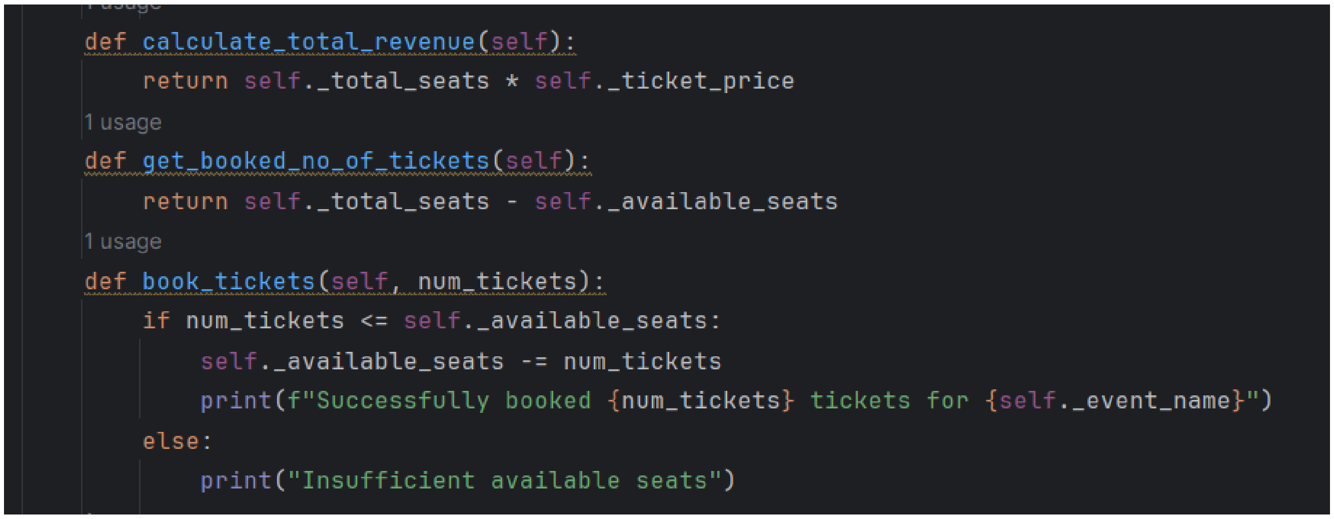
1. **Event** Class:
   - **Attributes:**
     - event_name,
     - event_date DATE,
     - event_time TIME,
     - venue_name,
     - total_seats,
     - available_seats,

- ticket_price DECIMAL,
- event_type ENUM('Movie', 'Sports', 'Concert')

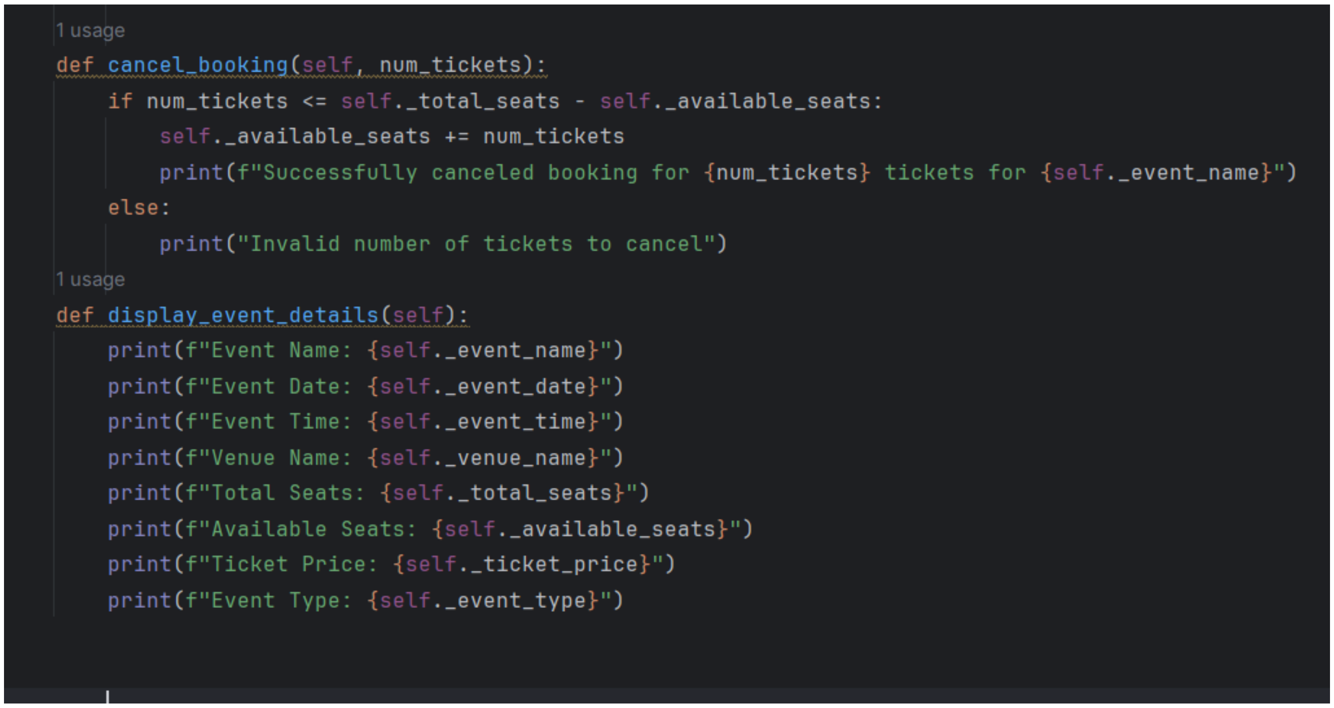


```python
class Event:
    EVENT_TYPES = ('Movie', 'Sports', 'Concert')

    def __init__(self, EventName="", EventDate=None, EventTime=None, VenueName="", TotalSeats=0, TicketPrice=0.0, EventType=""):
        self._event_name = EventName
        self._event_date = EventDate
        self._event_time = EventTime
        self._venue_name = VenueName
        self._total_seats = TotalSeats
        self._available_seats = TotalSeats
        self._ticket_price = TicketPrice
        if EventType in self.EVENT_TYPES:
            self._event_type = EventType
        else:
            raise ValueError("Invalid event type")
        self.bookings = []
```

- **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.

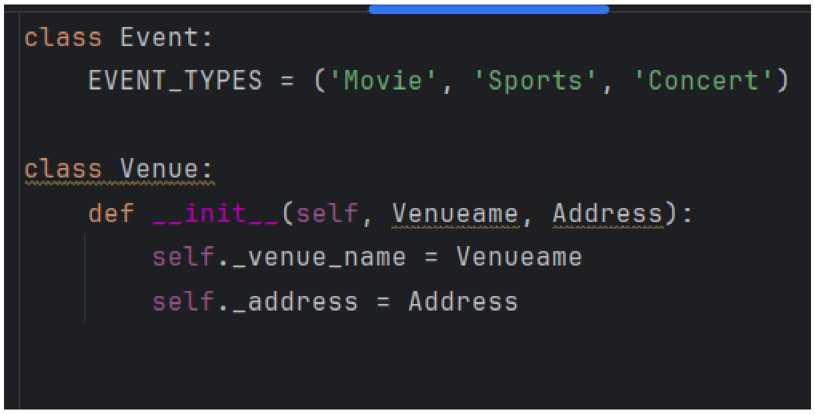


```python
    def calculate_total_revenue(self):
        return self._total_seats * self._ticket_price
    1 usage
    def get_booked_no_of_tickets(self):
        return self._total_seats - self._available_seats
    1 usage
    def book_tickets(self, num_tickets):
        if num_tickets <= self._available_seats:
            self._available_seats -= num_tickets
            print(f"Successfully booked {num_tickets} tickets for {self._event_name}")
        else:
            print("Insufficient available seats")
```



```
1 usage
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")
1 usage
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}")
```

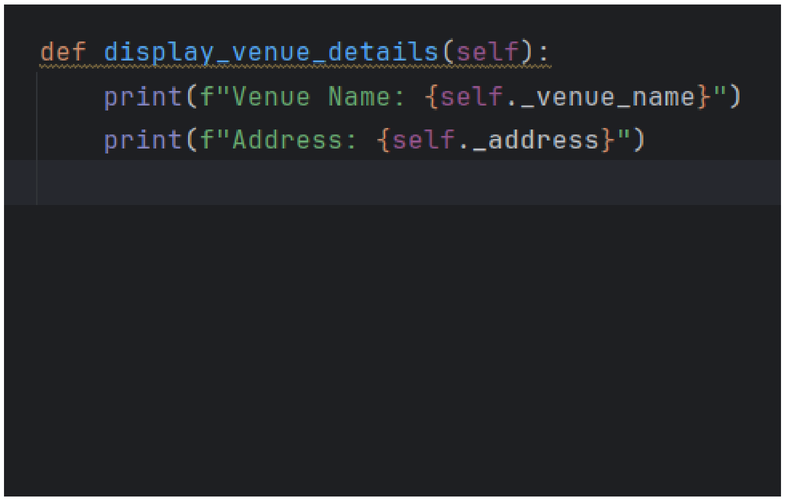
2. **Venue** Class
    - **Attributes**:
        - venue_name,
        - address



```
class Event:
    EVENT_TYPES = ('Movie', 'Sports', 'Concert')

class Venue:
    def __init__(self, Venueame, Address):
        self._venue_name = Venueame
        self._address = Address
```

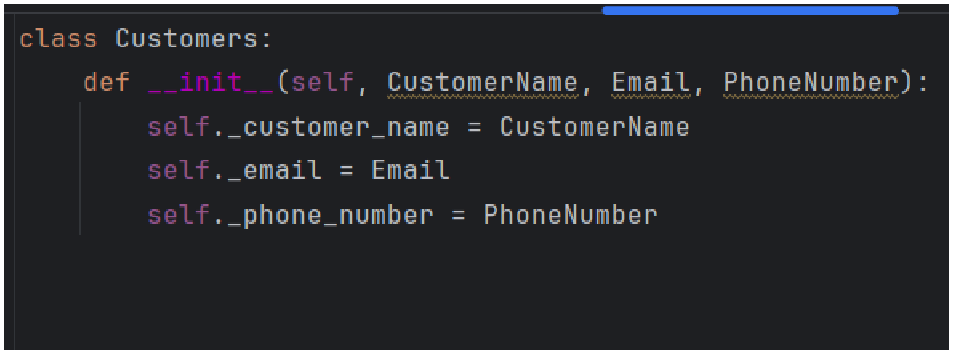
- **Methods and Constuctors:**
    - **display_venue_details():** Display venue details.

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



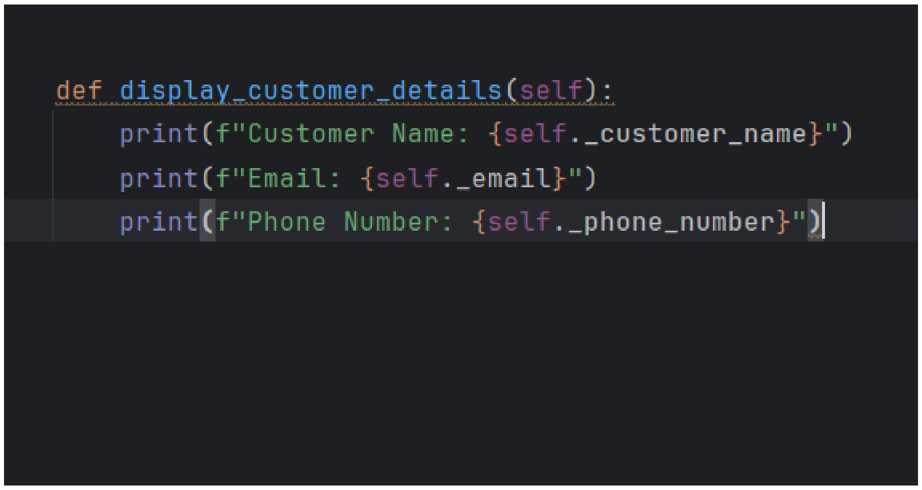
```
def display_venue_details(self):
    print(f"Venue Name: {self._venue_name}")
    print(f"Address: {self._address}")
```

3. **Customer** Class
    - **Attributes:**
        - customer_name,
        - email,
        - phone_number,



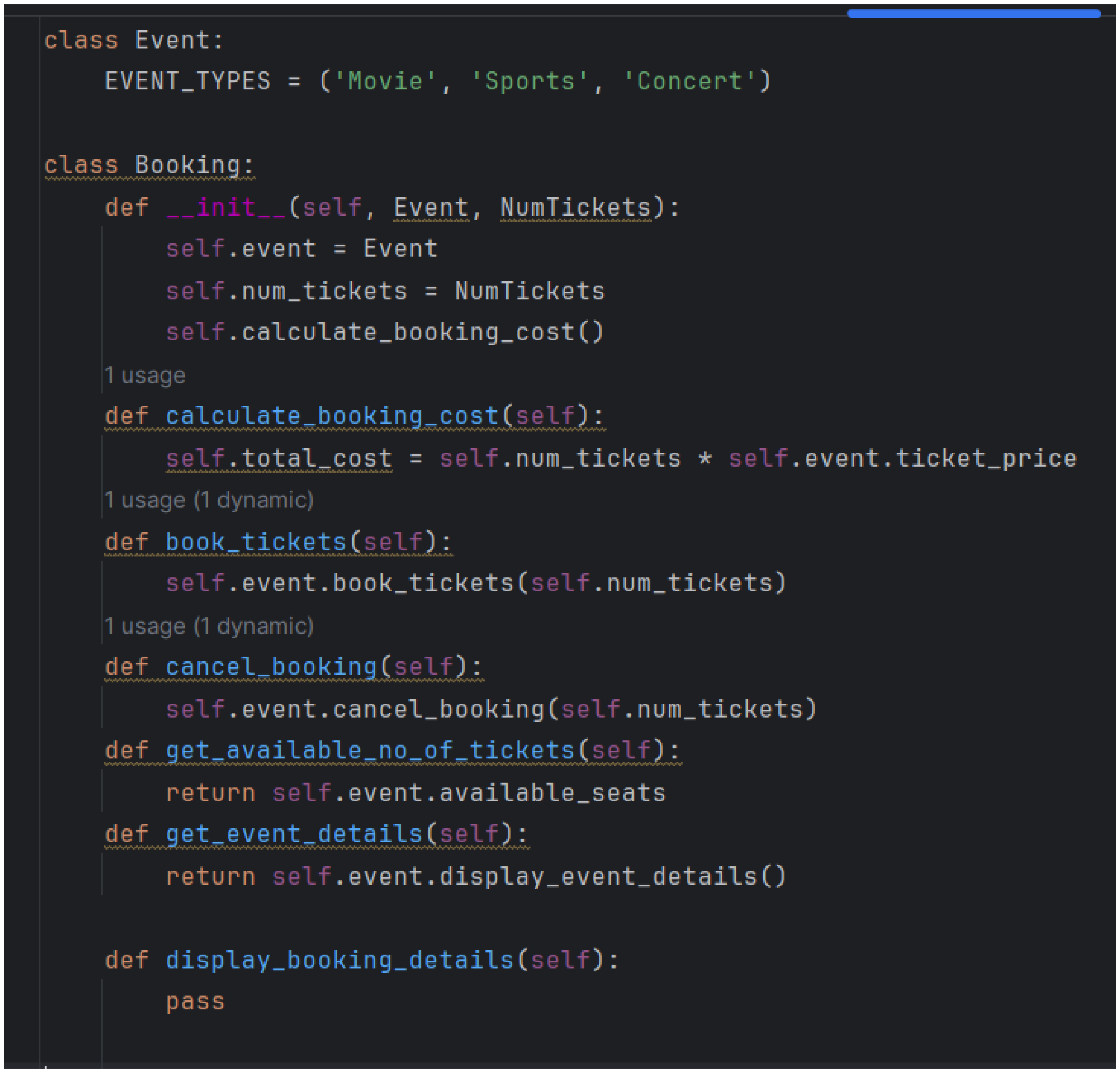
```
class Customers:
    def __init__(self, CustomerName, Email, PhoneNumber):
        self._customer_name = CustomerName
        self._email = Email
        self._phone_number = PhoneNumber
```

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



```
def display_customer_details(self):
    print(f"Customer Name: {self._customer_name}")
    print(f"Email: {self._email}")
    print(f"Phone Number: {self._phone_number}")
```

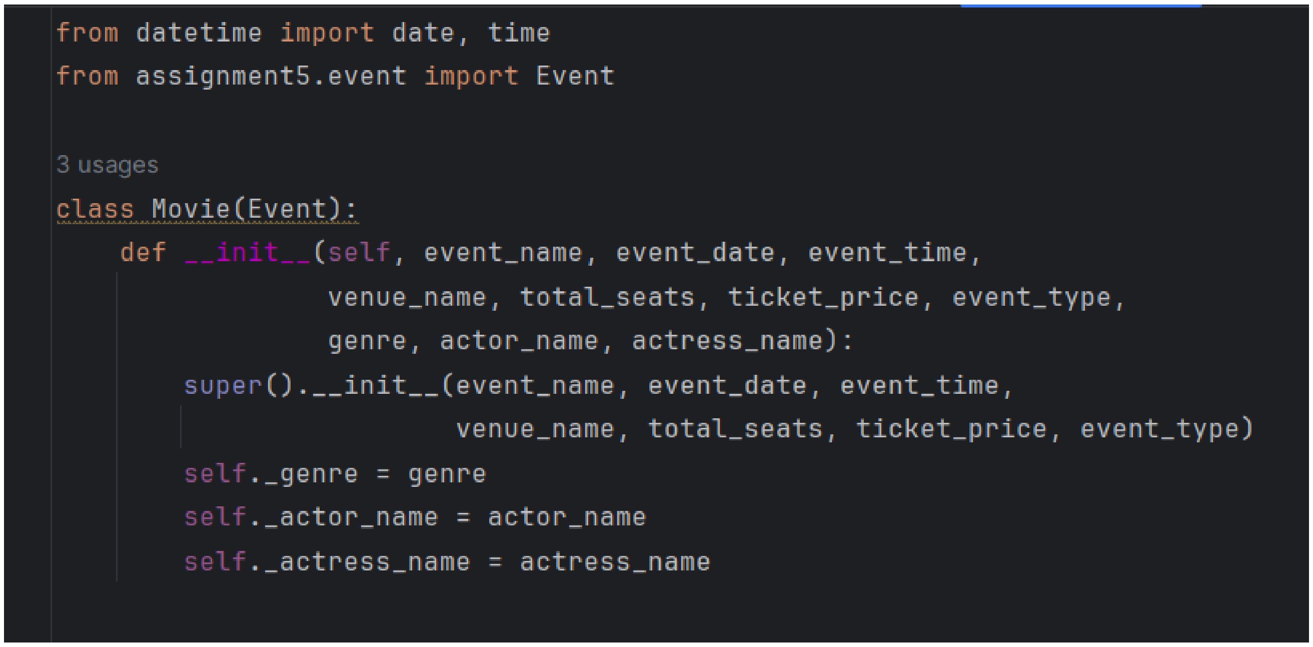
4. **Booking** Class to represent the Tiket booking system. Perform the following operation in main method. Note:- Use Event class object for the following operation.
   - **Methods and Constuctors:**
     - **calculate_booking_cost(num_tickets)**: Calculate and set the total cost of the booking.
     - **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



```
class Event:
    EVENT_TYPES = ('Movie', 'Sports', 'Concert')

class Booking:
    def __init__(self, Event, NumTickets):
        self.event = Event
        self.num_tickets = NumTickets
        self.calculate_booking_cost()
    1 usage
    def calculate_booking_cost(self):
        self.total_cost = self.num_tickets * self.event.ticket_price
    1 usage (1 dynamic)
    def book_tickets(self):
        self.event.book_tickets(self.num_tickets)
    1 usage (1 dynamic)
    def cancel_booking(self):
        self.event.cancel_booking(self.num_tickets)
    def get_available_no_of_tickets(self):
        return self.event.available_seats
    def get_event_details(self):
        return self.event.display_event_details()

    def display_booking_details(self):
        pass
```

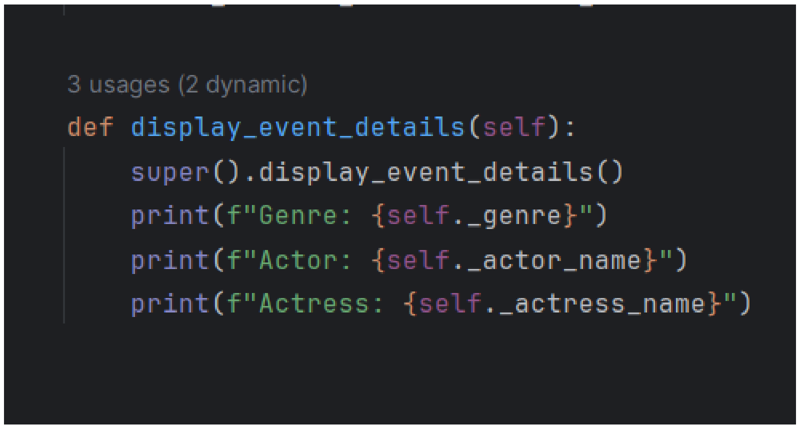
Task 5: Inheritance and polymorphism

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



```
from datetime import date, time
from assignment5.event import Event

3 usages
class Movie(Event):
    def __init__(self, event_name, event_date, event_time,
                 venue_name, total_seats, ticket_price, event_type,
                 genre, actor_name, actress_name):
        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
```

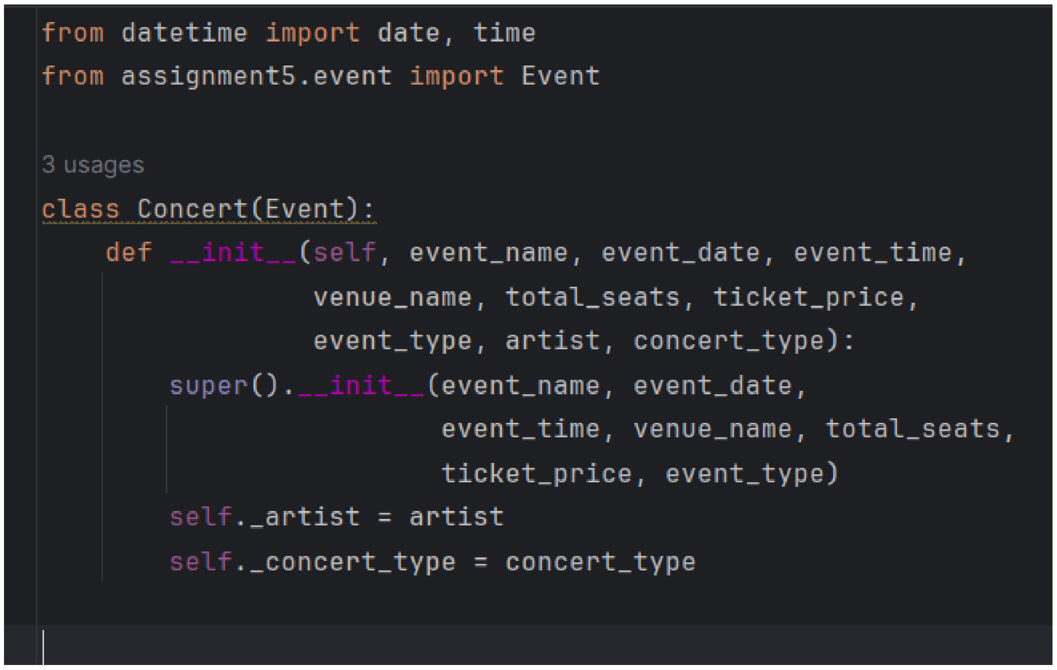
- **Methods:**
    1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.



```
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}")
```

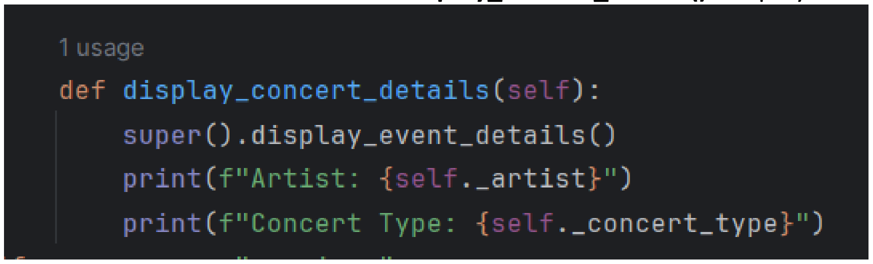
    2. **display_event_details():**Display movie details, including genre.

- 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)



```python
from datetime import date, time
from assignment5.event import Event

3 usages
class Concert(Event):
    def __init__(self, event_name, event_date, event_time,
                 venue_name, total_seats, ticket_price,
                 event_type, artist, concert_type):
        super().__init__(event_name, event_date,
                         event_time, venue_name, total_seats,
                         ticket_price, event_type)
        self._artist = artist
        self._concert_type = concert_type
```

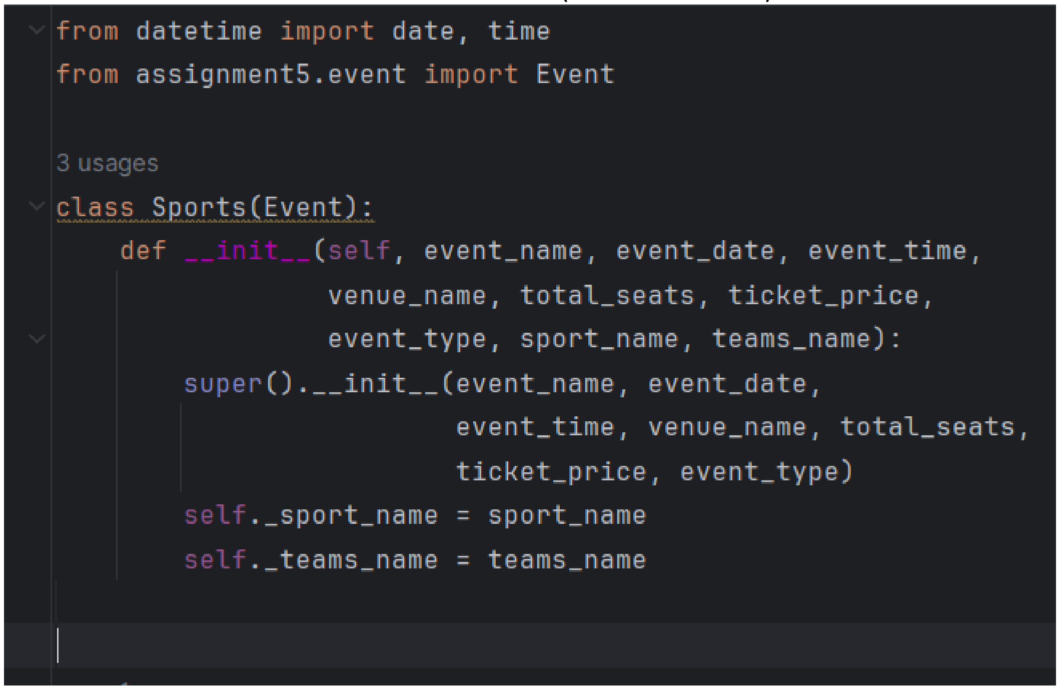
- **Methods:**
  1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
  2. **display_concert_details():** Display concert details, including the artist.



```python
1 usage
def display_concert_details(self):
    super().display_event_details()
    print(f"Artist: {self._artist}")
    print(f"Concert Type: {self._concert_type}")
```

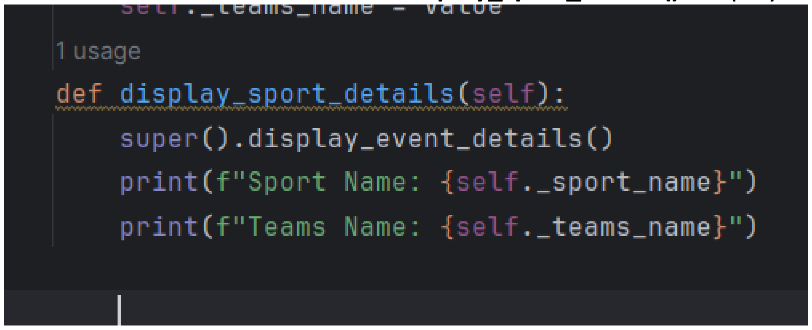
- Create another subclass **Sports** that inherits from **Event**. Add the following attributes and methods:
  - **Attributes:**

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



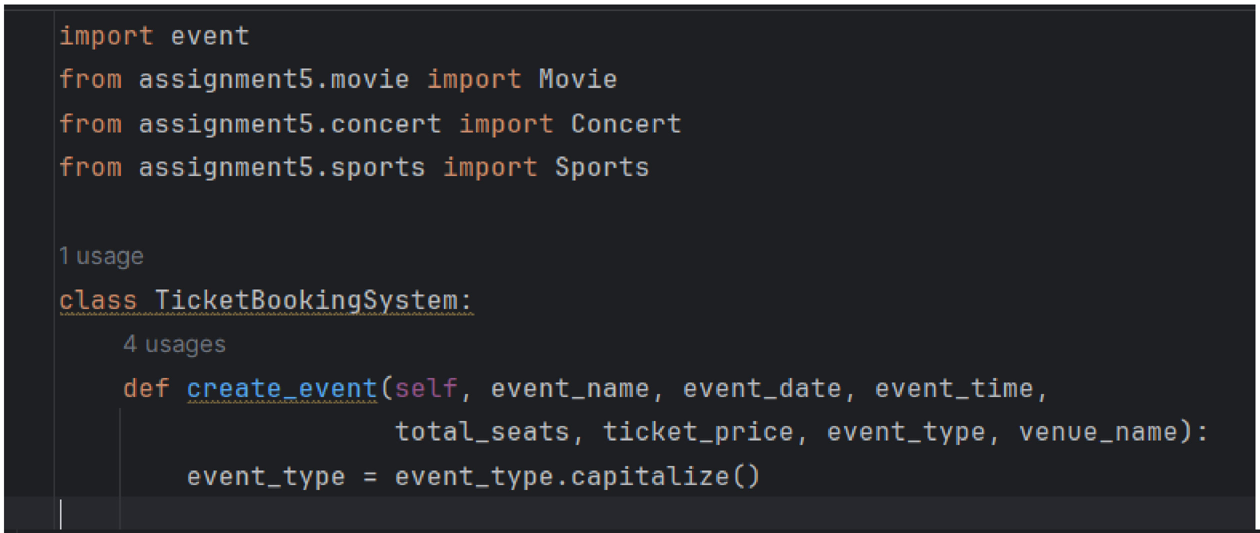
```
from datetime import date, time
from assignment5.event import Event

3 usages
class Sports(Event):
    def __init__(self, event_name, event_date, event_time,
                 venue_name, total_seats, ticket_price,
                 event_type, sport_name, teams_name):
        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
```

- **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.

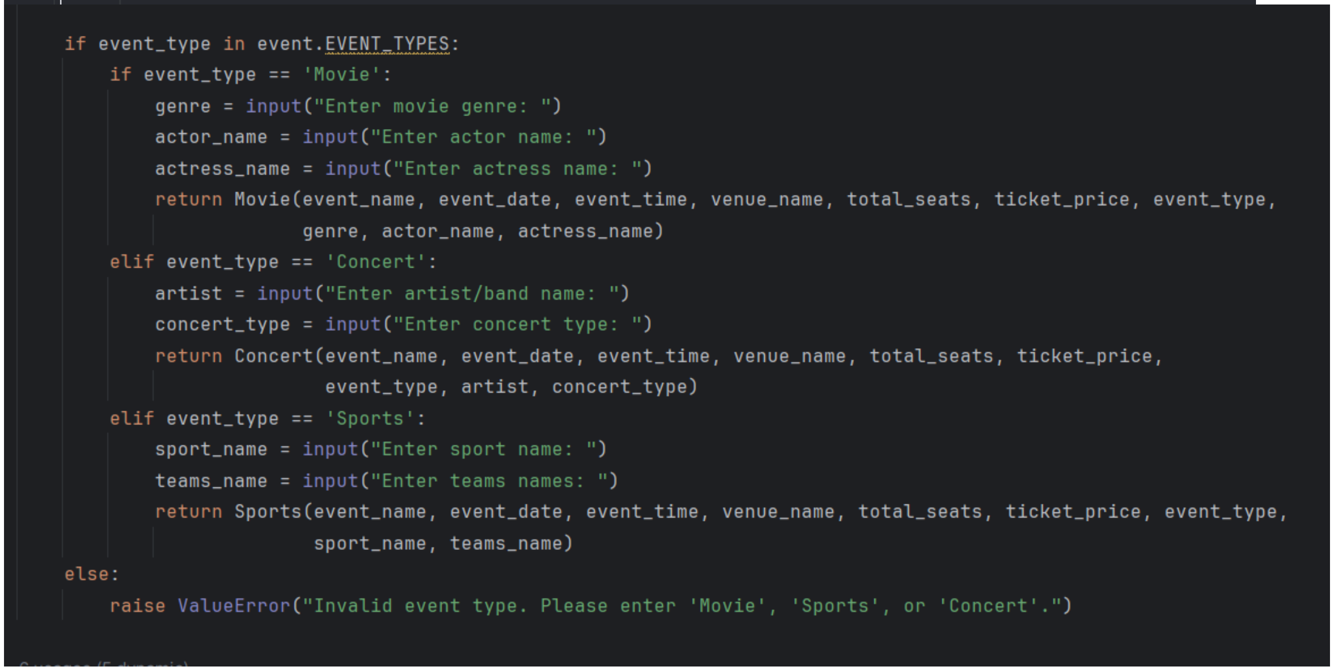


```
    self._teams_name = value

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}")
```

- Create a class **TicketBookingSystem** with the following methods:
  - **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu_name:str):** Create a new event with the specified details and event type (movie, sport or concert) and return event object.

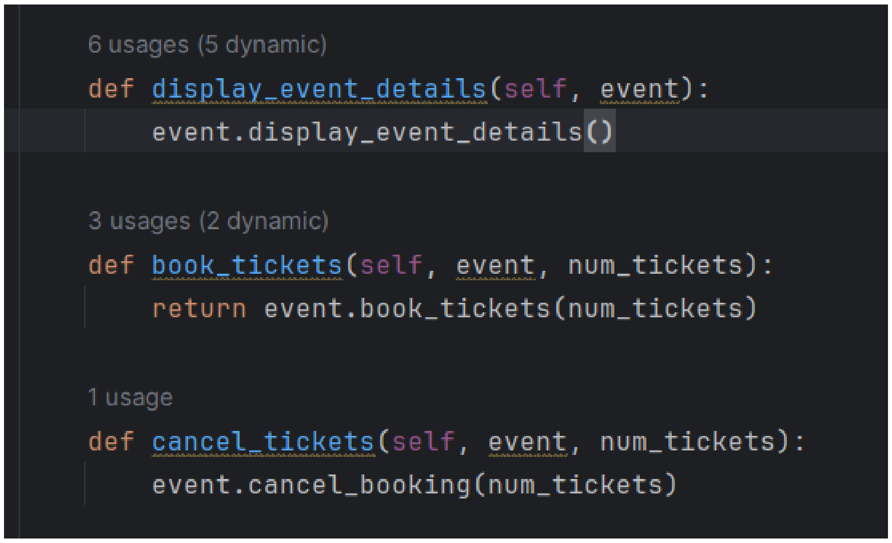


```
import event
from assignment5.movie import Movie
from assignment5.concert import Concert
from assignment5.sports import Sports

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()

```



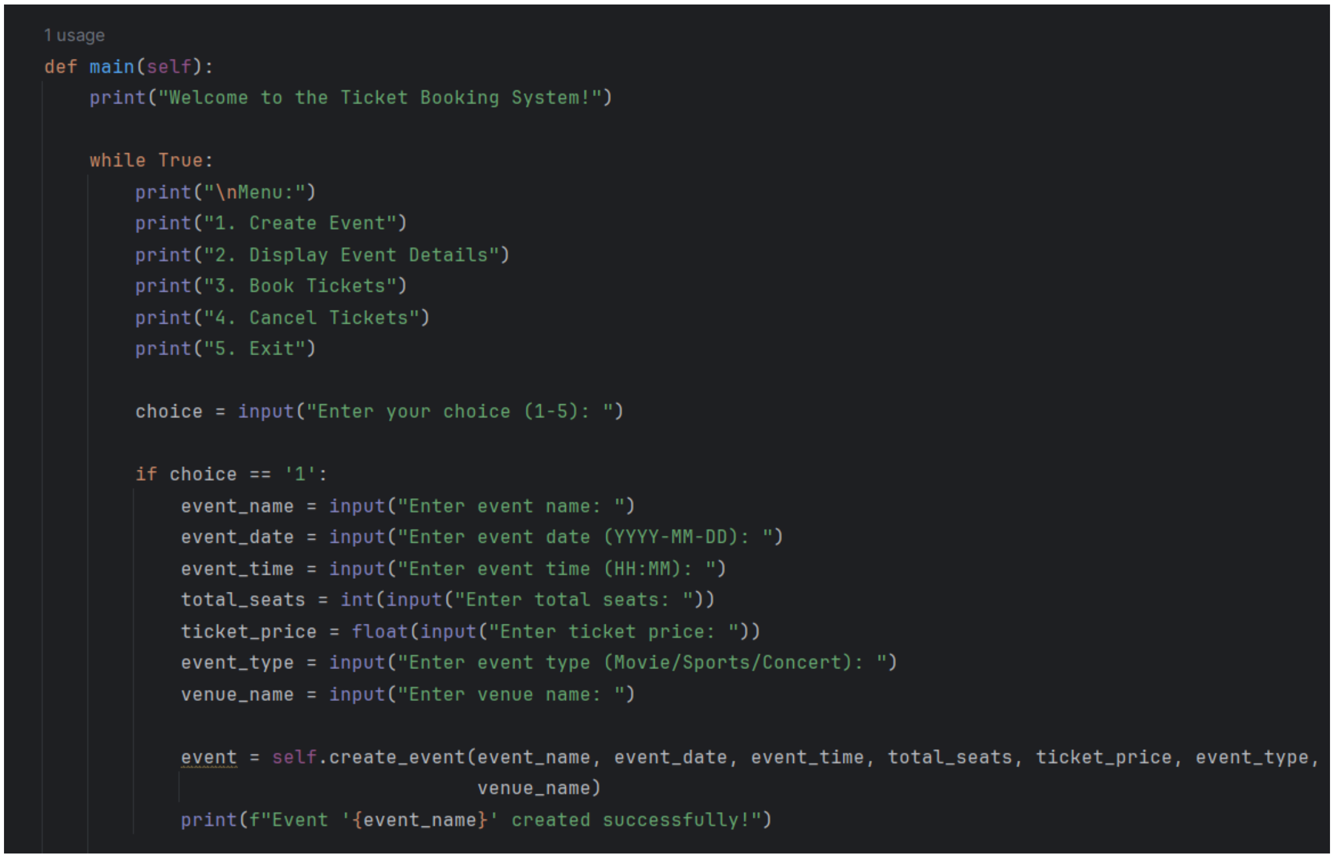
```
        if event_type in event.EVENT_TYPES:
            if event_type == 'Movie':
                genre = input("Enter movie genre: ")
                actor_name = input("Enter actor name: ")
                actress_name = input("Enter actress name: ")
                return Movie(event_name, event_date, 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'.")
```

- **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.

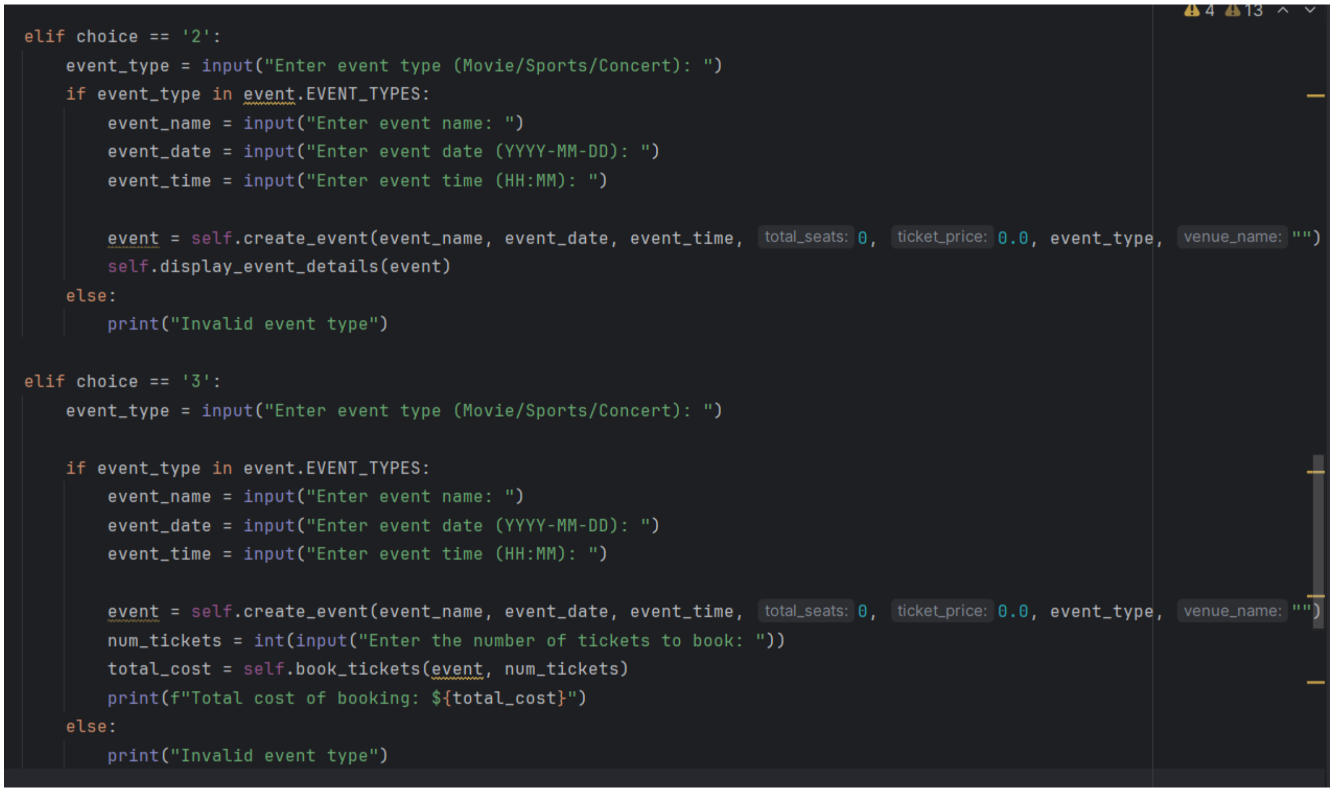


```
6 usages (5 dynamic)
def display_event_details(self, event):
    event.display_event_details()

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)
```

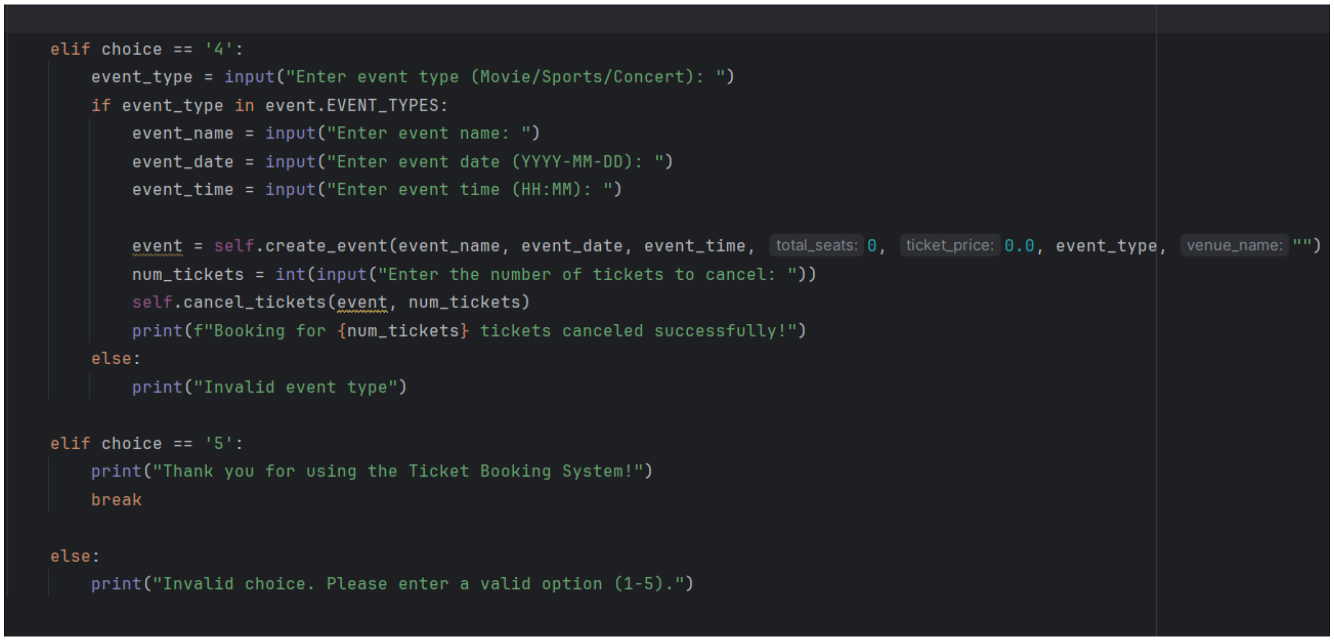
- **main():** simulates the ticket booking system
    1. User can book tickets and view the event details as per their choice in menu (movies, sports, concerts).
    2. Display event details using the display_event_details() method without knowing the specific event type (demonstrate polymorphism).
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



```
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")
        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!")
```



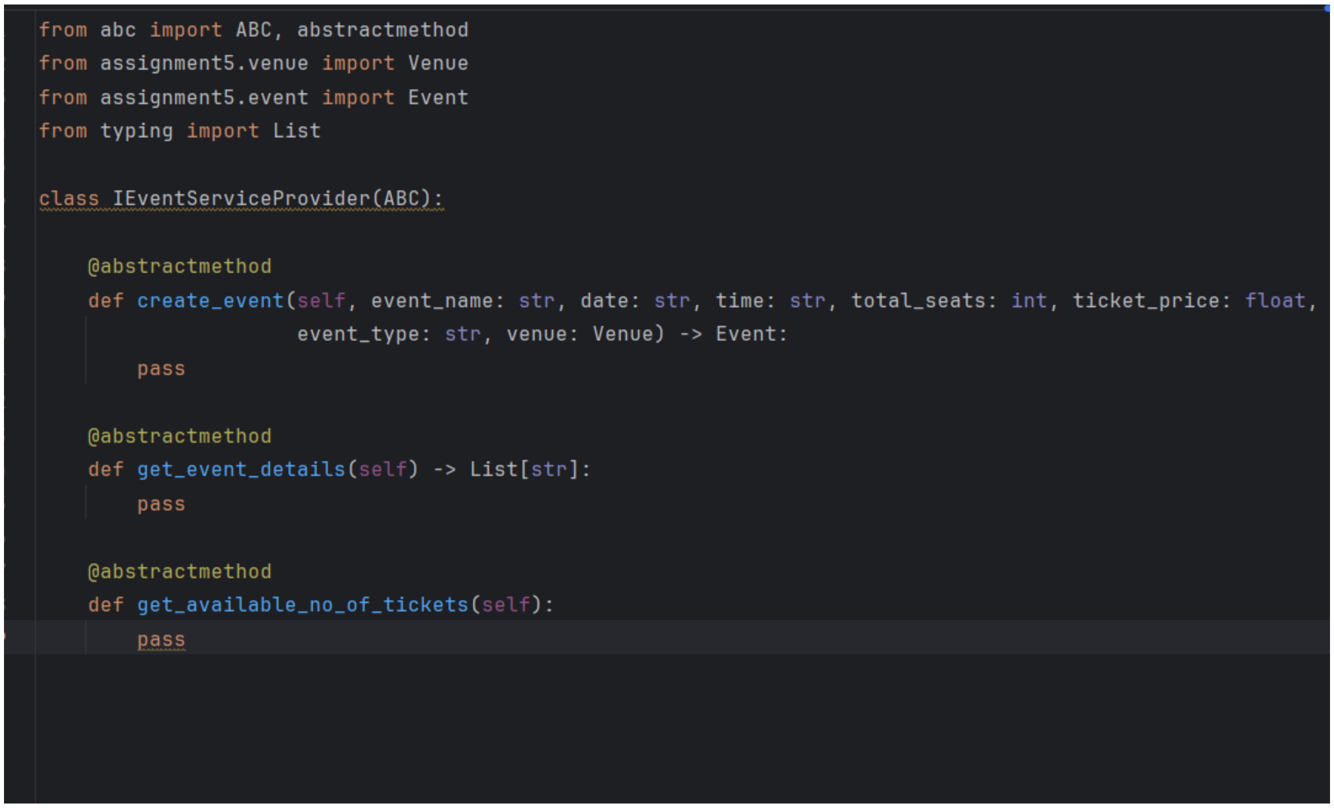
```
elif choice == '2':
    event_type = input("Enter event type (Movie/Sports/Concert): ")
    if event_type in event.EVENT_TYPES:
        event_name = input("Enter event name: ")
        event_date = input("Enter event date (YYYY-MM-DD): ")
        event_time = input("Enter event time (HH:MM): ")

        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).")
```

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

1. Create **Venue,** class as mentioned above Task 4.
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.
3. Create **Event** sub classes as mentioned in above Task 4**.**
4. Create a class **Customer** and **Booking** as mentioned in above Task 4**.**
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.

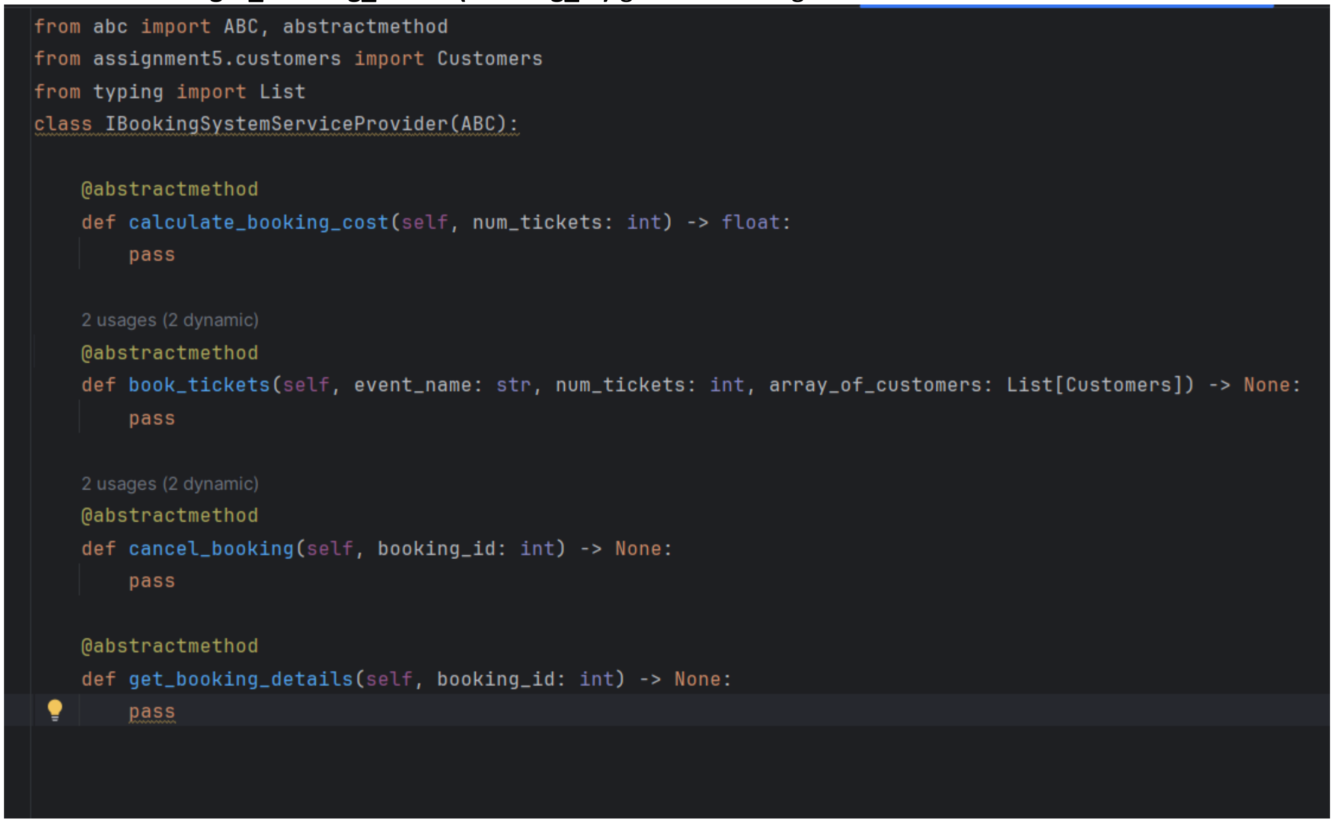


```
from abc import ABC, abstractmethod
from assignment5.venue import Venue
from assignment5.event import Event
from typing import List

class IEventServiceProvider(ABC):

    @abstractmethod
    def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue) -> Event:
        pass

    @abstractmethod
    def get_event_details(self) -> List[str]:
        pass

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

6. Create interface/abstract class **IBookingSystemServiceProvider** with following methods:
    - **calculate_booking_cost(num_tickets):** Calculate and set the total cost of the booking.
    - **book_tickets(eventname:str, num_tickets, arrayOfCustomer):** Book a specified number of

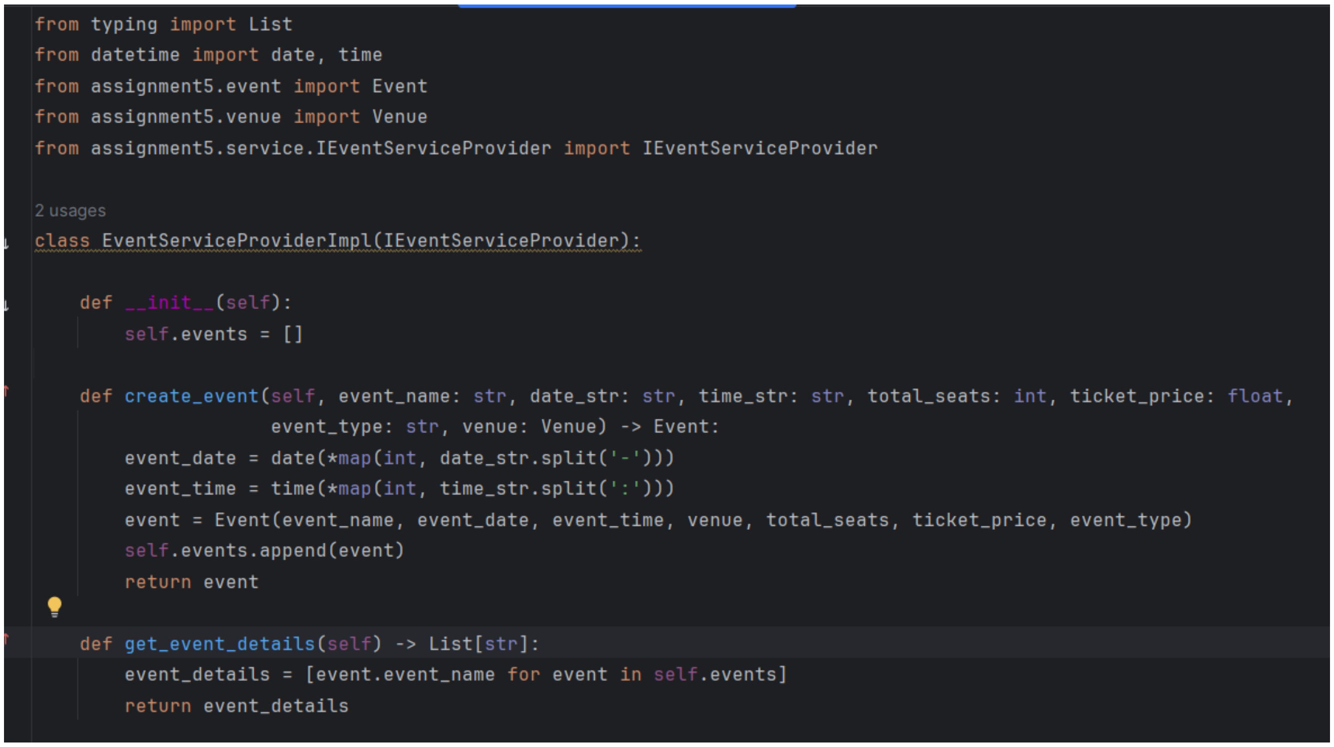
tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class.

- **cancel_booking(booking_id):** Cancel the booking and update the available seats.
- **get_booking_details(booking_id):**get the booking details.

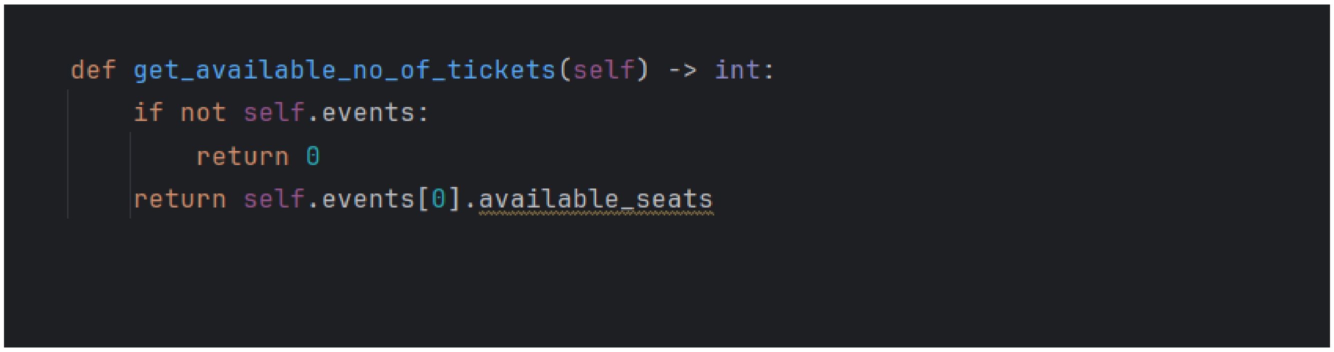


```
from abc import ABC, abstractmethod
from assignment5.customers import Customers
from typing import List
class IBookingSystemServiceProvider(ABC):

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

    2 usages (2 dynamic)
    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, array_of_customers: List[Customers]) -> None:
        pass

    2 usages (2 dynamic)
    @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.

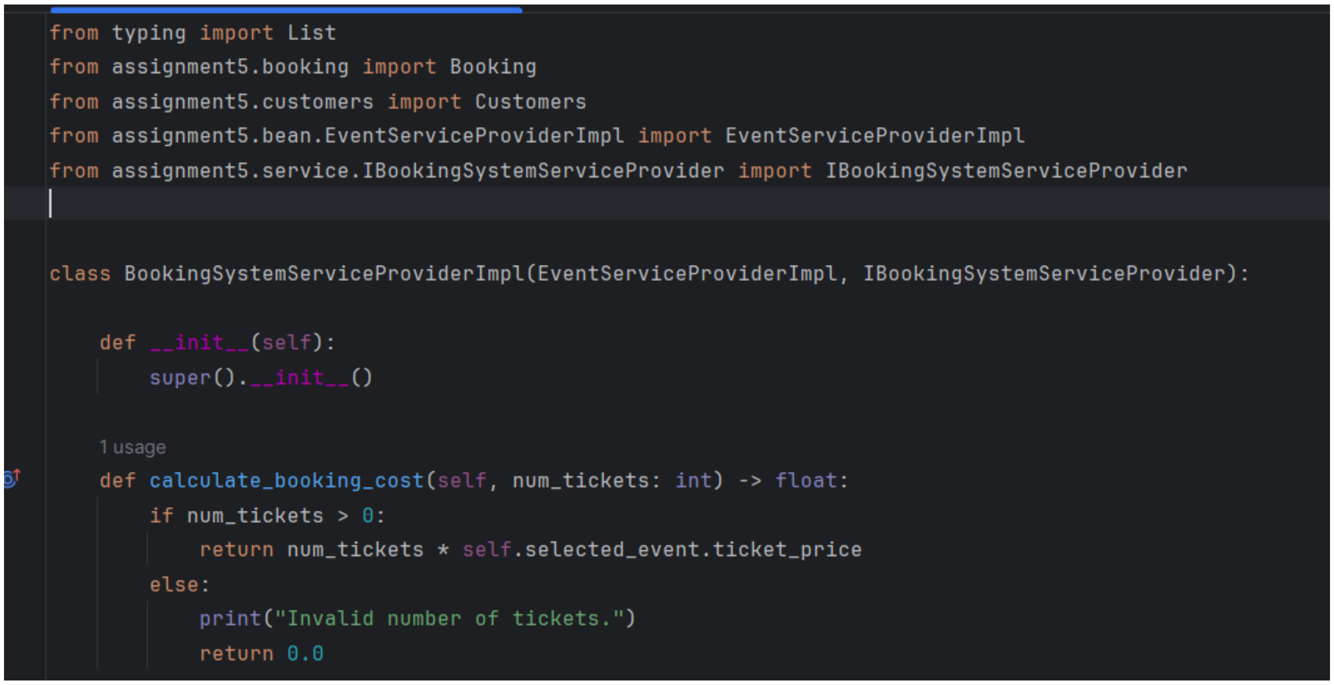


```
from typing import List
from datetime import date, time
from assignment5.event import Event
from assignment5.venue import Venue
from assignment5.service.IEventServiceProvider import IEventServiceProvider

2 usages
class EventServiceProviderImpl(IEventServiceProvider):

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

    def create_event(self, event_name: str, date_str: str, time_str: str, total_seats: int, ticket_price: float,
                     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, event_type)
        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
```

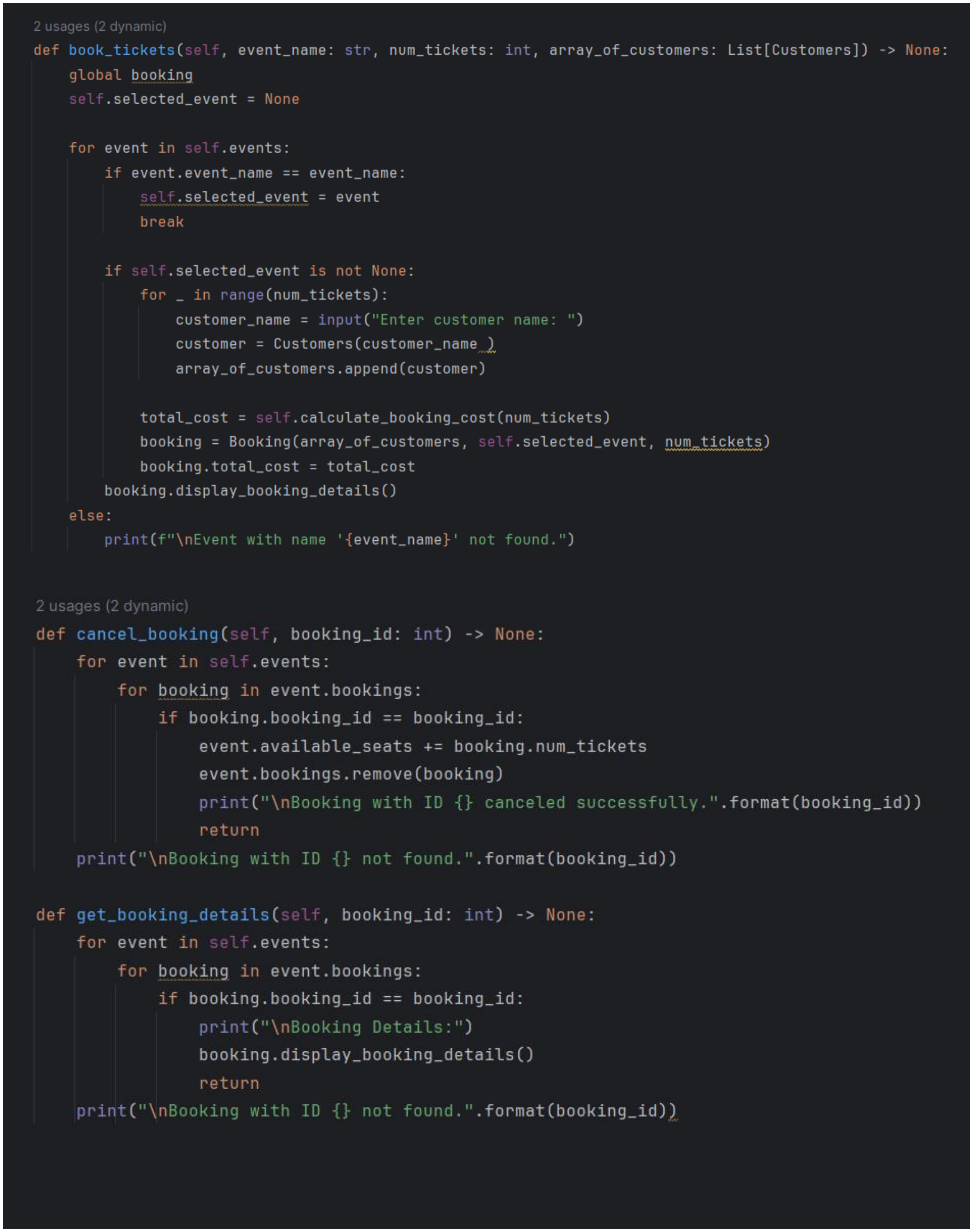


```python
def get_available_no_of_tickets(self) -> int:
    if not self.events:
        return 0
    return self.events[0].available_seats
```

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

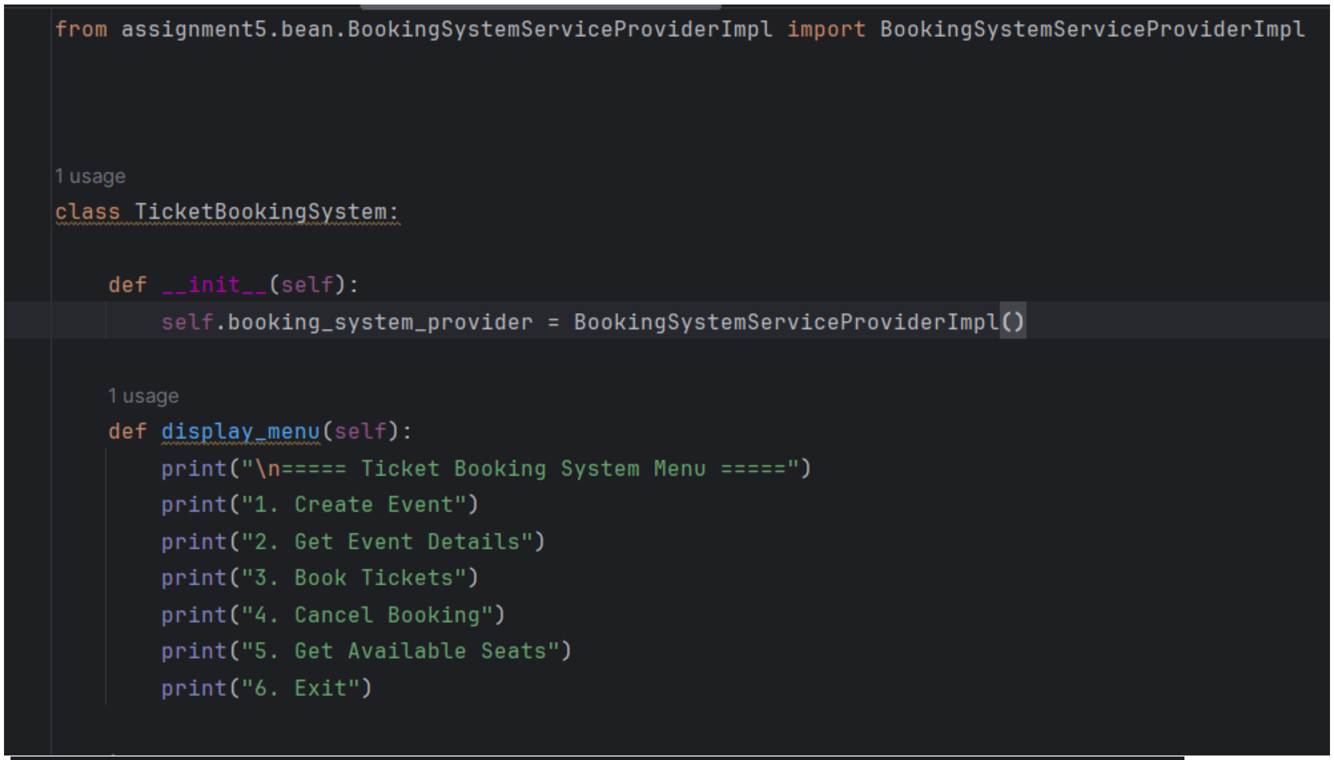


```python
from typing import List
from assignment5.booking import Booking
from assignment5.customers import Customers
from assignment5.bean.EventServiceProviderImpl import EventServiceProviderImpl
from assignment5.service.IBookingSystemServiceProvider import IBookingSystemServiceProvider


class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):

    def __init__(self):
        super().__init__()

    1 usage
    def calculate_booking_cost(self, num_tickets: int) -> float:
        if num_tickets > 0:
            return num_tickets * self.selected_event.ticket_price
        else:
            print("Invalid number of tickets.")
            return 0.0
```

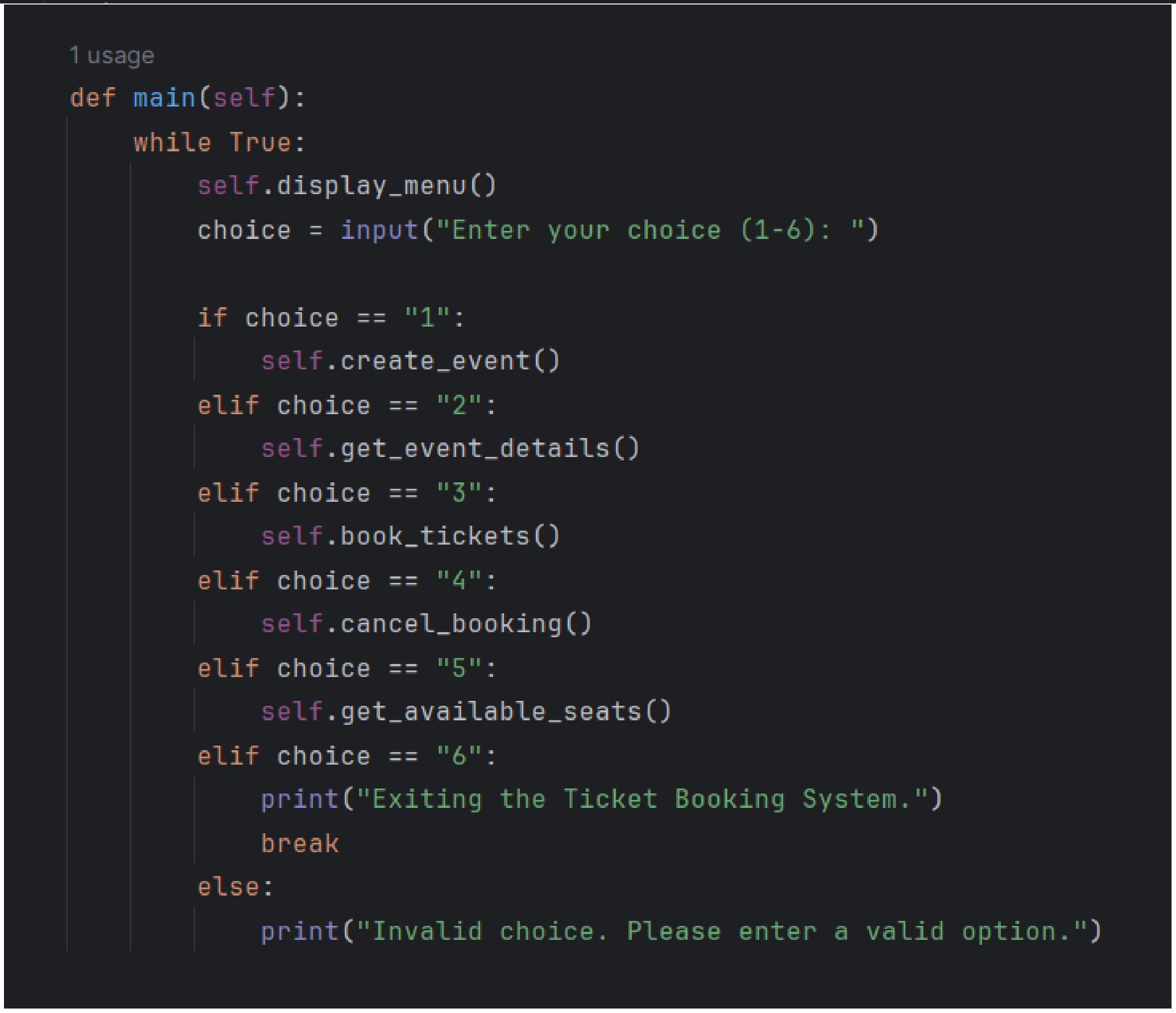


```
2 usages (2 dynamic)
def book_tickets(self, event_name: str, num_tickets: int, array_of_customers: List[Customers]) -> None:
    global booking
    self.selected_event = None

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

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


2 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))
```

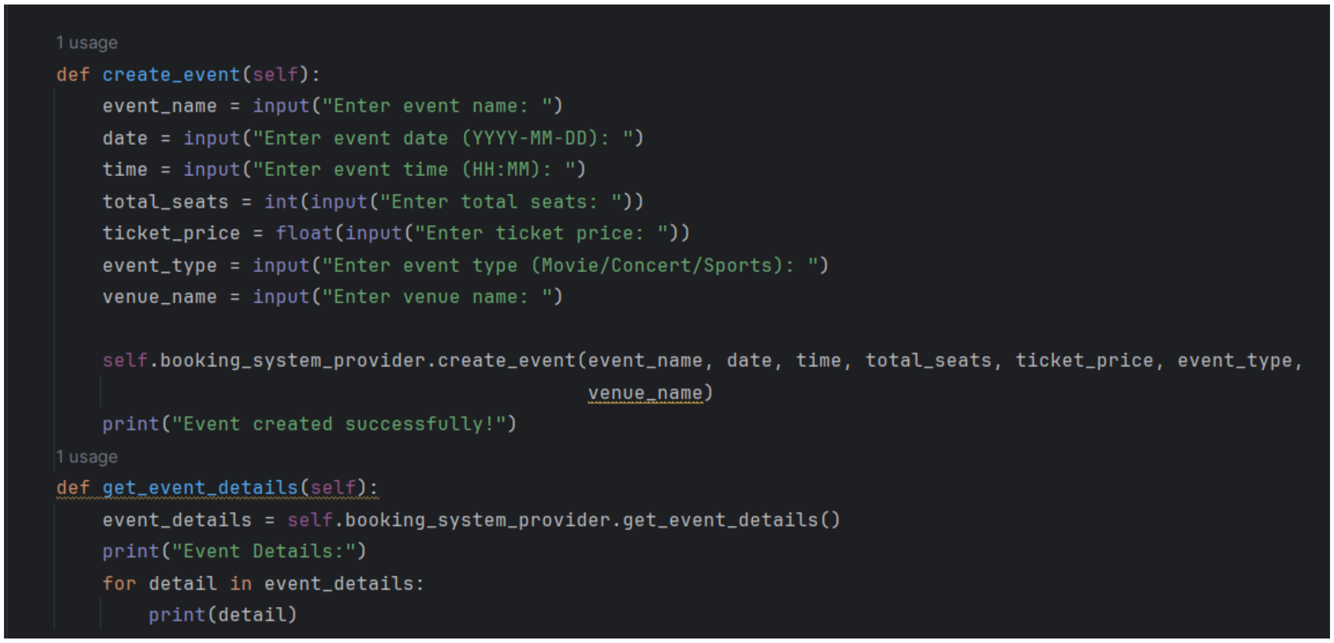
9. Create **TicketBookingSystem** class and perform following operations:
    - Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets",



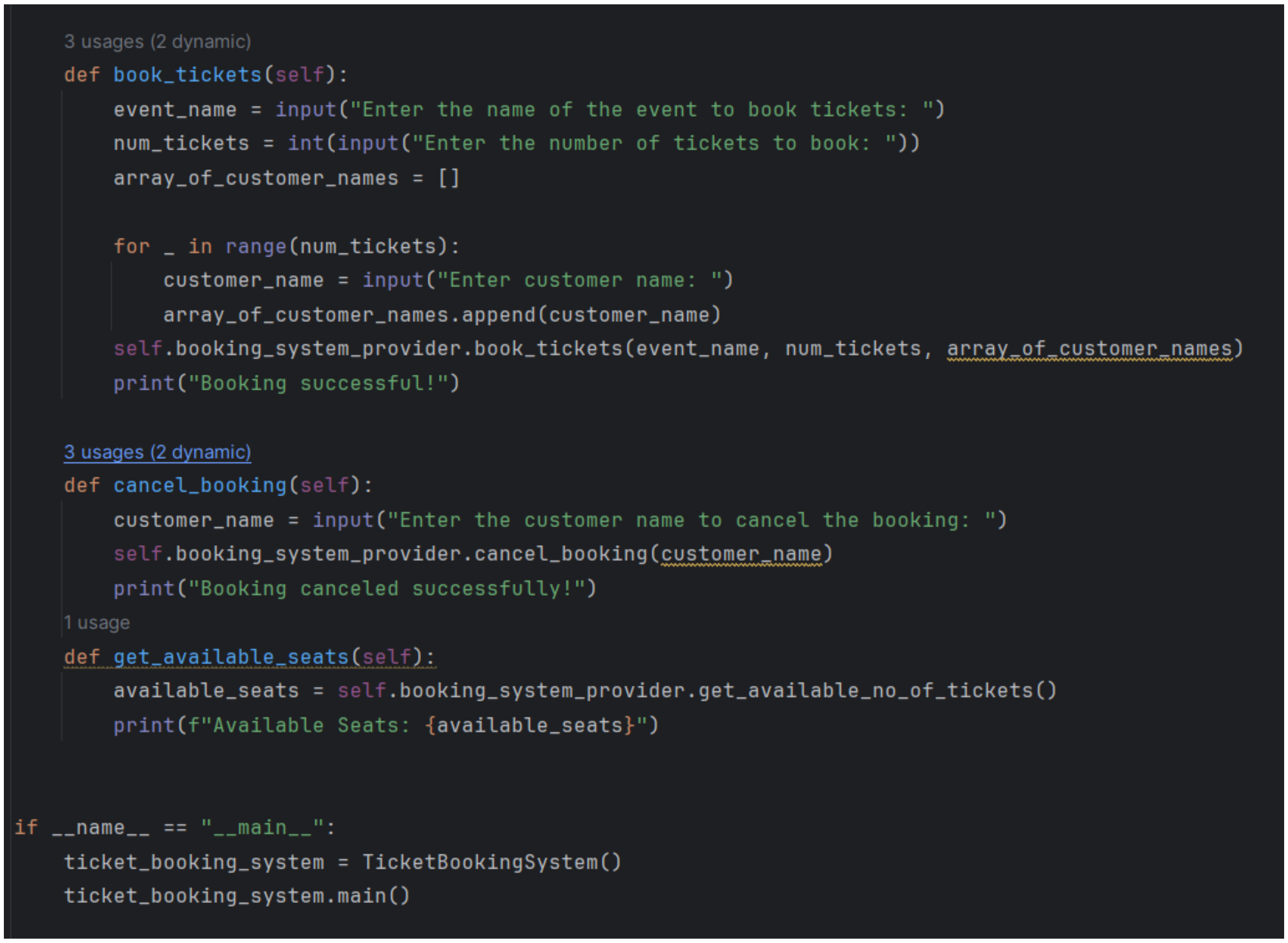
```
from assignment5.bean.BookingSystemServiceProviderImpl import BookingSystemServiceProviderImpl



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): ")

        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): ")
    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):
        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()
```

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

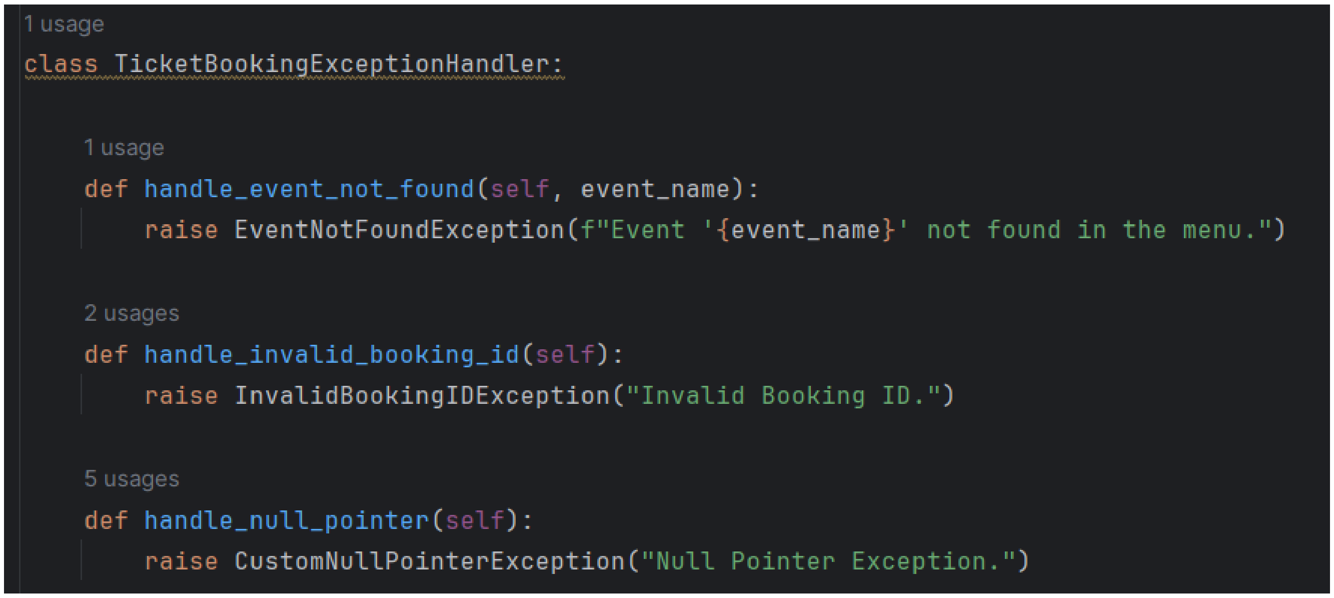
10. Place the interface/abstract class in service package and interface/abstract class implementation class, all concrete class in bean package and **TicketBookingSystem** class in app package.

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

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.



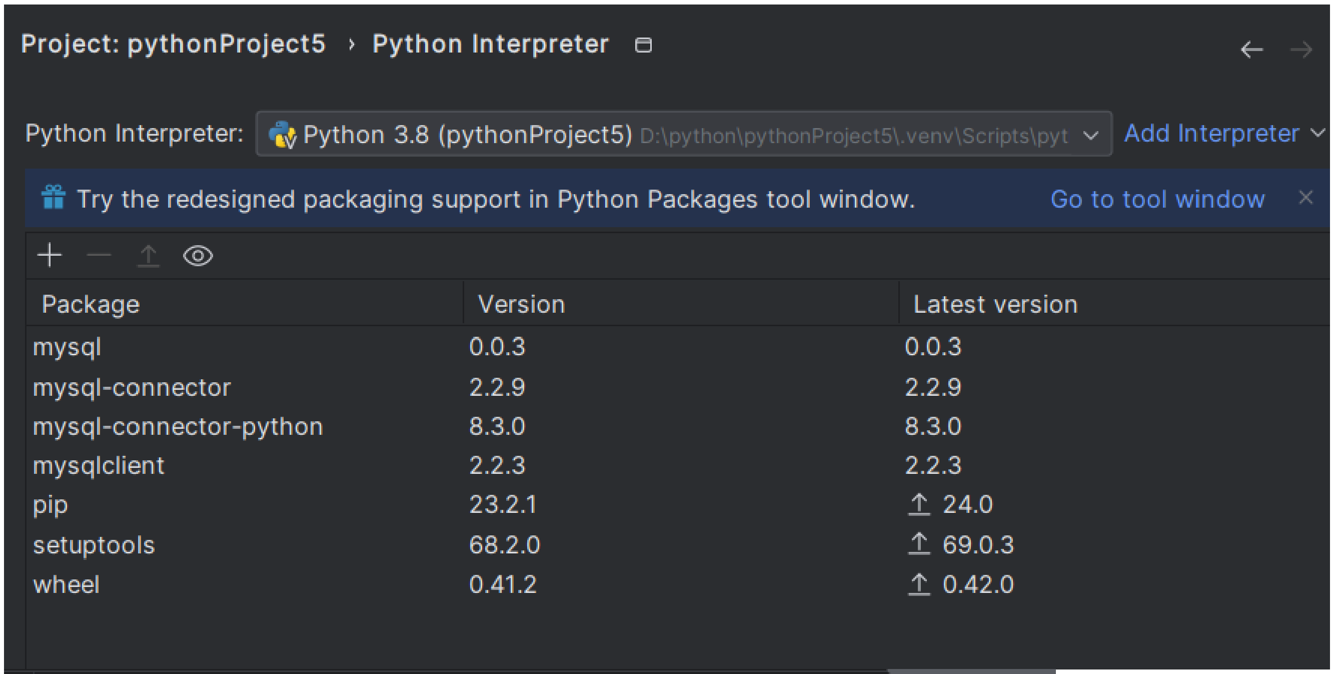
```
1 usage
class TicketBookingExceptionHandler:

    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.")
```

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**.**

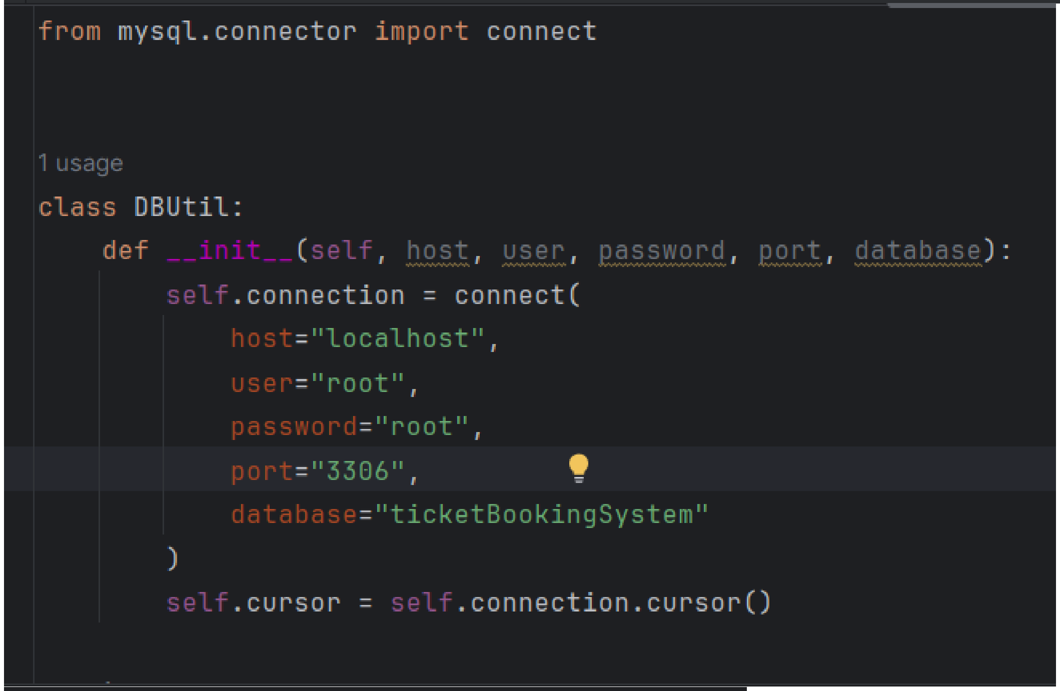
Throw these exceptions from the methods in **TicketBookingSystem** class. Make necessary changes to accommodate exception in the source code. Handle all these exceptions from the main program.

Task 11: Database Connectivity.

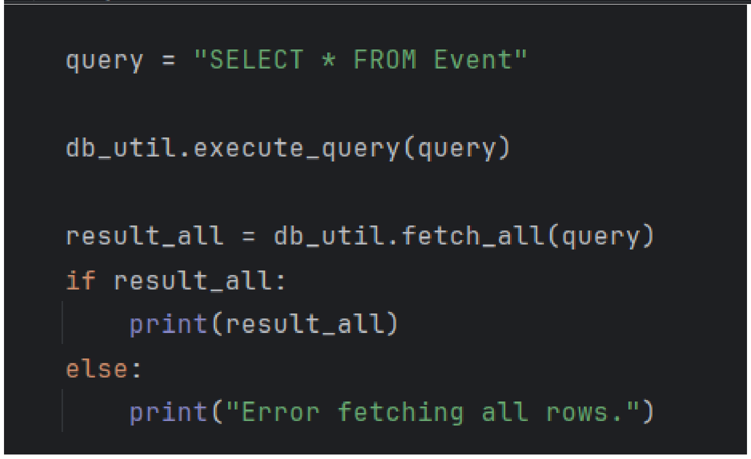
1. Create **DBUtil** class and add the following method.
    - **static getDBConn():Connection** Establish a connection to the database and return Connection reference


Project: pythonProject5 › Python Interpreter
Python Interpreter:
Python 3.8 (pythonProject5) D:\python\pythonProject5\.venv\Scripts\pyt
Add Interpreter
Try the redesigned packaging support in Python Packages tool window.
Go to tool window

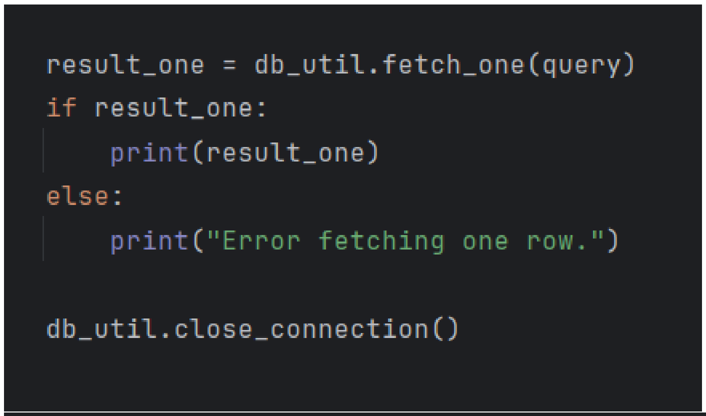
Package	Version	Latest version
mysql	0.0.3	0.0.3
mysql-connector	2.2.9	2.2.9
mysql-connector-python	8.3.0	8.3.0
mysqlclient	2.2.3	2.2.3
pip	23.2.1	↑ 24.0
setuptools	68.2.0	↑ 69.0.3
wheel	0.41.2	↑ 0.42.0



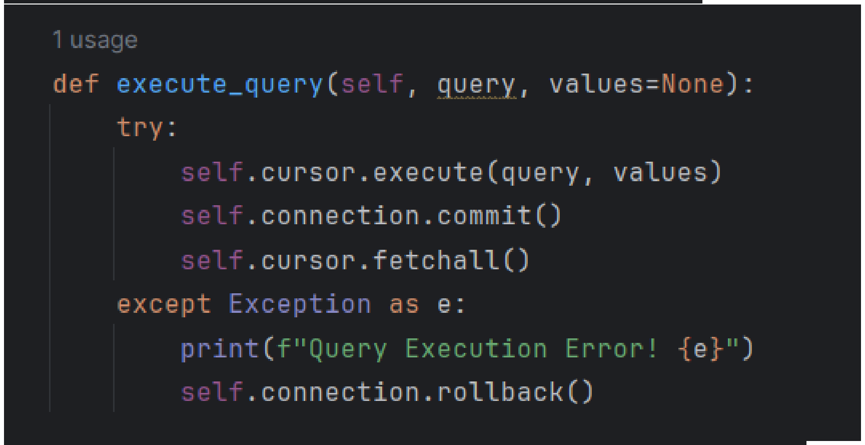
```
from mysql.connector import connect


1 usage
class DBUtil:
    def __init__(self, host, user, password, port, database):
        self.connection = connect(
            host="localhost",
            user="root",
            password="root",
            port="3306",
            database="ticketBookingSystem"
        )
        self.cursor = self.connection.cursor()
```



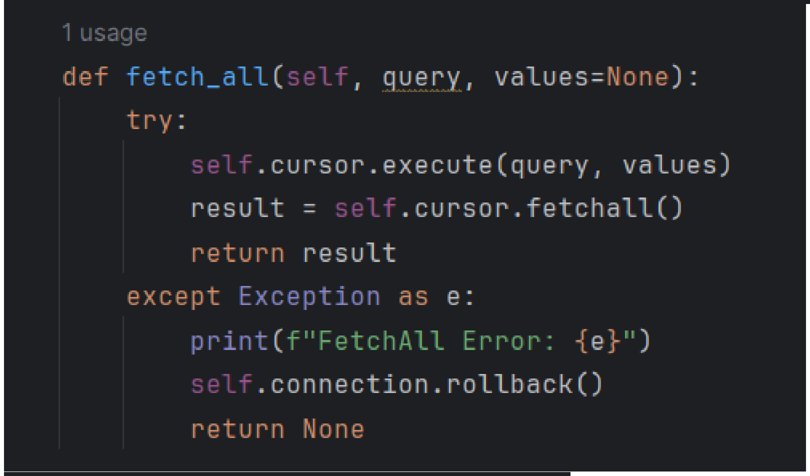
```
query = "SELECT * FROM Event"

db_util.execute_query(query)

result_all = db_util.fetch_all(query)
if result_all:
    print(result_all)
else:
    print("Error fetching all rows.")
```



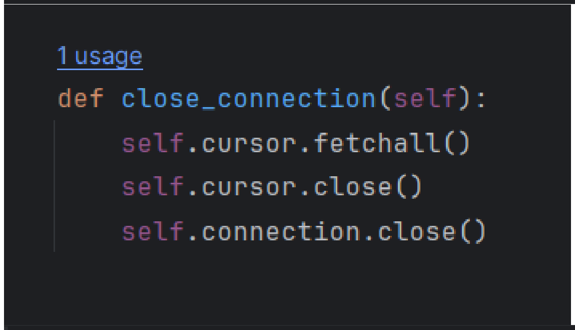
```
result_one = db_util.fetch_one(query)
if result_one:
    print(result_one)
else:
    print("Error fetching one row.")

db_util.close_connection()
```



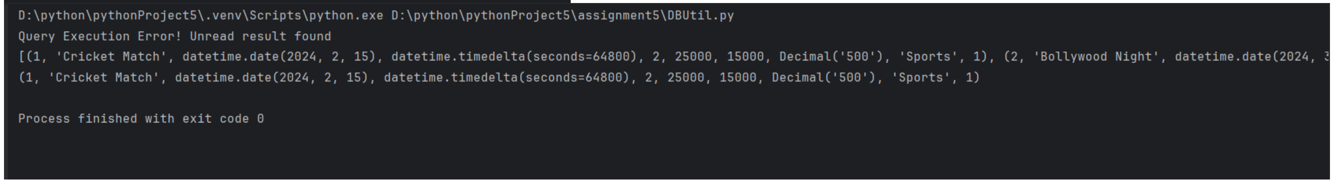
```
1 usage
def execute_query(self, query, values=None):
    try:
        self.cursor.execute(query, values)
        self.connection.commit()
        self.cursor.fetchall()
    except Exception as e:
        print(f"Query Execution Error! {e}")
        self.connection.rollback()
```



```
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 close_connection(self):
    self.cursor.fetchall()
    self.cursor.close()
    self.connection.close()
```



```
D:\python\pythonProject5\.venv\Scripts\python.exe D:\python\pythonProject5\assignment5\DBUtil.py
Query Execution Error! Unread result found
[(1, 'Cricket Match', datetime.date(2024, 2, 15), datetime.timedelta(seconds=64800), 2, 25000, 15000, Decimal('500'), 'Sports', 1), (2, 'Bollywood Night', datetime.date(2024, 3
(1, 'Cricket Match', datetime.date(2024, 2, 15), datetime.timedelta(seconds=64800), 2, 25000, 15000, Decimal('500'), 'Sports', 1)

Process finished with exit code 0
```