TICKET BOOKING SYSTEM

PROJECT INTRODUCTION:
The Ticket Booking System is a console-based application designed to manage and streamline the process of booking tickets for various events such as Movies, Concerts, and Sports. It supports both admin and customer roles, allowing admins to create and manage events, while customers can browse, book, or cancel tickets.Additionally, users can filter and view events by their interest, such as only movies, concerts, or sports.

OBJECTIVE:
- To implement an OOP-based, modular system for managing events and bookings.
- To allow admins to create, update, and view event statistics.
- To allow customers to book, view, and cancel tickets seamlessly.
- To utilize inheritance, interfaces, collections, and exception handling for real-world problem modeling.
- To provide a simple menu-driven interface for interaction**.**

Technologies used:
Language :Python,MySql
IDE: VS CODE

FILE STRUCTURE:
TicketBookingSystem/
|-- dao/
| |-- app/
| | |-- TicketBookingSystem.py
| |-- bean/
| | |-- BookingSystem.py
| | |-- TicketBookingSystem.py
|-- admin.py
|-- entity/model/
| |-- booking.py
| |-- concert.py
| |-- customer.py
| |-- event.py
| |-- movie.py
| |-- sports.py
| |-- venue.py
|-- exception/
| |-- BookingException.py
| |-- EventException.py
| |-- EventNotFoundException.py
| |-- InvalidBookingIDException.py
| |-- NullPointerException.py
|-- main/
| |-- MainModule.py
|-- util/
| |-- db_connection.py
|-- db.sql

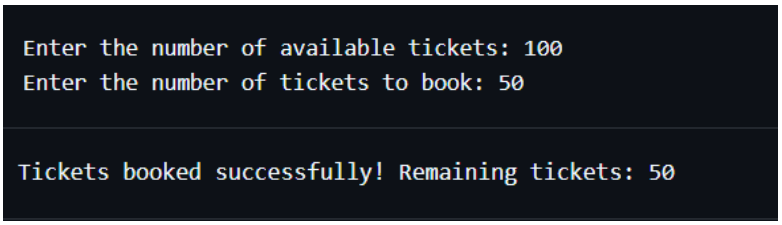
CONTROL STRUCTURE
Task 1: Conditional Statements
Task 2: Nested Conditional Statements
Task 3: Looping
Task 4: Class & Object
Task 5: Inheritance and polymorphism
Task 6: Abstraction
Task 7: Has A Relation / Association
Task 8: Interface/abstract class, and Single Inheritance, static variable
Task 9: Exception Handling
Task 10: Collection
Task 11: Database Connectivity

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.**

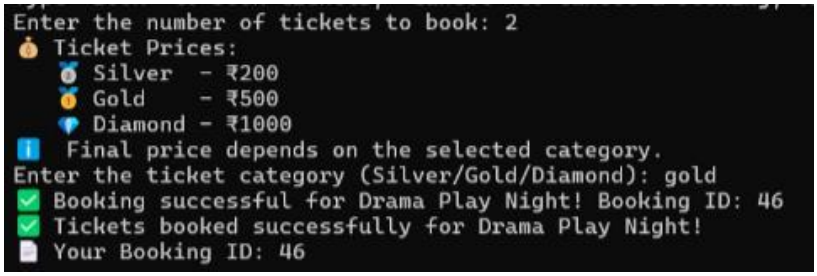
```
def book_tickets(available_tickets, no_of_booking_tickets):
    if available_tickets >= no_of_booking_tickets:
        remaining_tickets = available_tickets - no_of_booking_tickets
        print(f"Tickets booked successfully! Remaining tickets: {remaining_tickets}")
    else:
        print("Tickets unavailable.")

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

```
def calculate_booking_cost(self, num_tickets, ticket_category):
        if ticket_category == "Silver":
```

```
        price_per_ticket = 200
    elif ticket_category == "Gold":
        price_per_ticket = 500
    elif ticket_category == "Diamond":
        price_per_ticket = 1000
    else:
        raise ValueError("Invalid ticket category")

    total_cost = num_tickets * price_per_ticket
    return total_cost
```



Task 3: Looping From the above task book the tickets for repeatedly until user type "Exit"

```
def book_tickets():
    conn = DBConnUtil.get_db_connection()

    system = TicketBookingSystem(conn)

    try:
        print("\n 🔹 Welcome to the Ticket Booking System 🔹 ")
        role = input("Are you an Admin or a Customer? (Admin/Customer/Exit): ").strip().lower()

        if role == "customer":
            # Customer registration/login
            with conn.cursor() as cursor:
                email = input("Enter your email: ").strip()
                cursor.execute("SELECT customer_id FROM Customer WHERE email = %s", (email,))
                result = cursor.fetchone()

                if result:
                    customer_id = result[0]
                    print(f" ✅ Welcome back!")
                else:
                    print(" ❌ No user exists!")
                    print(" 📝 Please register to continue.")
                    customer_name = input("Enter your name: ").strip()
                    phone_number = input("Enter your phone number: ").strip()
                    cursor.execute("""
                        INSERT INTO Customer (customer_name, email, phone_number)
                        VALUES (%s, %s, %s)
                    """, (customer_name, email, phone_number))
                    conn.commit()
                    customer_id = cursor.lastrowid
```

```
            print("✅ Registration successful!")

        while True:
            event_type = input("Which event do you want to book? (Movie/Concert/Sports): ").strip().capitalize()

            if event_type not in ["Movie", "Concert", "Sports"]:
                print("❌ Invalid event type! Please enter 'Movie', 'Concert', or 'Sports'.")
                continue

            with conn.cursor() as cursor:
                cursor.execute("""
                    SELECT event_id, event_name, event_date, event_time
                    FROM Event
                    WHERE event_type = %s
                """, (event_type,))
                events = cursor.fetchall()

            if not events:
                print(f"❌ No {event_type} events available for booking!")
                continue

            print(f"\n🎤 Available {event_type} Events:")
            for event in events:
                print(f" 🔹 {event[0]}. {event[1]} on {event[2]} at {event[3]}")

            event_id = int(input("\nEnter Event ID to proceed with booking: "))
            try:
                event = system.create_event(event_id)
                if event is None or event.event_name is None:
                    raise EventNotFoundException("Invalid Event ID! Please select from the available events.")

                system.display_event_details(event)

                while True:
                    action = input("Type 'Book' to book tickets, 'Cancel' to cancel a booking, or 'Exit' to quit: ").strip().lower()
                    if action == "exit":
                        print("✅ Exiting the booking system. Thank you!")
                        return

                    elif action == "book":
                        num_tickets = int(input("Enter the number of tickets to book: "))
                        print("💰 Ticket Prices:")
                        print("  🥈 Silver  - ₹200")
                        print("  🥇 Gold    - ₹500")
                        print("  💎 Diamond - ₹1000")
                        print("ℹ️ Final price depends on the selected category.")
```

```
                ticket_category = input("Enter the ticket category (Silver/Gold/Diamond): ").strip().capitalize()
                booking_id = system.book_tickets(event_id, customer_id, num_tickets, ticket_category)

                if booking_id:
                    print(f"✅ Tickets booked successfully for {event.event_name}!")
                    print(f"📄 Your Booking ID: {booking_id}")
                else:
                    print("❌ Booking failed! Please try again.")

            elif action == "cancel":
                booking_id = int(input("Enter Booking ID to cancel: "))
                try:
                    system.cancel_booking(booking_id)
                    print("✅ Booking cancelled successfully!")
                except InvalidBookingIDException as e:
                    print(f"❌ {e}")

        except EventNotFoundException as e:
            print(f"❌ {e}")
```

```
Type 'Book' to book tickets, 'Cancel' to cancel a booking, or 'Exit' to quit: book
Enter the number of tickets to book: 2
💰 Ticket Prices:
   🥈 Silver  - ₹200
   🥇 Gold    - ₹500
   💎 Diamond - ₹1000
ℹ️  Final price depends on the selected category.
Enter the ticket category (Silver/Gold/Diamond): diamond
✅ Connection Successful!
✅ Tickets booked successfully for Cultural Dance Show!
📄 Your Booking ID: 44
Type 'Book' to book tickets, 'Cancel' to cancel a booking, or 'Exit' to quit: cancel
Enter Booking ID to cancel: 44
✅ Booking cancelled successfully!
Type 'Book' to book tickets, 'Cancel' to cancel a booking, or 'Exit' to quit: exit
✅ Exiting the booking system. Thank you!
```

Task 4: Class & Object
Create a Following classes with the following attributes and methods:
1. Event Class

Attributes: event_name, event_date, event_time, venue_name, total_seats, available_seats, ticket_price, event_type

Methods and Constructors:
__init__() ,__init__(self, event_name, event_date, event_time, venue_name, total_seats, available_seats, ticket_price, event_type), get_event_name(), set_event_name(event_name), get_event_date(), set_event_date(event_date), get_event_time(), set_event_time(event_time), get_venue_name(), set_venue_name(venue_name), get_total_seats(), set_total_seats(total_seats), get_available_seats(),set_available_seats(available_seats), get_ticket_price(),

set_ticket_price(ticket_price), get_event_type(), set_event_type(event_type), calculate_total_revenue(), get_booked_no_of_tickets(), book_tickets(num_tickets), cancel_booking(num_tickets), display_event_details()

```
class Event():
    def __init__(self, event_id, conn):
        self.event_id = event_id
        self.conn = conn
        self.event_name = None
        self.event_date = None
        self.event_time = None
        self.venue = None
        self.total_seats = 0
        self.available_seats = 0
        self.event_type = None
        self.load_event_details()

    def load_event_details(self):
        with self.conn.cursor() as cursor:
            cursor.execute("""
                    SELECT event_name, event_date, event_time, venue_id, total_seats, available_seats, event_type
                FROM Event WHERE event_id = %s
            """, (self.event_id,))
            result = cursor.fetchone()
            if result:
                (self.event_name, self.event_date, self.event_time,
                 venue_id, self.total_seats, self.available_seats, self.event_type) = result
                self.venue = self.load_venue(venue_id)

    def load_venue(self, venue_id):
        with self.conn.cursor() as cursor:
                cursor.execute("SELECT venue_name, address FROM Venue WHERE venue_id = %s", (venue_id,))
            result = cursor.fetchone()
            if result:
                venue_name, address = result
                return type("Venue", (), {"venue_name": venue_name, "location": address})()
        return None

    @abstractmethod
    def get_event_type(self):
        pass

    def calculate_total_revenue(self):
        with self.conn.cursor() as cursor:
            cursor.execute("""
                SELECT SUM(tc.price * b.no_of_tickets)
                FROM Booking b
                JOIN TicketCategory tc ON b.ticket_category_id = tc.ticket_category_id
```

```
            WHERE b.event_id = %s
        """, (self.event_id,))
        result = cursor.fetchone()
        return result[0] if result and result[0] else 0

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

    def book_tickets(self, customer_id, num_tickets, ticket_category):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            total_cost = self.calculate_total_cost(ticket_category, num_tickets)
            try:
                with self.conn.cursor() as cursor:
                    cursor.execute("""
                            INSERT INTO Booking (customer_id, event_id, num_tickets, ticket_category,
total_cost, booking_date)
                        VALUES (%s, %s, %s, %s, %s, %s)
                    """, (customer_id, self.event_id, num_tickets, ticket_category, total_cost, datetime.now()))
                    cursor.execute("""
                        UPDATE Event
                        SET available_seats = %s
                        WHERE event_id = %s
                    """, (self.available_seats, self.event_id))
                    self.conn.commit()
                    booking_id = cursor.lastrowid
                    if booking_id == 0:
                        cursor.execute("SELECT LAST_INSERT_ID()")
                        booking_id = cursor.fetchone()[0]
                    print(f"✅ Booking successful for {self.event_name}! Booking ID: {booking_id}")
                    return booking_id
            except Exception as e:
                print(f"❌ Booking failed due to: {e}")
                self.conn.rollback()
        else:
            print("❌ Not enough available seats!")
        return None

    def calculate_total_cost(self, ticket_category, num_tickets):
        price_mapping = {"Silver": 200, "Gold": 500, "Diamond": 1000}
        return price_mapping.get(ticket_category, 0) * num_tickets

    def cancel_booking(self, booking_id):
        with self.conn.cursor() as cursor:
                cursor.execute("SELECT num_tickets FROM Booking WHERE booking_id = %s",
(booking_id,))
            result = cursor.fetchone()
            if result:
                num_tickets = result[0]
                self.available_seats += num_tickets
```

```
            cursor.execute("DELETE FROM Booking WHERE booking_id = %s", (booking_id,))
            cursor.execute("""
                UPDATE Event
                SET available_seats = %s
                WHERE event_id = %s
            """, (self.available_seats, self.event_id))
            self.conn.commit()
            print(f"✅ Booking ID {booking_id} cancelled successfully!")
        else:
            print(f"❌ Booking ID {booking_id} not found!")

    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}")
        if self.venue:
            print(f"Venue: {self.venue.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Event Type: {self.event_type}")
```

2. Venue Class

Attributes:venue_name**,** address

Methods and Constructors:
__init__(), __init__(self, venue_name, address), get_venue_name(), set_venue_name(venue_name), get_address(), set_address(address), display_venue_details()

```
class Venue:
    def __init__(self, venue_name=None, address=None):
        self.venue_name = venue_name
        self.address = address

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

    def get_venue_name(self):
        return self.venue_name

    def set_venue_name(self, venue_name):
        self.venue_name = venue_name

    def get_address(self):
        return self.address

    def set_address(self, address):
        self.address = address
```

3. Customer Class

Attributes: customer_name, email
- phone_number

Methods and Constructors:
__init__(), __init__(self, customer_name, email, phone_number), get_customer_name(), set_customer_name(customer_name), get_email(), set_email(email), get_phone_number(), set_phone_number(phone_number), display_customer_details()

```
class Customer:
    def __init__(self, customer_name=None, email=None, phone_number=None):
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number

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

    def get_customer_name(self):
        return self.customer_name

    def set_customer_name(self, customer_name):
        self.customer_name = customer_name

    def get_email(self):
        return self.email

    def set_email(self, email):
        self.email = email

    def get_phone_number(self):
        return self.phone_number

    def set_phone_number(self, phone_number):
        self.phone_number = phone_number
```

4. Booking Class

Attributes:
booking_id, customer, event, num_tickets, total_cost, booking_date

Methods and Constructors:
__init__(), __init__(self, customer, event, num_tickets), get_booking_id(), get_customer(), set_customer(customer), get_event(), set_event(event), get_num_tickets(), set_num_tickets(num_tickets), get_total_cost(), set_total_cost(total_cost), get_booking_date(),

set_booking_date(booking_date), calculate_booking_cost(num_tickets), book_tickets(num_tickets), cancel_booking(num_tickets), get_available_no_of_tickets(), get_event_details()

```
from datetime import datetime
from util.db_connection import DBConnUtil
from exception.NullPointerException import NullPointerException
class Booking:
    def __init__(self, customer=None, event=None, num_tickets=0, ticket_category=None):
        self.booking_id = self.generate_booking_id()
        self.customer = customer
        self.event = event
        self.num_tickets = num_tickets
        self.ticket_category = ticket_category
        self.total_cost = self.calculate_total_cost()
        self.booking_date = datetime.now()

    def generate_booking_id(self):
        conn = DBConnUtil.get_db_connection()
        if conn is None:
            raise NullPointerException("Database connection is null")
        with conn.cursor() as cursor:
            cursor.execute("SELECT MAX(booking_id) FROM Booking")
            result = cursor.fetchone()
            max_booking_id = result[0] if result[0] is not None else 0
            return max_booking_id + 1

    def calculate_total_cost(self):
        price_mapping = {"Silver": 200, "Gold": 500, "Diamond": 1000}
        return price_mapping.get(self.ticket_category, 0) * self.num_tickets

    def display_booking_details(self):
        print(f"Booking ID: {self.booking_id}")
        print(f"Customer Name: {self.customer.customer_name}")
        print(f"Event Name: {self.event.event_name}")
        print(f"Number of Tickets: {self.num_tickets}")
        print(f"Ticket Category: {self.ticket_category}")
        print(f"Total Cost: {self.total_cost}")
        print(f"Booking Date: {self.booking_date}")

    def get_booking_id(self):
        return self.booking_id

    def get_customer(self):
        return self.customer

    def set_customer(self, customer):
        self.customer = customer

    def get_event(self):
        return self.event
```

```
    def set_event(self, event):
        self.event = event

    def get_num_tickets(self):
        return self.num_tickets

    def set_num_tickets(self, num_tickets):
        self.num_tickets = num_tickets

    def get_total_cost(self):
        return self.total_cost

    def set_total_cost(self, total_cost):
        self.total_cost = total_cost

    def get_booking_date(self):
        return self.booking_date

    def set_booking_date(self, booking_date):
        self.booking_date = booking_date
```

Task 5: Inheritance and Polymorphism – Summary

1. Subclass: Movie (inherits Event)

- **Attributes:**
  genre, actor_name, actress_name
- **Methods:**
  - **Default and parameterized constructor**
  - **Getters and setters**
  - **display_event_details(): Shows movie info including genre**

2. Subclass: Concert (inherits Event)

- **Attributes:**
  artist, type (e.g., Theatrical, Classical)
- **Methods:**
  - **Default and parameterized constructor**
  - **Getters and setters**
  - **display_concert_details(): Shows concert and artist details**

3. Subclass: Sports (inherits Event)

- **Attributes:**
  sport_name, teams_name (e.g., India vs Pakistan)
- **Methods:**
  - **Default and parameterized constructor**
  - **Getters and setters**
  - **display_sport_details(): Shows match and team info**

TicketBookingSystem Class

- **Methods:**
  - **create_event(...): Creates and returns appropriate subclass instance (Movie/Concert/Sports)**

- **display_event_details(event): Calls event.display_event_details() → Polymorphism**
- **book_tickets(event, num_tickets): Books tickets if available, returns cost**
- **cancel_tickets(event, num_tickets): Cancels given number of tickets**
- **main():**
  - **Menu to choose event type and view/book/cancel**
  - **Demonstrates polymorphism by calling display_event_details() without knowing event type**

Movie.py

```
from entity.model.event import Event

class Movie(Event):
    def __init__(self, event_id, conn):
        super().__init__(event_id, conn)
        self.genre = None
        self.actor_name = None
        self.actress_name = None
        self.load_movie_details()

    def load_movie_details(self):
        with self.conn.cursor() as cursor:
            cursor.execute("SELECT genre, actor_name, actress_name FROM Movie WHERE event_id
= %s", (self.event_id,))
            result = cursor.fetchone()
            if result:
                self.genre, self.actor_name, self.actress_name = result

    def display_event_details(self):
        super().display_event_details()
        print(f"🎬 Genre: {self.genre}")
        print(f"🎭 Actors: {self.actor_name} & {self.actress_name}\n")

    def get_event_type(self):
        return "Movie"
```

Concert.py

```
from entity.model.event import Event

class Concert(Event):
    def __init__(self, event_id, conn):
        super().__init__(event_id, conn)
        self.artist = None
        self.concert_type = None
        self.load_concert_details()

    def load_concert_details(self):
        with self.conn.cursor() as cursor:
```

```
            cursor.execute("SELECT artist, concert_type FROM Concert WHERE event_id = %s",
(self.event_id,))
            result = cursor.fetchone()
            if result:
                self.artist, self.concert_type = result

    def display_event_details(self):
        super().display_event_details()
        print(f"🎤 Artist: {self.artist}")
        print(f"🎵 Concert Type: {self.concert_type}\n")

    def get_event_type(self):
        return "Concert"
```

Sport.py

```
from entity.model.event import Event

class Sports(Event):
    def __init__(self, event_id, conn):
        super().__init__(event_id, conn)
        self.sport_name = None
        self.teams_name = None
        self.load_sports_details()

    def load_sports_details(self):
        with self.conn.cursor() as cursor:
            cursor.execute("SELECT sport_name, teams_name FROM Sports WHERE event_id = %s",
(self.event_id,))
            result = cursor.fetchone()
            if result:
                self.sport_name, self.teams_name = result

    def display_event_details(self):
        super().display_event_details()
        print(f"⚽ Sport: {self.sport_name}")
        print(f"🏆 Teams: {self.teams_name}\n")

    def get_event_type(self):
        return "Sports"
```

TicketBookingSystem.py

```
class TicketBookingSystem(BookingSystem):
    def __init__(self, conn=None):
        if conn is None:
            raise NullPointerException("Database connection is null")
        self.conn = conn
        self.events = []
        self.bookings = []
```

```
    def create_event(self, event_id):
        if self.conn is None:
            raise NullPointerException("Database connection is null")

        try:
            with self.conn.cursor() as cursor:
                cursor.execute("SELECT event_name, event_date, venue_id, total_seats, event_type FROM
Event WHERE event_id = %s", (event_id,))
                result = cursor.fetchone()

            if result:
                event_name, event_date, venue_id, total_seats, event_type = result
                if event_type.lower() == "movie":
                    event = Movie(event_id, self.conn)
                elif event_type.lower() == "concert":
                    event = Concert(event_id, self.conn)
                elif event_type.lower() == "sports":
                    event = Sports(event_id, self.conn)
                else:
                    event = None

                if event:
                    self.events.append(event)
                    return event
            else:
                raise EventNotFoundException(f"Event with ID {event_id} not found.")
        except Exception as e:
            print(f"Error creating event: {e}")
        return None

    def display_event_details(self, event):
        event.display_event_details()

    def book_tickets(self, event_id, customer_id, num_tickets, ticket_category):
        try:
            event = next((e for e in self.events if e.event_id == event_id), None)
            if not event:
                event = self.create_event(event_id)
            if event and event.available_seats >= num_tickets:
                total_cost = self.calculate_booking_cost(num_tickets, ticket_category)
                customer = Customer(customer_id)
                booking = Booking(customer, event, num_tickets, ticket_category)
                self.bookings.append(booking)
                event.available_seats -= num_tickets

                # Update database
                with self.conn.cursor() as cursor:
                    cursor.execute("""
```

```
                    INSERT INTO Booking (booking_id, customer_id, event_id, num_tickets, ticket_category, total_cost, booking_date)
                    VALUES (%s, %s, %s, %s, %s, %s, %s)
                """, (booking.booking_id, customer_id, event_id, num_tickets, ticket_category, total_cost, booking.booking_date))
                    cursor.execute("UPDATE Event SET available_seats = %s WHERE event_id = %s", (event.available_seats, event_id))
                self.conn.commit()

            return booking.booking_id
        else:
                raise EventNotFoundException(f"Event with ID {event_id} not found or not enough available seats!")
    except Exception as e:
        print(f"Error booking tickets: {e}")
        self.conn.rollback()
        return None

def cancel_booking(self, booking_id):
    try:
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if booking:
            booking.event.available_seats += booking.num_tickets
            self.bookings.remove(booking)

            # Update database
            with self.conn.cursor() as cursor:
                cursor.execute("DELETE FROM Booking WHERE booking_id = %s", (booking_id,))
                    cursor.execute("UPDATE Event SET available_seats = %s WHERE event_id = %s", (booking.event.available_seats, booking.event.event_id))
                self.conn.commit()

            return booking.booking_id
        else:
            raise InvalidBookingIDException(f"Booking with ID {booking_id} not found!")
    except Exception as e:
        print(f"Error canceling booking: {e}")
        self.conn.rollback()
        return None

def get_available_seats(self, event_id):
    try:
        event = next((e for e in self.events if e.event_id == event_id), None)
        if not event:
            event = self.create_event(event_id)
        if event:
            return event.available_seats
        else:
            raise EventNotFoundException(f"Event with ID {event_id} not found!")
    except Exception as e:
```

```
            print(f"Error getting available seats: {e}")
            return None

    def calculate_booking_cost(self, num_tickets, ticket_category):
        price_mapping = {"Silver": 200, "Gold": 500, "Diamond": 1000}
        return num_tickets * price_mapping.get(ticket_category, 0)

    def get_booking_details(self, booking_id):
        try:
            booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
            if booking:
                return booking.display_booking_details()
            else:
                raise InvalidBookingIDException(f"Booking with ID {booking_id} not found!")
        except Exception as e:
            print(f"Error getting booking details: {e}")
            return None
```

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.**

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."

The abstract event class code is given below task 5. The inherited concrete class such as Movie, Concert, and Sports implementation is also given below task5

Task 7: Has-A Relationship – Ticket Booking System
1. Venue Class

- **Attributes: venue_name, address**

- **Methods:**
  - **Constructors (default & overloaded)**
  - **display_venue_details()**
  - **Getters/Setters**

2. Event Class (Base Class)

- **Attributes: event_name, event_date, event_time, venue (Venue), total_seats, available_seats, ticket_price, event_type**
- **Methods:**
  - **Constructors (default & overloaded)**
  - **calculate_total_revenue()**
  - **getBookedNoOfTickets()**
  - **book_tickets(num_tickets)**
  - **cancel_booking(num_tickets)**
  - **display_event_details()**

3. Subclasses of Event

- **Movie, Concert, Sport – Inherit and override from Event**

4. Customer Class

- **Attributes: customer_name, email, phone_number**
- **Methods:**
  - **Constructors (default & overloaded)**
  - **display_customer_details()**
  - **Getters/Setters**

5. Booking Class

- **Attributes: bookingId, customers[], event, num_tickets, total_cost, booking_date**
- **Methods:**
  - **Constructors (default & overloaded)**
  - **display_booking_details()**
  - **Getters/Setters**

6. BookingSystem Class

- **Attributes: events[]**
- **Methods:**
  - **create_event(...)**
  - **calculate_booking_cost(num_tickets)**
  - **book_tickets(event_name, num_tickets, arrayOfCustomer)**
  - **cancel_booking(booking_id)**
  - **getAvailableNoOfTickets()**
  - **getEventDetails()**

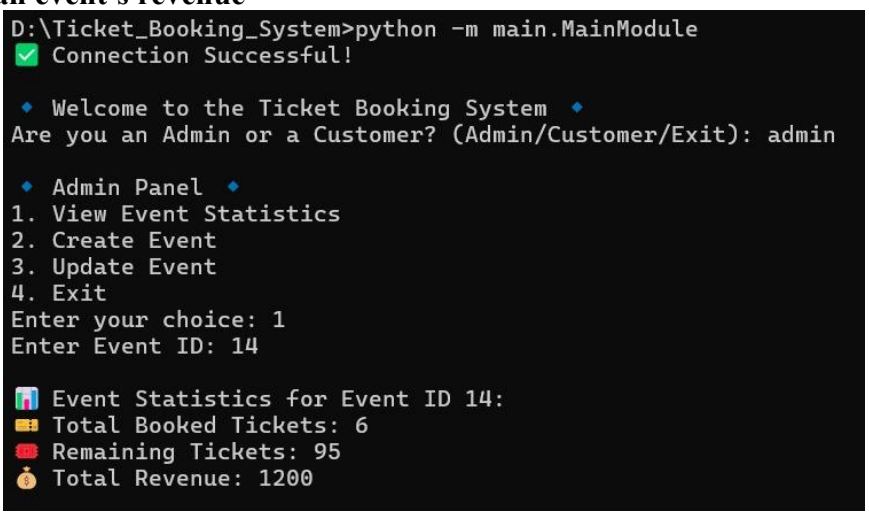
```
class Admin:
    def __init__(self, conn):
        self.conn = conn

    def create_event(self, event_name, event_date, event_time, venue_id, total_seats, event_type):
        try:
            with self.conn.cursor() as cursor:
                # Insert into Event table
                cursor.execute("""
                    INSERT INTO Event (event_name, event_date, event_time, venue_id, total_seats, available_seats, event_type)
                    VALUES (%s, %s, %s, %s, %s, %s, %s)
```

```
            """, (event_name, event_date, event_time, venue_id, total_seats, total_seats, event_type))

            event_id = cursor.lastrowid

            # Prompt for additional info based on event type
            if event_type.lower() == 'movie':
                genre = input("🎬 Enter Genre: ")
                actor_name = input("🎭 Enter Actor Name: ")
                actress_name = input("🎭 Enter Actress Name: ")
                cursor.execute("""
                    INSERT INTO Movie (event_id, genre, actor_name, actress_name)
                    VALUES (%s, %s, %s, %s)
                """, (event_id, genre, actor_name, actress_name))

            elif event_type.lower() == 'concert':
                artist = input("🎤 Enter Artist Name: ")
                concert_type = input("🎵 Enter Concert Type (e.g., Solo, Band, Festival): ")
                cursor.execute("""
                    INSERT INTO Concert (event_id, artist, concert_type)
                    VALUES (%s, %s, %s)
                """, (event_id, artist, concert_type))

            elif event_type.lower() == 'sports':
                sport_name = input("🏅 Enter Sport Name: ")
                teams_name = input("🏆 Enter Teams (e.g., Team A vs Team B): ")
                cursor.execute("""
                    INSERT INTO Sports (event_id, sport_name, teams_name)
                    VALUES (%s, %s, %s)
                """, (event_id, sport_name, teams_name))

            self.conn.commit()
            print(f"\n✅ Event '{event_name}' created successfully with ID {event_id}!")

        except Exception as e:
            self.conn.rollback()
            print(f"❌ Failed to create event: {e}")

    def update_event(self, event_id, event_name, event_date, event_time, venue_id, total_seats, available_seats):
        """Updates event details in Event table and synchronizes changes with Movie, Concert, or Sports tables"""
        with self.conn.cursor() as cursor:
            cursor.execute("""
                UPDATE Event
                SET event_name = %s, event_date = %s, event_time = %s,
                    venue_id = %s, total_seats = %s, available_seats = %s
                WHERE event_id = %s
            """, (event_name, event_date, event_time, venue_id, total_seats, available_seats, event_id))
```

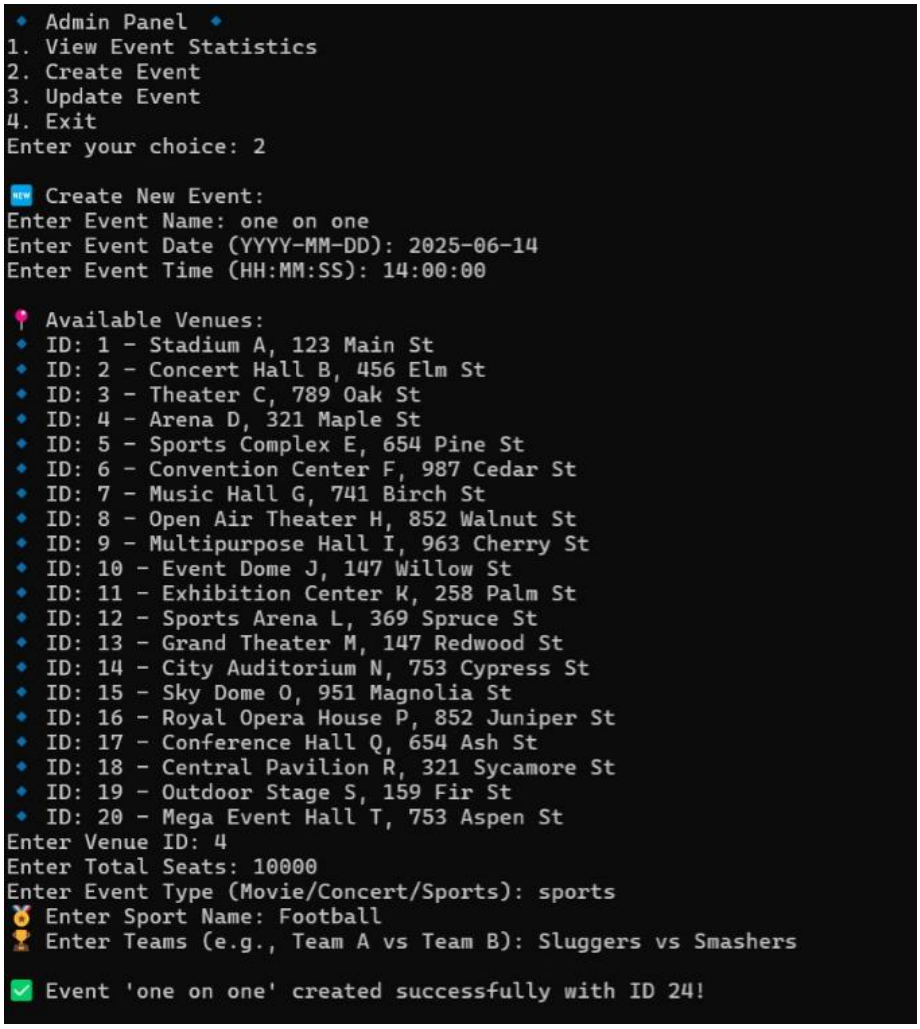
```
        cursor.execute("SELECT event_type FROM Event WHERE event_id = %s", (event_id,))
        result = cursor.fetchone()

        if result:
            event_type = result[0].lower()

            if event_type == "movie":
                genre = input("Enter genre: ")
                actor_name = input("Enter actor name: ")
                actress_name = input("Enter actress name: ")

                cursor.execute("""
                    UPDATE Movie
                    SET genre = %s, actor_name = %s, actress_name = %s
                    WHERE event_id = %s
                """, (genre, actor_name, actress_name, event_id))

            elif event_type == "concert":
                artist = input("Enter artist name: ")
                concert_type = input("Enter concert type: ")

                cursor.execute("""
                    UPDATE Concert
                    SET artist = %s, concert_type = %s
                    WHERE event_id = %s
                """, (artist, concert_type, event_id))

            elif event_type == "sports":
                sport_name = input("Enter sport name: ")
                teams_name = input("Enter new teams: ")

                cursor.execute("""
                    UPDATE Sports
                    SET sport_name = %s, teams_name = %s
                    WHERE event_id = %s
                """, (sport_name, teams_name, event_id))

    self.conn.commit()
    print(f"✅ Event ID {event_id} updated successfully!")

def calculate_total_revenue(self, event_id):
    """Calculates total revenue based on ticket sales"""
    price_mapping = {"Silver": 200, "Gold": 500, "Diamond": 1000}
    with self.conn.cursor() as cursor:
        cursor.execute("""
            SELECT ticket_category, SUM(num_tickets)
            FROM Booking
            WHERE event_id = %s
            GROUP BY ticket_category
        """, (event_id,))
```

```
        total_revenue = sum(price_mapping[category] * num_tickets for category, num_tickets in cursor.fetchall())
        return total_revenue

    def get_event_statistics(self, event_id):
        """Returns total booked tickets, remaining tickets, and total revenue"""
        with self.conn.cursor() as cursor:
            cursor.execute("SELECT SUM(num_tickets) FROM Booking WHERE event_id = %s", (event_id,))
            booked_tickets = cursor.fetchone()[0] or 0

            cursor.execute("SELECT available_seats FROM Event WHERE event_id = %s", (event_id,))
            remaining_tickets = cursor.fetchone()[0] or 0

            total_revenue = self.calculate_total_revenue(event_id)
        return booked_tickets, remaining_tickets, total_revenue
```

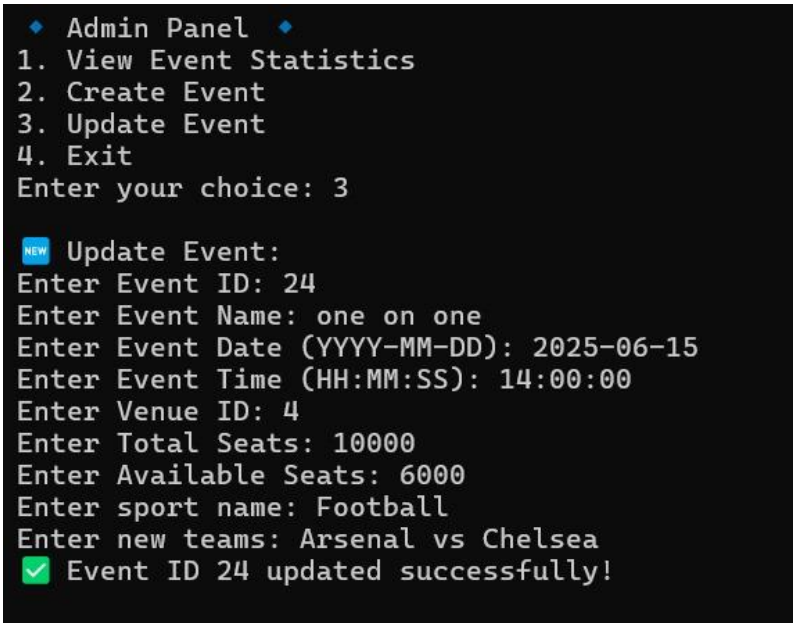
- **To view an event's revenue**



- **To create a new event by checking the available venues**



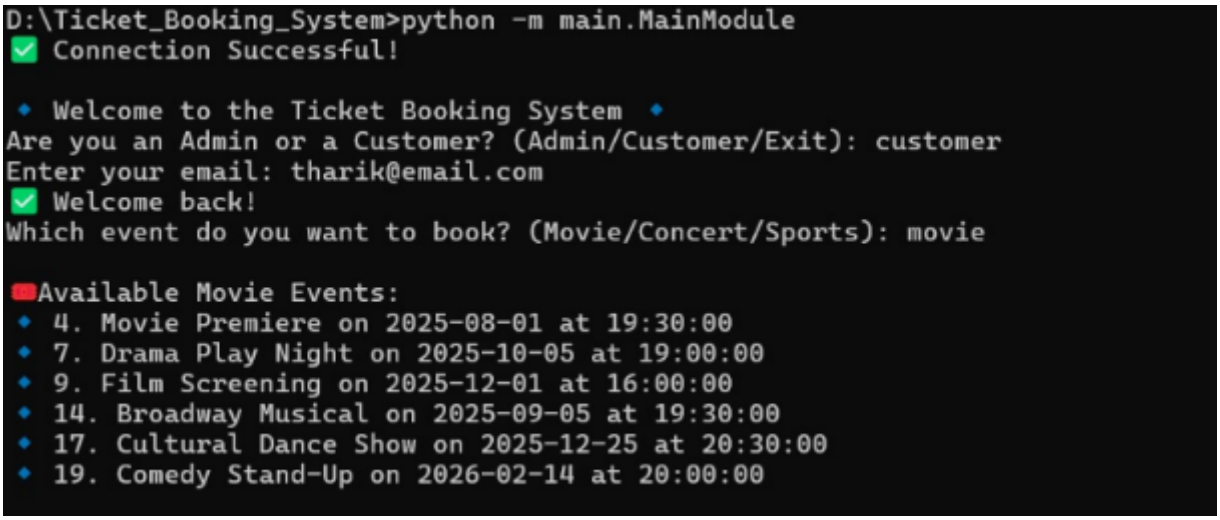
- **To update an existing event**



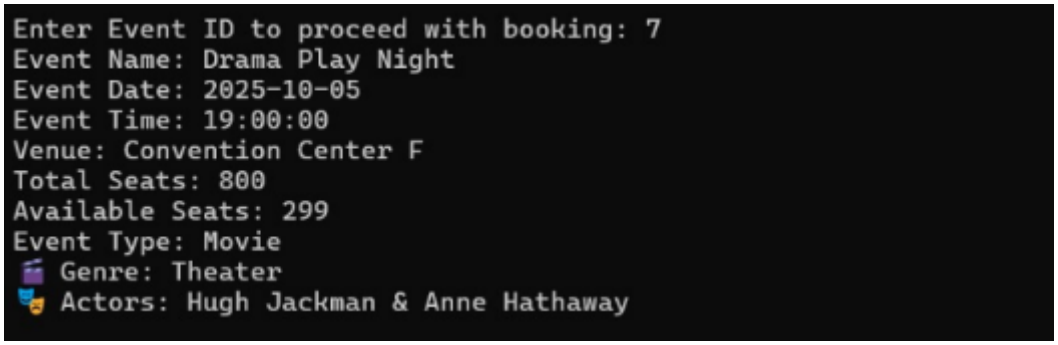
Task 8: Ticket Booking System – Interface, Inheritance, Static, Abstract

1. **Create Venue class (as per Task 4).**
2. **Create Event class:**
    - **Attributes: event_name, event_date, event_time, venue, total_seats, available_seats, ticket_price, event_type**
    - **Methods: Default + parameterized constructor, getters/setters, print method.**
3. **Create Event subclasses: Movie, Concert, Sport.**
4. **Create Customer and Booking classes (as per Task 4).**
5. **Interface EventService (in service package):**
    - **create_event(...) → Event**
    - **getEventDetails() → Event[]**
    - **getAvailableNoOfTickets() → int**
6. **Interface IBookingSystemServiceProvider (in service package):**
    - **calculate_booking_cost(num_tickets)**
    - **book_tickets(event_name, num_tickets, customers[])**
    - **cancel_booking(booking_id)**
    - **get_booking_details(booking_id)**
7. **Create TicketBookingSystem (in bean package): Implements EventService.**
8. **Create BookingSystemServiceProviderImpl (in bean package): Implements BookingSystem, inherits TicketBookingSystem.**
    - **Attribute: Array of Events**
9. **Create TicketBookingSystem (in app package):**
    - **Console-based UI to handle commands: create_event, book_tickets, cancel_tickets, get_available_seats, get_event_details, exit.**

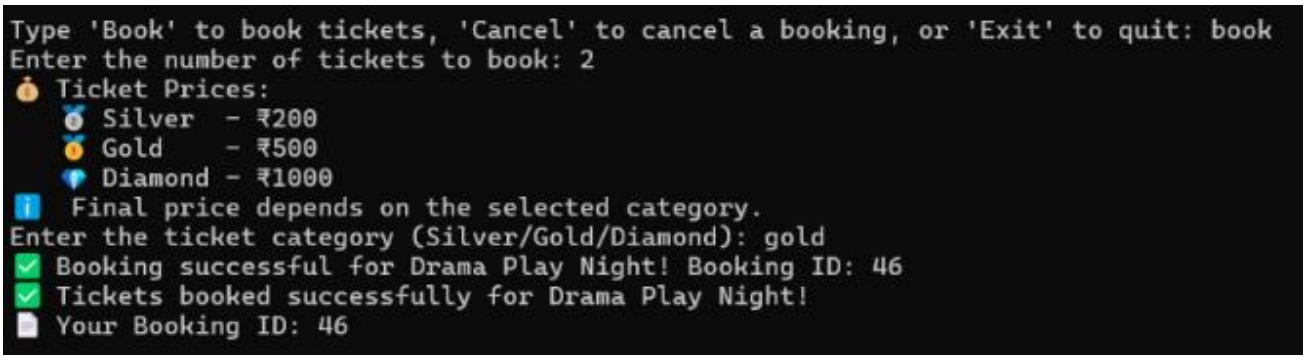
- **Displays the options that are available as per user interest**



- **The user can choose the event they like and proceed to book ticket**



- **The user is prompt to book,cancel or exit .Once the user proceeds with booking, they're given choice of choosing between different categories like "silver","gold", or "diamond"**

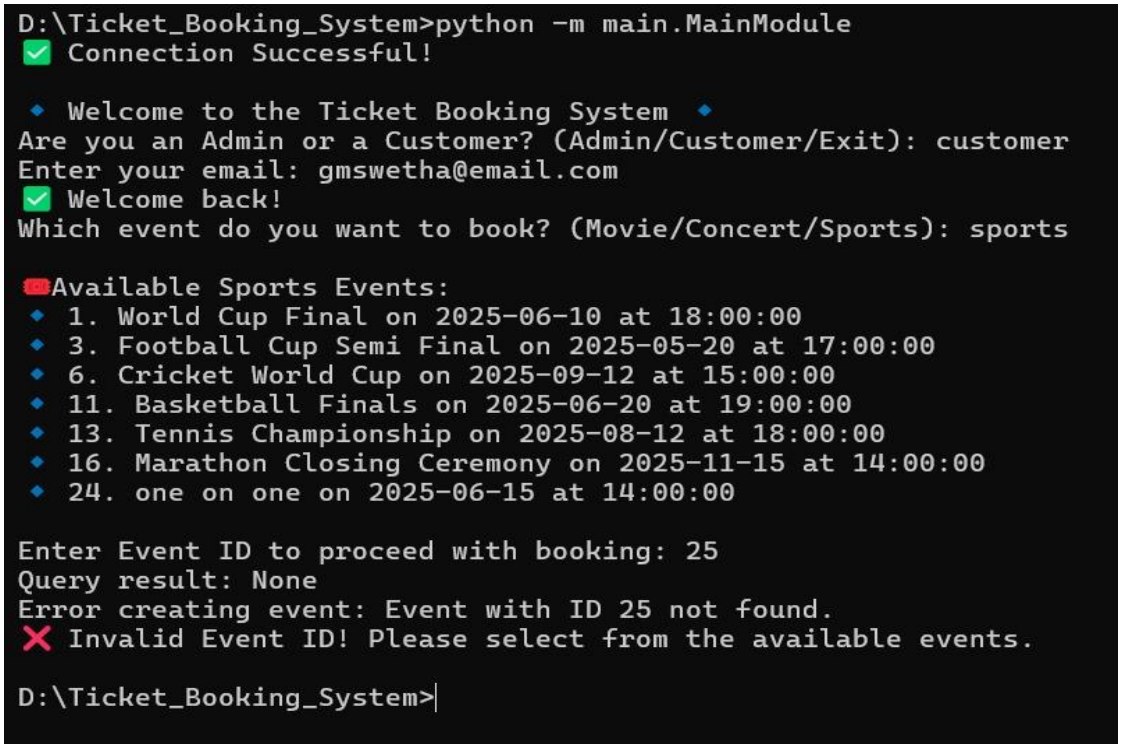


- **If the user wants to cancel their booking , they can use their booking_id which is generated at the timing of booking confirmation to cancel booking**

Task 9 Exception Handling

- **Custom Exceptions:**
  - **EventNotFoundException: Thrown if event not found during booking.**
  - **InvalidBookingIDException: Thrown for invalid booking ID during view/cancel.**
- **NullPointerException: Handle in main() method.**
- **Throw exceptions in TicketBookingSystem methods and handle all in main program.**

Below is the demonstration of EventNotFoundException



Task 10: Collection

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

2. From the previous task change all list type of attribute to type Set in Booking and BookingSystem

class and perform the same operation.

• Avoid adding duplicate Account object to the set.

• Create Comparator<Event> object to sort the event based on event name and location in alphabetical order.

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

Entity\model\booking.py

```
class Booking:
    booking_counter = 0

    def __init__(self, customers=None, event=None, num_tickets=0):
        Booking.booking_counter += 1
        self.booking_id = Booking.booking_counter
        self.customers = customers if customers else {}
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_booking_cost()
        self.booking_date = datetime.now()

    def calculate_booking_cost(self):
        return self.num_tickets * self.event.ticket_price
```

```
    def get_booking_id(self):
        return self.booking_id

    def get_customers(self):
        return self.customers

    def set_customers(self, customers):
        self.customers = customers

    def get_event(self):
        return self.event

    def set_event(self, event):
        self.event = event

    def get_num_tickets(self):
        return self.num_tickets

    def set_num_tickets(self, num_tickets):
        self.num_tickets = num_tickets

    def get_total_cost(self):
        return self.total_cost

    def set_total_cost(self, total_cost):
        self.total_cost = total_cost

    def get_booking_date(self):
        return self.booking_date

    def set_booking_date(self, booking_date):
        self.booking_date = booking_date

    def display_booking_details(self):
        print(f"Booking ID: {self.booking_id}")
        print(f"Booking Date: {self.booking_date}")
        print(f"Event: {self.event.event_name}")
        print(f"Number of Tickets: {self.num_tickets}")
        print(f"Total Cost: {self.total_cost}")
        print("Customers:")
        for key, customer in self.customers.items():
            print(f"Customer ID: {key}")
            customer.display_customer_details()
```

dao\app\TicketBookingSystem.py

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

```
    def create_event(self, event_name, event_date, event_time, total_seats, ticket_price, event_type, venue_name, **kwargs):
        venue = Venue(venue_name, kwargs.get('address', ''))
        if event_type.lower() == 'movie':
            event = Movie(event_name, event_date, event_time, venue, total_seats, total_seats, ticket_price, event_type, kwargs.get('genre', ''), kwargs.get('actor_name', ''), kwargs.get('actress_name', ''))
        elif event_type.lower() == 'concert':
            event = Concert(event_name, event_date, event_time, venue, total_seats, total_seats, ticket_price, event_type, kwargs.get('artist', ''), kwargs.get('concert_type', ''))
        elif event_type.lower() == 'sports':
            event = Sports(event_name, event_date, event_time, venue, total_seats, total_seats, ticket_price, event_type, kwargs.get('sport_name', ''), kwargs.get('teams_name', ''))
        else:
            raise ValueError("Invalid event type")
        self.events[event_name] = event
        return event

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

    def book_tickets(self, event_name, num_tickets, array_of_customers):
        event = self.events.get(event_name)
        if event:
            if event.book_tickets(num_tickets):
                customers_dict = {i: customer for i, customer in enumerate(array_of_customers)}
                booking = Booking(customers_dict, event, num_tickets)
                return booking
            else:
                print("Tickets unavailable.")
                return None
        else:
            print("Event not found.")
            return None

    def cancel_booking(self, booking):
        booking.event.cancel_booking(booking.num_tickets)
        print(f"Cancelled booking with ID: {booking.booking_id}")

    def get_available_no_of_tickets(self, event_name):
        event = self.events.get(event_name)
        if event:
            return event.available_seats
        else:
            print("Event not found.")
            return None

    def get_event_details(self, event_name):
        event = self.events.get(event_name)
        if event:
```

```
            event.display_event_details()
        else:
            print("Event not found.")
```

Task 11: Database Connectivity

The database connectivity is established through method get_db_connection() by providing details like host, user, password, database name. The connection established is closed at the end of process to prevent data loss

```
import mysql.connector

class DBConnUtil:
    @staticmethod
    def get_db_connection():
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",
                password="root",
                database="TicketBookingSystem"
            )
            if conn.is_connected():
                print("✅ Connection Successful!")
            return conn
        except mysql.connector.Error as err:
            print(f"❌ Error: {err}")
            return None

# Run the connection check
if __name__ == "__main__":
    conn = DBConnUtil.get_db_connection()
    if conn:
        conn.close()
```