TICKET BOOKING SYSTEM

1.Project Overview:

The Ticket Booking System is a project designed to demonstrate the integration of SQL database operations with Object-Oriented Programming (OOP) principles. The system allows users to interact with a movie ticket booking service, perform CRUD operations, and manage ticket records effectively.

2.Objectives:

- To implement a ticket booking system using SQL and OOP concepts.
- To facilitate the insertion, display, update, and deletion of booking records.
- To demonstrate interaction between Java (or equivalent OOP language) and SQL databases.

3.Approach and Implementation:

OOP Concepts Used:

The project uses **classes and objects** to model the system.

Key classes might include:

- Event
- Customer
- Venue
- Booking
- Movie that inherits from Event
- Sports that inherits from Event
- Concert that inherits from Event
- TicketBookingSystem

Encapsulation is used to bundle data (like customer info, ticket details) with methods that operate on them.

Constructors (__init__) are used to initialize object data.

Python Functionality:

- User input is taken through the command line to simulate ticket booking operations using main() method.

Methods are created for:

- Booking a ticket
- Canceling a booking
- Creating a event
- Getting Booking Summary
- Getting customer booked maximum ticket
- Viewing Event Details
- Viewing available tickets or bookings

Database Integration:

- The project connects Python with an SQL database using a module like mysql.connector or Pymysql

Tables are used to store data such as:

- User details
- Ticket information
- Booking history

SQL queries are used for:

- Inserting new bookings
- Deleting/canceling bookings
- Fetching and displaying records

Workflow:

- The system starts by connecting to the database.
- The user is prompted to choose an operation (book, cancel, view).
- Based on input, the respective method is triggered.
- Updates are made in both the object and the database to reflect changes.

4.Functional Modules

1.Database setup:

Event Table:

```
mysql> select * from event;
+----------+------------+------------+------------+----------+-------------+-----------------+--------------+------------+------------+
| event_id | event_name | event_date | event_time | venue_id | total_seats | available_seats | ticket_price | event_type | booking_id |
+----------+------------+------------+------------+----------+-------------+-----------------+--------------+------------+------------+
|        1 | rock cup   | 2025-01-01 | 09:00:00   |        1 |       10050 |              50 |       500.05 | concert    |        110 |
|        2 | worldcup   | 2025-01-02 | 09:00:00   |        2 |       10100 |             100 |       500.05 | sports     |        120 |
|        3 | moonlight  | 2025-01-03 | 09:30:00   |        3 |       10050 |               0 |      1500.09 | concert    |        130 |
|        4 | twilt      | 2025-01-04 | 10:00:00   |        4 |       15500 |             200 |       200.05 | movie      |        140 |
|        5 | football   | 2025-01-01 | 10:00:00   |        5 |       10500 |               0 |       700.05 | sports     |        150 |
|        6 | rocky      | 2025-01-05 | 11:10:00   |        6 |       20500 |              30 |      5000.05 | concert    |        160 |
|        7 | popz       | 2025-02-01 | 10:30:00   |        7 |       10050 |              50 |       300.05 | movie      |        170 |
|        8 | cricket    | 2025-01-03 | 09:30:00   |        8 |       18500 |             100 |      2000.09 | sports     |        180 |
|        9 | fire       | 2025-01-04 | 08:30:00   |        9 |       10250 |               0 |       200.05 | movie      |        190 |
|       10 | tennis     | 2025-01-05 | 10:00:00   |       10 |       10020 |              20 |      1000.00 | sports     |        200 |
+----------+------------+------------+------------+----------+-------------+-----------------+--------------+------------+------------+
10 rows in set (0.01 sec)
```

Venue Table:

```
mysql> SELECT * from venue;
+----------+------------+----------+
| venue_id | venue_name | address  |
+----------+------------+----------+
|        1 | casagrand  | chennai  |
|        2 | pears      | mumbai   |
|        3 | laxi       | kolkata  |
|        4 | royal      | chennai  |
|        5 | popi       | salem    |
|        6 | tres       | goa      |
|        7 | keer       | ranchi   |
|        8 | pears      | chennai  |
|        9 | tulip      | banglore |
|       10 | bivec      | chennai  |
+----------+------------+----------+
10 rows in set (0.00 sec)
```

Customer Table:

```
mysql> select * from customer;
+-------------+---------------+---------+--------------+------------+
| customer_id | customer_name | email   | phone_number | booking_id |
+-------------+---------------+---------+--------------+------------+
|         101 | ramya         | gmail   |    123444000 |        110 |
|         102 | saru          | gmail   |       908767 |        120 |
|         103 | kaviya        | yahoo   |       896457 |        130 |
|         104 | priya         | outlook |       892456 |        140 |
|         105 | sam           | yahoo   |       123097 |        150 |
|         106 | teja          | outlook |      7456000 |        160 |
|         107 | geetha        | gmail   |      3444000 |        170 |
|         108 | sai           | email   |        90674 |        180 |
|         109 | sara          | yahoo   |       784563 |        190 |
|         110 | eucha         | outlook |       781245 |        200 |
+-------------+---------------+---------+--------------+------------+
10 rows in set (0.00 sec)
```

Booking Table:

```
mysql> select * from booking;
+------------+-------------+----------+-------------+------------+--------------+
| booking_id | customer_id | event_id | num_tickets | total_cost | booking_date |
+------------+-------------+----------+-------------+------------+--------------+
|        110 |         101 |        1 |           2 |    1000.05 | 2025-02-23   |
|        120 |         101 |        2 |           1 |     300.05 | 2025-02-24   |
|        130 |         103 |        3 |           2 |    1000.05 | 2025-02-20   |
|        140 |         104 |        4 |           5 |    6000.00 | 2024-11-12   |
|        150 |         105 |        5 |           2 |     300.45 | 2024-10-10   |
|        160 |         106 |        6 |           4 |    1200.90 | 2025-01-02   |
|        170 |         104 |        7 |           3 |     800.90 | 2025-02-01   |
|        180 |         108 |        8 |           1 |     500.00 | 2024-12-04   |
|        190 |         109 |        9 |          16 |    2000.09 | 2025-01-10   |
|        200 |         110 |       10 |           0 |     300.05 | 2024-11-11   |
+------------+-------------+----------+-------------+------------+--------------+
10 rows in set (0.00 sec)
```

2.Python Functional Modules:

TBS_EXAMPLE/

```
├── dao/
│   ├── BookingSystemRepositoryImpl.py
│   ├── IBookingSystemRepository.py
│
├── entity/
│   ├── Booking.py
│   ├── Concert.py
│   ├── Customer.py
│   ├── Event.py
│   ├── Movie.py
│   ├── Sport.py
│   ├── Venue.py
│
├── exception/
│   ├── EventNotFoundException.py
│   ├── InvalidBookingIDException.py
│
├── util/
│   ├── DBUtil.py
│
├── mainmodule.py
```

Venue.py:

```
class Venue:
    def __init__(self, venue_name="", address=""):
        self._venue_name = venue_name
        self._address = address
    @property
    def venue_name(self):
        return self._venue_name
   @venue_name.setter
    def venue_name(self, value):
        self._venue_name = value
# Getter for address
    @property
    def address(self):
        return self._address
    # Setter for address
    @address.setter
    def address(self, value):
        self._address = value
    def display_venue_details(self):
        print(f"Venue: {self.venue_name}, Address: {self.address}")
```

Event.py:

```
from abc import ABC, abstractmethod

class Event(ABC):

    def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, event_type):

        self._event_name = event_name

        self._event_date = event_date

        self._event_time = event_time

        self._venue = venue

        self._total_seats = total_seats

        self._available_seats = total_seats  # Initially equal to total_seats

        self._ticket_price = ticket_price

        self._event_type = event_type

    @property

    def event_name(self):

        return self._event_name

    @event_name.setter

    def event_name(self, value):

        self._event_name = value

    @property

    def event_date(self):

        return self._event_date
```

```
    @event_date.setter
    def event_date(self, value):
        self._event_date = value
    @property
    def event_time(self):
        return self._event_time
    @event_time.setter
    def event_time(self, value):
        self._event_time = value
    @property
    def venue(self):
        return self._venue
    @venue.setter
    def venue(self, value):
        self._venue = value
    @property
    def total_seats(self):
        return self._total_seats
    @total_seats.setter
    def total_seats(self, value):
        self._total_seats = value
        self._available_seats = value  # Optionally reset available seats
```

```
    @property
    def available_seats(self):
        return self._available_seats
    @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
    @property
    def event_type(self):
        return self._event_type
@event_type.setter
    def event_type(self, value):
        self._event_type = value
@abstractmethod
    def display_event_info(self):
        """Abstract method to display event type, date, and time"""
            pass
```

Subclasses:

Movie.py:

```
from entity.Event import Event

class Movie(Event):

   def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, genre, actor_name, actress_name):

        super().__init__(event_name, event_date, event_time, venue, total_seats,ticket_price, "Movie")

        self.genre = genre

        self.actor_name = actor_name

        self.actress_name = actress_name

    def display_event_info(self):

        print(f"Event Type: {self.event_type}")

        print(f"Event Date: {self.event_date}")

        print(f"Event Time: {self.event_time}")
```

Sports.py:

```
from entity.Event import Event

class Sport(Event):

   def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, sport_name, teams_name):

        super().__init__(event_name, event_date, event_time, venue, total_seats, ticket_price, "Sport")

        self.sport_name = sport_name

        self.teams_name = teams_name
```

```
    def display_event_info(self):
        print(f"Event Type: {self.event_type}")
        print(f"Event Date: {self.event_date}")
        print(f"Event Time: {self.event_time}")
```

Concert.py:

```
from entity.Event import Event

class Concert(Event):
    def __init__(self, event_name, event_date, event_time, venue, total_seats, ticket_price, artist, concert_type):
        super().__init__(event_name, event_date, event_time, venue, total_seats, ticket_price, "Concert")
        self.artist = artist
        self.concert_type = concert_type
    def display_event_info(self):
        print(f"Event Type: {self.event_type}")
        print(f"Event Date: {self.event_date}")
        print(f"Event Time: {self.event_time}")
```

Customer.py:

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

```
@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 phone_number(self):
    return self._phone_number

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

Booking.py:

```
class Booking:
    booking_counter = 1
    def __init__(self, customers, event, num_tickets):
        self._bookingId = Booking.booking_counter
        Booking.booking_counter += 1
        self._customers = customers
        self._event = event
        self._num_tickets = num_tickets
        self._total_cost = event.ticket_price * num_tickets
    @property
    def bookingId(self):
        return self._bookingId
    @property
    def customers(self):
        return self._customers

    @customers.setter
    def customers(self, value):
        self._customers = value

    @property
    def event(self)
```

```
return self._event


    @event.setter
    def event(self, value):
        self._event = value
        self._total_cost = value.ticket_price * self._num_tickets  # Update cost if event changes


    @property
    def num_tickets(self):
        return self._num_tickets


    @num_tickets.setter
    def num_tickets(self, value):
        self._num_tickets = value
        self._total_cost = self._event.ticket_price * value  # Update total cost on ticket change
   @property
    def total_cost(self):
         return self._total_cost
```

IBookingSystemRepository.py:

```
from abc import ABC, abstractmethod

class IBookingSystemRepository(ABC):

    @abstractmethod

    def create_event(self, event_name, date, time, total_seats, ticket_price,
event_type, venue):

        pass


    @abstractmethod

    def getEventDetails(self):

        pass


    @abstractmethod

    def getAvailableNoOfTickets(self):

        pass


    @abstractmethod

    def calculate_booking_cost(self, num_tickets, ticket_price):

        pass


    @abstractmethod

    def book_tickets(self, eventname, num_tickets):

        pass
```

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

    @abstractmethod
    def get_booking_details(self, booking_id):
        pass

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

BookingSystemRepositoryImpl.py:

```
import pymysql
from dao.IBookingSystemRepository import IBookingSystemRepository
from util.DBUtil import DBUtil
from exception.EventNotFoundException import EventNotFoundException
from exception.InvalidBookingIDException import InvalidBookingIDException
from tabulate import tabulate

class BookingSystemRepositoryImpl(IBookingSystemRepository):
    def __init__(self):
```

```
        self.conn = DBUtil.getDBConn()
        self.cursor = self.conn.cursor()

    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue):
        sql = """INSERT INTO event (event_name, event_date, event_time, total_seats, available_seats, ticket_price, event_type, venue_name)
                 VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"""

        values = (event_name, date, time, total_seats, total_seats, ticket_price, event_type, venue.venue_name)

        try:
            self.cursor.execute(sql, values)
            self.conn.commit()
            print(f"Event '{event_name}' created successfully at venue '{venue.venue_name}'!")

            # Fetch and display the newly inserted row
            self.cursor.execute("SELECT * FROM event WHERE event_name = %s AND event_date = %s AND event_time = %s",
                                (event_name, date, time))
            new_event = self.cursor.fetchall()
```

```
        print("\n Added Event Details:")
        for row in new_event:
            print(row)

    except Exception as e:
        print("Error creating event:", e)




def getEventDetails(self):
    self.cursor.execute("SELECT * FROM event")
    events = self.cursor.fetchall()

    if not events:
        raise EventNotFoundException("No events found in the system.")

    for event in events:
        print(event)




def getAvailableNoOfTickets(self):
```

```
        try:
            self.cursor.execute("SELECT event_name, available_seats FROM event")
            events = self.cursor.fetchall()

            if not events:
                raise EventNotFoundException("No events found in the system.")

            # Catchy message
            print("\n These are the available limited offers! Grab your tickets before they're gone! \n")

            # Convert data to a table format
            table = tabulate(events, headers=["Event Name", "Available Tickets"], tablefmt="fancy_grid")

            print(table)

        except EventNotFoundException as e:
            print(f" {e}")
        except Exception as e:
            print(f"An unexpected error occurred: {e}")
```

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

    def book_tickets(self, eventname, num_tickets):
        try:
            # Fetch available seats & ticket price
            self.cursor.execute(
                "SELECT available_seats, ticket_price FROM event WHERE event_name = %s",
                (eventname,)
            )
            events = self.cursor.fetchone()

            if not events:
                raise EventNotFoundException("Event not found!")

            available_seats, ticket_price = events

            if available_seats >= num_tickets:
                total_cost = self.calculate_booking_cost(num_tickets, ticket_price)

                # Update available seats (no tracking in bookings table)
                self.cursor.execute(
```

```
            "UPDATE event SET available_seats = available_seats - %s WHERE
event_name = %s",
            (num_tickets, eventname)
        )

        self.conn.commit()

        # Fetch updated available seats
        self.cursor.execute("SELECT available_seats FROM event WHERE
event_name = %s", (eventname,))
        updated = self.cursor.fetchone()

        #  Final output
        print(f"\n Tickets booked successfully for event: {eventname}")
        print(f" Tickets Booked: {num_tickets}")
        print(f" Total Cost: ₹{total_cost:.2f}")
        print(f" Available Seats Left: {updated[0]}")
        print(" Hurry up! Only a few seats left!")

    else:
        print(" Not enough available seats!")

except Exception as e:
```

```
            print("□ Error booking tickets:", e)


    def cancel_booking(self, booking_id):
        # Step 1: Get event_name and num_tickets from booking
        self.cursor.execute("SELECT event_name, num_tickets FROM booking
WHERE booking_id = %s", (booking_id,))
        booking = self.cursor.fetchone()
        if not booking:
            raise InvalidBookingIDException("Invalid booking ID!")


        event_name, num_tickets = booking


        # Step 2: Get ticket price from events table
        self.cursor.execute("SELECT ticket_price FROM event WHERE event_name
= %s", (event_name,))
        event = self.cursor.fetchone()
        if not event:
            raise Exception("Event not found for the booking!")  # just in case


        ticket_price = event[0]


        # Step 3: Calculate refund
        refund_amount = self.calculate_booking_cost(num_tickets, ticket_price)
```

```
        # Step 4: Update available seats
        self.cursor.execute(
            "UPDATE event SET available_seats = available_seats + %s WHERE event_name = %s",
            (num_tickets, event_name)
        )

        # Step 5: Delete booking
        self.cursor.execute("DELETE FROM booking WHERE booking_id = %s", (booking_id,))
        self.conn.commit()

        # Step 6: Show success and refund
        print("Booking cancelled successfully!")
        print(f"Refund Amount: ₹{refund_amount:.2f}")

    def get_booking_details(self, booking_id):
        self.cursor.execute("SELECT * FROM booking WHERE booking_id = %s", (booking_id,))
        booking = self.cursor.fetchone()

        if booking:
```

```
            headers = ["Booking ID", "No. of Tickets", "Total Cost", "Booking Date",
"Event Name"]


            # Sentence format summary
            print("\n Booking Summary:")
            print(f"Booking ID {booking[0]} is for event '{booking[4]}' on
{booking[3]}.")
            print(f"Number of tickets: {booking[1]}, Total cost: ₹{booking[2]:.2f}.\n")


            # Tabular format
            print(" Booking Details:")
            print(tabulate([booking], headers=headers, tablefmt="fancy_grid"))
        else:
            print("No booking found with the given ID.")


    def get_customer_with_max_tickets(self):
        query = """
            SELECT c.customer_name, c.phone_number, b.num_tickets
            FROM customer c
            JOIN booking b ON c.BOOKING_ID = b.booking_id
            ORDER BY b.num_tickets DESC
            LIMIT 1
        """
```

self.cursor.execute(query)

result = self.cursor.fetchone()

if result:

    print("\n Customer with the Highest Ticket Booking:\n")

    print(f" Name : {result[0]}")

    print(f" Phone Number : {result[1]}")

    print(f" Tickets Booked: {result[2]}")

else:

    print("No bookings found.")

dao > BookingSystemRepositoryImpl.py > ...

```python
import pymysql
from dao.IBookingSystemRepository import IBookingSystemRepository
from util.DBUtil import DBUtil
from exception.EventNotFoundException import EventNotFoundException
from exception.InvalidBookingIDException import InvalidBookingIDException
from tabulate import tabulate

class BookingSystemRepositoryImpl(IBookingSystemRepository):
    def __init__(self):
        self.conn = DBUtil.getDBConn()
        self.cursor = self.conn.cursor()

    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue):
        sql = """INSERT INTO event (event_name, event_date, event_time, total_seats, available_seats, ticket_price, event_type, venue_name)
                 VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"""

        values = (event_name, date, time, total_seats, total_seats, ticket_price, event_type, venue.venue_name)

        try:
            self.cursor.execute(sql, values)
            self.conn.commit()
            print(f"Event '{event_name}' created successfully at venue '{venue.venue_name}'!")

            # Fetch and display the newly inserted row
            self.cursor.execute("SELECT * FROM event WHERE event_name = %s AND event_date = %s AND event_time = %s",
                                (event_name, date, time))
            new_event = self.cursor.fetchall()

            print("\n Added Event Details:")
            for row in new_event:
                print(row)
```

```
        except Exception as e:
            print("Error creating event:", e)



    def getEventDetails(self):
        self.cursor.execute("SELECT * FROM event")
        events = self.cursor.fetchall()

        if not events:
            raise EventNotFoundException("No events found in the system.")

        for event in events:
            print(event)



    def getAvailableNoOfTickets(self):
        try:
            self.cursor.execute("SELECT event_name, available_seats FROM event")
            events = self.cursor.fetchall()

            if not events:
                raise EventNotFoundException("No events found in the system.")

            # Catchy message
            print("\n🎉 These are the available limited offers! Grab your tickets before they're gone! 🎟️🔥\n")
```

```
        # Convert data to a table format
        table = tabulate(events, headers=["Event Name", "Available Tickets"], tablefmt="fancy_grid")

        print(table)

    except EventNotFoundException as e:
        print(f"⚠️ {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")


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

def book_tickets(self, eventname, num_tickets):
    try:
        # Fetch available seats & ticket price
        self.cursor.execute(
            "SELECT available_seats, ticket_price FROM event WHERE event_name = %s",
            (eventname,)
        )
        events = self.cursor.fetchone()

        if not events:
            raise EventNotFoundException("Event not found!")

        available_seats, ticket_price = events

        if available_seats >= num_tickets:
```

```python
def cancel_booking(self, booking_id):
    # Step 1: Get event_name and num_tickets from booking
    self.cursor.execute("SELECT event_name, num_tickets FROM booking WHERE booking_id = %s", (booking_id,))
    booking = self.cursor.fetchone()
    if not booking:
        raise InvalidBookingIDException("Invalid booking ID!")

    event_name, num_tickets = booking

    # Step 2: Get ticket price from events table
    self.cursor.execute("SELECT ticket_price FROM event WHERE event_name = %s", (event_name,))
    event = self.cursor.fetchone()
    if not event:
        raise Exception("Event not found for the booking!")  # just in case

    ticket_price = event[0]

    # Step 3: Calculate refund
    refund_amount = self.calculate_booking_cost(num_tickets, ticket_price)

    # Step 4: Update available seats
    self.cursor.execute(
        "UPDATE event SET available_seats = available_seats + %s WHERE event_name = %s",
        (num_tickets, event_name)
    )

    # Step 5: Delete booking
    self.cursor.execute("DELETE FROM booking WHERE booking_id = %s", (booking_id,))
    self.conn.commit()
```

```
    def cancel_booking(self, booking_id):
        )

        # Step 5: Delete booking
        self.cursor.execute("DELETE FROM booking WHERE booking_id = %s", (booking_id,))
        self.conn.commit()

        # Step 6: Show success and refund
        print("Booking cancelled successfully!")
        print(f"Refund Amount: ₹{refund_amount:.2f}")

    def get_booking_details(self, booking_id):
        self.cursor.execute("SELECT * FROM booking WHERE booking_id = %s", (booking_id,))
        booking = self.cursor.fetchone()

        if booking:
            headers = ["Booking ID", "No. of Tickets", "Total Cost", "Booking Date", "Event Name"]

            # Sentence format summary
            print("\n📄 Booking Summary:")
            print(f"Booking ID {booking[0]} is for event '{booking[4]}' on {booking[3]}.")
            print(f"Number of tickets: {booking[1]}, Total cost: ₹{booking[2]:.2f}.\n")

            # Tabular format
            print("📊 Booking Details:")
            print(tabulate([booking], headers=headers, tablefmt="fancy_grid"))
        else:
            print("No booking found with the given ID.")
```

```
def get_customer_with_max_tickets(self):
    query = """
        SELECT c.customer_name, c.phone_number, b.num_tickets
        FROM customer c
        JOIN booking b ON c.BOOKING_ID = b.booking_id
        ORDER BY b.num_tickets DESC
        LIMIT 1
    """
    self.cursor.execute(query)
    result = self.cursor.fetchone()

    if result:
        print("\n📢 Customer with the Highest Ticket Booking:\n")
        print(f"👤 Name         : {result[0]}")
        print(f"📞 Phone Number : {result[1]}")
        print(f"🎟️ Tickets Booked: {result[2]}")
    else:
        print("No bookings found.")
```

Exception :

```
class EventNotFoundException(Exception):
    pass
class InvalidBookingIDException(Exception):
    pass
```

util:

DBUtil.py:

```
import pymysql


class DBUtil:
    @staticmethod
    def getDBConn():
        return pymysql.connect(
        host="127.0.0.1",
        user="root",
        password="root",
        database="ticketbookingsystem",
        port=3306
    )
```

Mainmodule.py:

```
from dao.BookingSystemRepositoryImpl import BookingSystemRepositoryImpl
from entity.Venue import Venue
from entity.Customer import Customer
```

```
from entity.Event import Event
from entity.Booking import Booking
from exception.EventNotFoundException import EventNotFoundException
from exception.InvalidBookingIDException import InvalidBookingIDException

class TicketBookingSystem:
    def __init__(self):
        self.repository = BookingSystemRepositoryImpl()

    def run(self):
        while True:
            try:
                print("1. Create Event")
                print("2. Book Tickets")
                print("3. Cancel Tickets")
                print("4. Get Available Seats")
                print("5. Get Event Details")
                print("6. Get Booking Details Summary")
                print("7. Customer Booked maximum Tickets")
                print("8. Exit")

                choice = int(input("Enter your choice: "))
```

```
            if choice == 1:
                event_name = input("Enter event name: ")
                date = input("Enter event date (YYYY-MM-DD): ")
                time = input("Enter event time (HH:MM:SS): ")
                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: ")
                address = input("Enter venue address: ")

                venue = Venue(venue_name, address)  # Create a Venue object
                self.repository.create_event(event_name, date, time, total_seats,
ticket_price, event_type, venue)



            elif choice == 2:
                eventname = input("Enter event name: ")
                num_tickets = int(input("Enter number of tickets: "))
               self.repository.book_tickets(eventname, num_tickets)

            elif choice == 3:
                try:
```

```
            booking_id = int(input("Enter booking ID: "))
            self.repository.cancel_booking(booking_id)
        except InvalidBookingIDException as e:
            print(e)

    elif choice == 4:
        try:
            self.repository.getAvailableNoOfTickets()
        except EventNotFoundException as e:
            print(f"□ {e}")
        except Exception as e:
            print(f"An error occurred: {e}")


    elif choice == 5:
        try:
            self.repository.getEventDetails()
        except EventNotFoundException as e:
            print(f"Error: {e}")

    elif choice == 6:
        booking_id = int(input("Enter your booking ID: "))
        self.repository.get_booking_details(booking_id)
```

```
            elif choice == 7:
                self.repository.get_customer_with_max_tickets()

            elif choice == 8:
                break

            else:
                print("Invalid choice!")

        except EventNotFoundException as e:
            print(e)

        except InvalidBookingIDException as e:
            print(e)

        except Exception as e:
            print("An error occurred:", e)

if __name__ == "__main__":
    system = TicketBookingSystem()
    system.run()
```

```python
from dao.BookingSystemRepositoryImpl import BookingSystemRepositoryImpl
from entity.Venue import Venue
from entity.Customer import Customer
from entity.Event import Event
from entity.Booking import Booking
from exception.EventNotFoundException import EventNotFoundException
from exception.InvalidBookingIDException import InvalidBookingIDException

class TicketBookingSystem:
    def __init__(self):
        self.repository = BookingSystemRepositoryImpl()

    def run(self):
        while True:
            try:
                print("            WELCOME TO TICKET BOOKING SYSTEM         ")
                print("1. Create Event")
                print("2. Book Tickets")
                print("3. Cancel Tickets")
                print("4. Get Available Seats")
                print("5. Get Event Details")
                print("6. Get Booking Details Summary")
                print("7. Customer Booked maximum Tickets")
                print("8. Exit")

                choice = int(input("Enter your choice: "))

                if choice == 1:
                    event_name = input("Enter event name: ")
                    date = input("Enter event date (YYYY-MM-DD): ")
                    time = input("Enter event time (HH:MM:SS): ")
                    total_seats = int(input("Enter total seats: "))
```

```python
        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: ")
        address = input("Enter venue address: ")

        venue = Venue(venue_name, address)  # Create a Venue object
        self.repository.create_event(event_name, date, time, total_seats, ticket_price, event_type, venue)



    elif choice == 2:
        eventname = input("Enter event name: ")
        num_tickets = int(input("Enter number of tickets: "))

        # Directly call book_tickets without customers
        self.repository.book_tickets(eventname, num_tickets)

    elif choice == 3:
        try:
            booking_id = int(input("Enter booking ID: "))
            self.repository.cancel_booking(booking_id)
        except InvalidBookingIDException as e:
            print(e)

    elif choice == 4:
        try:
            self.repository.getAvailableNoOfTickets()
        except EventNotFoundException as e:
            print(f"⚠️ {e}")
```

```
            print(f"⚠ {e}")
        except Exception as e:
            print(f"An error occurred: {e}")


    elif choice == 5:
        try:
            self.repository.getEventDetails()
        except EventNotFoundException as e:
            print(f"Error: {e}")

    elif choice == 6:
        booking_id = int(input("Enter your booking ID: "))
        self.repository.get_booking_details(booking_id)

    elif choice == 7:
        self.repository.get_customer_with_max_tickets()

    elif choice == 8:
        break

    else:
        print("Invalid choice!")

except EventNotFoundException as e:
    print(e)

except InvalidBookingIDException as e:
    print(e)
```

```
                except Exception as e:
                    print("An error occurred:", e)

if __name__ == "__main__":
    system = TicketBookingSystem()
    system.run()
```

5.Output:

1.CREATING EVENT:

```
C:\Users\admin\Downloads\tbs-example>python mainmodule.py
           WELCOME TO TICKET BOOKING SYSTEM
1. Create Event
2. Book Tickets
3. Cancel Tickets
4. Get Available Seats
5. Get Event Details
6. Get Booking Details Summary
7. Customer Booked maximum Tickets
8. Exit
Enter your choice: 1
Enter event name: cookery
Enter event date (YYYY-MM-DD): 2026-09-09
Enter event time (HH:MM:SS): 11:09:09
Enter total seats: 250
Enter ticket price: 400.09
Enter event type (Movie/Sports/Concert): Sports
Enter venue name: france
Enter venue address: island
Event 'cookery' created successfully at venue 'france '!

 🔹 Added Event Details:
('cookery', datetime.date(2026, 9, 9), datetime.timedelta(seconds=40149), 250, 250, Decimal('400.09'), 'Sports', 'france ')
```

Also updated in database:

fire	2025-01-04	08:30:00	10250	10247	200.05	Movie	theatre
tennis	2025-01-05	10:00:00	10020	10020	1000.00	Sports	tulip
sai	2003-09-09	12:56:09	32	32	50.00	Movie	jesus
saraaa	2002-09-09	09:09:09	67	61	89.00	Sports	jesus
ipl	2025-07-09	04:44:44	245	245	1000.09	Sports	chepauk
cookery	2026-09-09	11:09:09	250	250	400.09	Sports	france

2.Booking Tickets:

```
            WELCOME TO TICKET BOOKING SYSTEM
1. Create Event
2. Book Tickets
3. Cancel Tickets
4. Get Available Seats
5. Get Event Details
6. Get Booking Details Summary
7. Customer Booked maximum Tickets
8. Exit
Enter your choice: 2
Enter event name: fire
Enter number of tickets: 8

✅ Tickets booked successfully for event: fire
🎟️ Tickets Booked: 8
💰 Total Cost: ₹1600.40
🪑 Available Seats Left: 10239
⚠️ Hurry up! Only a few seats left!
```

Also updated in database

popz	2025-02-01	10:30:00	10050	10050	300.05
cricket	2025-01-03	09:30:00	18500	18500	2000.09
fire	2025-01-04	08:30:00	10250	10239	200.05

3.cancel booking:

```
            WELCOME TO TICKET BOOKING SYSTEM
1. Create Event
2. Book Tickets
3. Cancel Tickets
4. Get Available Seats
5. Get Event Details
6. Get Booking Details Summary
7. Customer Booked maximum Tickets
8. Exit
Enter your choice: 3
Enter booking ID: 130
Booking cancelled successfully!
Refund Amount: ₹3000.18
```

Also updated in database:

```
mysql> select * from booking where booking_id=130;
Empty set (0.01 sec)
```

4.Displaying available seats in each event

:

```
            WELCOME TO TICKET BOOKING SYSTEM
1. Create Event
2. Book Tickets
3. Cancel Tickets
4. Get Available Seats
5. Get Event Details
6. Get Booking Details Summary
7. Customer Booked maximum Tickets
8. Exit
Enter your choice: 4

🎉 These are the available limited offers! Grab your tickets before they're gone! 🎟️🔥
```

Event Name	Available Tickets
rock cup	10050
worldcup	10100
moonlight	10052
twilt	15500
football	10502
rocky	20500
popz	10050
cricket	18500
fire	10239
tennis	10020
sai	32
saraaa	61
ipl	245
cookery	250

5.Get event details:

```
          WELCOME TO TICKET BOOKING SYSTEM
1. Create Event
2. Book Tickets
3. Cancel Tickets
4. Get Available Seats
5. Get Event Details
6. Get Booking Details Summary
7. Customer Booked maximum Tickets
8. Exit
Enter your choice: 5
('rock cup', datetime.date(2025, 1, 1), datetime.timedelta(seconds=32400), 10050, 10050, Decimal('500.05'), 'Concert', 'casagrand')
('worldcup', datetime.date(2025, 1, 2), datetime.timedelta(seconds=32400), 10100, 10100, Decimal('500.05'), 'Sports', None)
('moonlight', datetime.date(2025, 1, 3), datetime.timedelta(seconds=34200), 10050, 10052, Decimal('1500.09'), 'Concert', None)
('twilt', datetime.date(2025, 1, 4), datetime.timedelta(seconds=36000), 15500, 15500, Decimal('200.05'), 'Movie', 'royal')
('football', datetime.date(2025, 1, 1), datetime.timedelta(seconds=36000), 10500, 10502, Decimal('700.05'), 'Sports', 'popi')
('rocky', datetime.date(2025, 1, 5), datetime.timedelta(seconds=40200), 20500, 20500, Decimal('5000.05'), 'Concert', 'tres')
('popz', datetime.date(2025, 2, 1), datetime.timedelta(seconds=37800), 10050, 10050, Decimal('300.05'), 'Movie', 'cres')
('cricket', datetime.date(2025, 1, 3), datetime.timedelta(seconds=34200), 18500, 18500, Decimal('2000.09'), 'Sports', 'chepauk')
('fire', datetime.date(2025, 1, 4), datetime.timedelta(seconds=30600), 10250, 10239, Decimal('200.05'), 'Movie', 'theatre')
('tennis', datetime.date(2025, 1, 5), datetime.timedelta(seconds=36000), 10020, 10020, Decimal('1000.00'), 'Sports', 'tulip')
('sai', datetime.date(2003, 9, 9), datetime.timedelta(seconds=46569), 32, 32, Decimal('50.00'), 'Movie', 'jesus')
('saraaa', datetime.date(2002, 9, 9), datetime.timedelta(seconds=32949), 67, 61, Decimal('89.00'), 'Sports', 'jesus')
('ipl', datetime.date(2025, 7, 9), datetime.timedelta(seconds=17084), 245, 245, Decimal('1000.09'), 'Sports', 'chepauk')
('cookery', datetime.date(2026, 9, 9), datetime.timedelta(seconds=40149), 250, 250, Decimal('400.09'), 'Sports', 'france ')
```

6.Booking details summary:

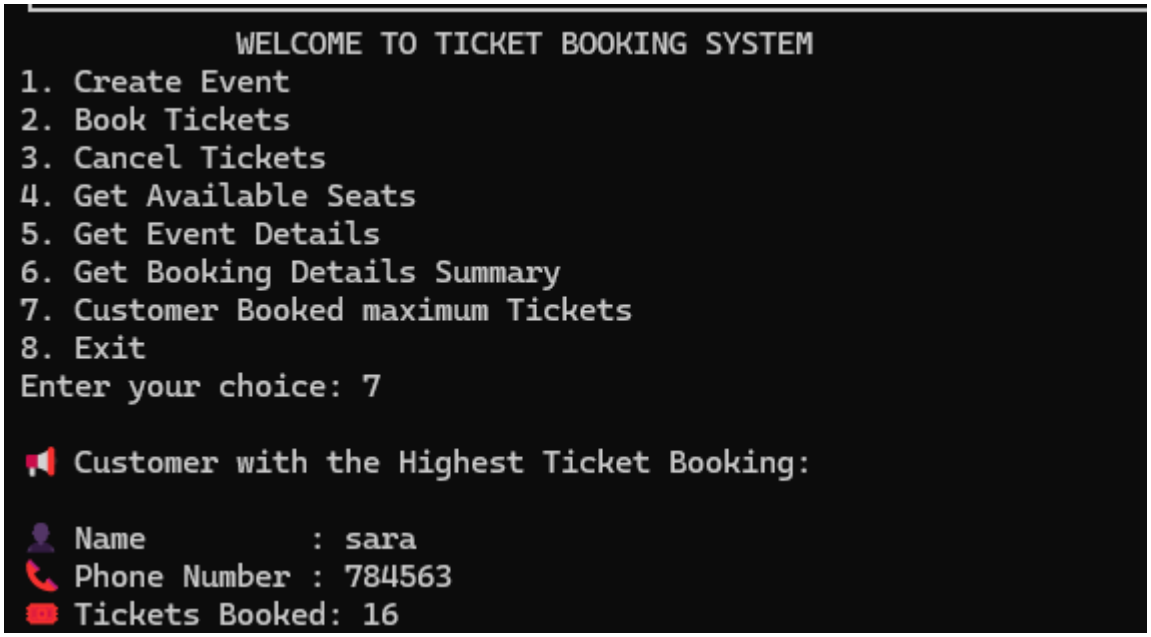
```
            WELCOME TO TICKET BOOKING SYSTEM
1. Create Event
2. Book Tickets
3. Cancel Tickets
4. Get Available Seats
5. Get Event Details
6. Get Booking Details Summary
7. Customer Booked maximum Tickets
8. Exit
Enter your choice: 6
Enter your booking ID: 140

📄 Booking Summary:
Booking ID 140 is for event 'twilt' on 2024-11-12.
Number of tickets: 5, Total cost: ₹6000.00.

📊 Booking Details:
```

Booking ID	No. of Tickets	Total Cost	Booking Date	Event Name
140	5	6000	2024-11-12	twilt

7.Displaying customers who booked maximum tickets



1.DAO (Data Access Object) Layer – dao/

The DAO layer is responsible for database operations such as creating, updating, deleting, and retrieving event and booking details.

IBookingSystemRepository.py (Interface):

- Defines abstract methods for CRUD operations related to events and bookings.
- Ensures a consistent structure across different database implementations.

BookingSystemRepositoryImpl.py (Implementation):

- Implements the database interaction methods defined in IBookingSystemRepository.py.
- Uses SQL queries (via DBUtil.py) to store and retrieve data.
- Handles operations like creating an event, booking a ticket, retrieving available seats, and canceling bookings.

2.Entity Layer – entity/

- This layer consists of data models representing different real-world objects involved in the booking system.
- Booking.py – Represents a ticket booking record with details like event ID, customer ID, number of seats, etc.
- Concert.py, Movie.py, Sport.py – Represent different types of events, inheriting from Event.py.
- Event.py – A generic event class containing attributes like event name, date, time, total seats, venue, and ticket price.
- Customer.py – Represents customer details such as name, contact info, and booking history.
- Venue.py – Stores venue-related details such as name, location, and capacity.

OOP Concept Used: Inheritance (e.g., Concert.py, Movie.py, Sport.py inherit from Event.py).

Exception Handling Layer – exception/

- To ensure robust error handling, custom exceptions are created for specific scenarios.
- EventNotFoundException.py – Raised when an event is not found in the database.
- InvalidBookingIDException.py – Raised when a user enters an invalid booking ID.

OOP Concept Used: Custom Exceptions (subclassing Exception).

Utility Layer – util/

- This layer provides helper functions, mainly for database connections.

- DBUtil.py – Establishes a connection to the SQL database (e.g., MySQL, SQLite) and executes queries.
- Design Pattern Used: Singleton Pattern (ensures a single database connection instance).

Main Execution – mainmodule.py

- This is the entry point of the system, responsible for:Initializing the database using DBUtil.py.
- Handling user interactions (e.g., event creation, booking, and cancellation).
- Calling repository methods (BookingSystemRepositoryImpl.py) to perform actions.
- Displaying responses (e.g., available events, booking status).

6.Conclusion:

This project effectively combines SQL with object-oriented programming in Python to create a functioning ticket booking system. It demonstrates the use of Python classes, mysql.connector or pymysql for database connectivity, and standard CRUD operations using SQL queries. The project follows a modular architecture, separating concerns through layered components such as entities, data access objects (DAO), utility classes, and custom exceptions.

LOOPING AND CONTROL STRUCTURES:

Task 1,2,3:

C:\Users\admin\Downloads\BATCH 4 HEXA\assign\task1.py

```
def
    print("1. Silver - Rs 50")
    print("2. Gold - Rs 100")
    print("3. Diamond - Rs 150")

    ticket_type = input("Enter ticket category (Silver/Gold/Diamond): ").strip().lower()
    no_of_tickets = int(input("Enter the number of tickets to book: "))

    if no_of_tickets > 0:
        if ticket_type == "silver":
            price = 50
        elif ticket_type == "gold":
            price = 100
        elif ticket_type == "diamond":
            price = 150
        else:
            print("Invalid ticket type. Please try again.")
            return

        total_cost = no_of_tickets * price
        print(f"Booking successful Total cost: Rs{total_cost}")
    else:
        print("Invalid number of tickets.")

book_tickets()
```

```
Ticket Categories:
1. Silver - Rs 50
2. Gold - Rs 100
3. Diamond - Rs 150
Enter ticket category (Silver/Gold/Diamond): gold
Enter the number of tickets to book: 2
Booking successful Total cost: Rs200
```