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.

```python
# Task-1
def checkBookingTicket(availableTicket,noOfBookingTicket):
    if availableTicket >= noOfBookingTicket:
        remainingTicket = availableTicket - noOfBookingTicket
        print(f"remainingTickets {remainingTicket}")
    else:
        print("Sorry Ticket unavailable")


availableTicket = int(input("Enter the available Ticket "))
noOfBookingTicket = int(input("Enter the number of booking ticket "))
checkBookingTicket(availableTicket,noOfBookingTicket)
```

```
n   main ×
⋮
C:\Users\prash\PycharmProjects\Assignment5\.venv\Scripts\python.exe C:\Users
Enter the available Ticket 20
Enter the number of booking ticket 15
remainingTickets 5

Process finished with exit code 0
```

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.

```python
# Task-2
def calculateCostOfBooking(category,noOfTickets):
    Clist = {"silver":100, "gold":200, "diamond":3000}
    if category in Clist:
        basePrice = Clist[category]
        totalPrice = basePrice*noOfTickets
        print(f"Your total Price {totalPrice}")

    else:
         print("Invaild ticket category")
#
category = input("Enter the  category(silver,gold,diamond) ")
noOfTickets = int(input("Enter the numbers of tickets "))

calculateCostOfBooking(category,noOfTickets)
```

```
main1
C:\Users\prash\PycharmProjects\Assignment5\.venv\Scripts\python.exe
Enter the  category(silver,gold,diamond) silver
Enter the numbers of tickets 3
Your total Price 300

Process finished with exit code 0
```

Task 3: Looping

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

```
while(True):
    category = input("Enter the  category(silver,gold,diamond) ")
    if category == "Exit":
        break
    noOfTickets = int(input("Enter the numbers of tickets "))

    calculateCostOfBooking(category, noOfTickets)
```

```
main1 ×

C:\Users\prash\PycharmProjects\Assignment5\.venv\Scripts\python.exe C:\U
Enter the  category(silver,gold,diamond) diamond
Enter the numbers of tickets 2
Your total Price 6000
Enter the  category(silver,gold,diamond) Exit
```

Implement oops

Task 4: Class & Object

Create a Following classes with the following attributes and methods:

1. Event Class:

2. Venue Class

3. Customer Class

4. Booking Class

```python
from enum import Enum

class eventType(Enum):
    MOVIE = "Movie"
    SPORTS = "Sports"
    CONCERT = "Concert"

class Event:
    def __init__(self, event_name, event_date, event_time, venue_name, total_seats, available_seats, ticket_price,event_type):
        self.event_name = event_name
        self.event_date = event_date
        self.event_time = event_time
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats
        self.event_type = event_type
        self.ticket_price = ticket_price
        self.booked_tickets = 0


    @property
    def getEventName(self):
         return self.event_name

    @getEventName.setter
    def setEventName(self,newEventName):
        self.event_name = newEventName

```

```python
    @property
    def getEventDate(self):
        return self.event_date

    @getEventDate.setter
    def setEventDate(self, newEventDate):
        self.event_date = newEventDate

    @property
    def getEventTime(self):
        return self.event_time

    @getEventTime.setter
    def setEventTime(self, newEventTime):
        self.event_name = newEventTime

    @property
    def getVenueName(self):
        return self.venue_name

    @getVenueName.setter
    def setVenueName(self, newVenueName):
        self.venue_name = newVenueName
```

```python
    @property
    def getTotalSeats(self):
        return self.total_seats

    @getTotalSeats.setter
    def setTotalSeats(self, newTotalSeats):
        self.total_seats = newTotalSeats

    @property
    def getAvailableSeats(self):
        return self.available_seats

    @getAvailableSeats.setter
    def setAvailableSeats(self, newAvailableSeats):
        self.available_seats = newAvailableSeats

    @property
    def getTicketPrice(self):
        return self.ticket_price

    @getTicketPrice.setter
    def setTicketPrice(self, newTicketPrice):
        self.ticket_price = newTicketPrice
```

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

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

    def book_tickets(self, num_tickets):
        if num_tickets > self.available_seats:
            print("Sorry, Tickets not available")
        else:
            self.available_seats -= num_tickets
            self.booked_tickets += num_tickets
            print("tickets booked successfully")

    def cancel_booking(self, num_tickets):
        if num_tickets > self.booked_tickets:
            print("Invalid number of tickets to cancel.")

        else:
            self.available_seats += num_tickets
            self.booked_tickets -= num_tickets
            print("tickets cancelled successfully")

    def display_event_details(self):
        print(f"Event Name: {self.event_name}")
        print(f"Date: {self.event_date}")
        print(f"Time: {self.event_time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}")
```

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

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

Task 5: Inheritance and polymorphism

1. Inheritance

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

o Attributes: 1. genre: Genre of the movie (e.g., Action, Comedy, Horror). 2. ActorName 3. ActresName

o Methods: 1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods. 2. display_event_details(): Display movie details, including genre.

```python
from event import Event
class Movie(Event):
    def __init__(self,event_name, event_date, event_time, venue_name, total_seats,available_seats, ticket_price, genre,actor_name,actress_name):
        super().__init__(event_name, event_date, event_time, venue_name, total_seats,available_seats,ticket_price)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    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}")
```

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

o Attributes: 1. artist: Name of the performing artist or band. 2. type: (Theatrical, Classical, Rock, Recital)

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

```python
from event import Event
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

    def display_event_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: o Attributes: Task 6: Abstraction Requirements: 1. sportName: Name of the game. 2. teamsName: (India vs Pakistan)

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

```python
from event import Event
class Sport(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

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

• Create a class TicketBookingSystem with the following methods:

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

o display_event_details(event: Event): Accepts an event object and calls its display_event_details() method to display event details.

o book_tickets(event: Event, num_tickets: int): 1. Accepts an event object and the number of tickets to be booked. 2. Checks if there are enough available seats for the booking. 3. If seats are available, updates the available seats and returns the total cost of the booking. 4. If seats are not available, displays a message indicating that the event is sold out.

o cancel_tickets(event: Event, num_tickets): cancel a specified number of tickets for an event.

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

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:

```python
from abc import ABC, abstractmethod

class Event(ABC):
    def __init__(self, event_name, date, time, venue_name, total_seats, available_seats, ticket_price, event_type):
        self.event_name = event_name
        self.date = date
        self.time = time
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

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

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.

```python
class Movie(Event):
    def display_event_details(self):
        print(f"Event Type: Movie")
        print(f"Event Name: {self.event_name}")
        print(f"Date: {self.date}")
        print(f"Time: {self.time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}")
```

```python
class Concert(Event):
    def display_event_details(self):
        print(f"Event Type: Concert")
        print(f"Event Name: {self.event_name}")
        print(f"Date: {self.date}")
        print(f"Time: {self.time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}")
```

```python
class Sport(Event):
    def display_event_details(self):
        print(f"Event Type: Sport")
        print(f"Event Name: {self.event_name}")
        print(f"Date: {self.date}")
        print(f"Time: {self.time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}")
```

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

```python
class BookingSystem(ABC):
    @abstractmethod
    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue_name):
        pass

    @abstractmethod
    def display_event_details(self, event):
        pass

    @abstractmethod
    def book_tickets(self, event, num_tickets):
        pass

    @abstractmethod
    def cancel_tickets(self, event, num_tickets):
        pass
```

4. Concrete TicketBookingSystem Class:

• Create a concrete class TicketBookingSystem that inherits from BookingSystem:

• TicketBookingSystem: Implement the abstract methods to create events, book t ickets, 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."

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

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

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

    def display_venue_details(self):
        print("Venue Name:", self.venue_name)
        print("Address:", self.address)
```

2. Event Class: • Attributes:o event_name, o event_date DATE, o event_time TIME, o venue (reference of class Venu), o total_seats, o available_seats, o ticket_price DECIMAL, o event_type ENUM('Movie', 'Sports', 'Concert')

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

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:

```
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):
        pass

    @abstractmethod
    def getEventDetails(self):
        pass

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

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

• getEventDetails(): return array of event details from the event class.

• getAvailableNoOfTickets(): return the total available tickets.

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

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

    @abstractmethod
    def book_tickets(self, eventname: str, num_tickets: int, arrayOfCustomer):
        pass

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

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

• 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.

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

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

    def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue):
        event = None
        if event_type == EventType.MOVIE.value:
            event = Movie(event_name, date, time, venue, total_seats, ticket_price, EventType.MOVIE)
        elif event_type == EventType.CONCERT.value:
            event = Concert(event_name, date, time, venue, total_seats, ticket_price, EventType.CONCERT)
        elif event_type == EventType.SPORTS.value:
            event = Sport(event_name, date, time, venue, total_seats, ticket_price, EventType.SPORTS)
        if event:
            self.events.append(event)
            return event
        else:
            print("Invalid event type.")
            return None

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

    def getAvailableNoOfTickets(self):
        total_available_tickets = 0
        for event in self.events:
            total_available_tickets += event.available_seats
        return total_available_tickets

```

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

• Attributes

o array of events

```python
class BookingSystemServiceProviderImpl(IBookingSystemServiceProvider, EventServiceProviderImpl):
    def calculate_booking_cost(self, num_tickets):
        pass

    def book_tickets(self, eventname, num_tickets, arrayOfCustomer):
        pass

    def cancel_booking(self, booking_id):
        pass

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

9. Create TicketBookingSystem class and perform following operations:

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

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

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.

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.

```
3   class EventNotFoundException(Exception):
4       pass
5
6
7   class InvalidBookingIDException(Exception):
8       pass
9
```

```
ticket_booking_system = TicketBookingSystem()
try:
    ticket_booking_system.book_tickets( event_name: " ",  num_tickets: 2,  array_of_customers: [])
except EventNotFoundException as e:
    print("Exception:", e)
except Exception as e:
    print("An error occurred:", e)

try:
    ticket_booking_system.cancel_booking()
except InvalidBookingIDException as e:
    print("Exception:", e)
except Exception as e:
    print("An error occurred:", e)
```

Task 11: Database Connectivity.

1. Create Venue, Event, Customer and Booking class as mentioned above Task 5.

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

3. Create interface/abstract class IEventServiceProvider, IBookingSystemServiceProvider and its implementation classes as mentioned in above Task 5.

4. Create IBookingSystemRepository interface/abstract class which include following methods to interact with database.

• create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu: Venu): Create a new event with the specified details and event type (movie, sport or concert) and return event object and should store in database.

• getEventDetails(): return array of event details from the database.

• getAvailableNoOfTickets(): return the total available tickets from the database.

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

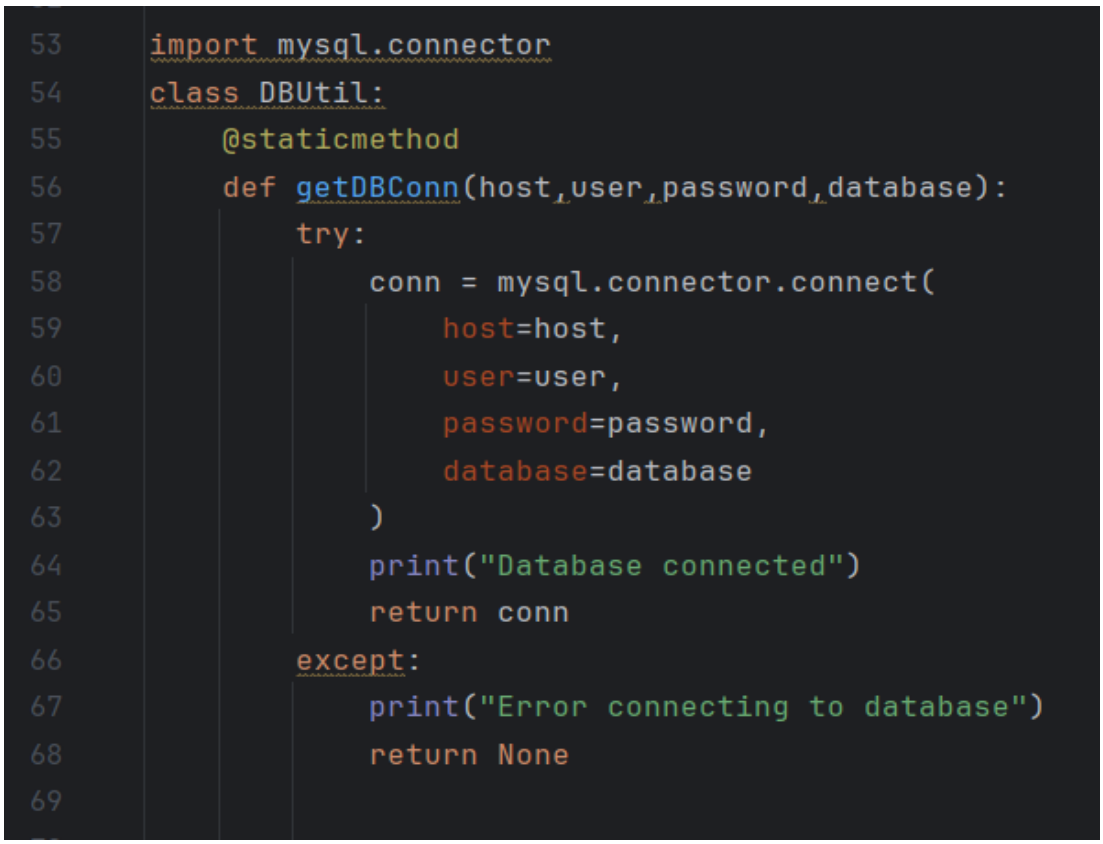
• book_tickets(eventname:str, num_tickets, listOfCustomer): Book a specified number of t ickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class and stored in database.

• cancel_booking(booking_id): Cancel the booking and update the available seats and stored in database.

• get_booking_details(booking_id): get the booking details from database.

5. Create BookingSystemRepositoryImpl interface/abstract class which implements IBookingSystemRepository interface/abstract class and provide implementation of all methods and perform the database operations.

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



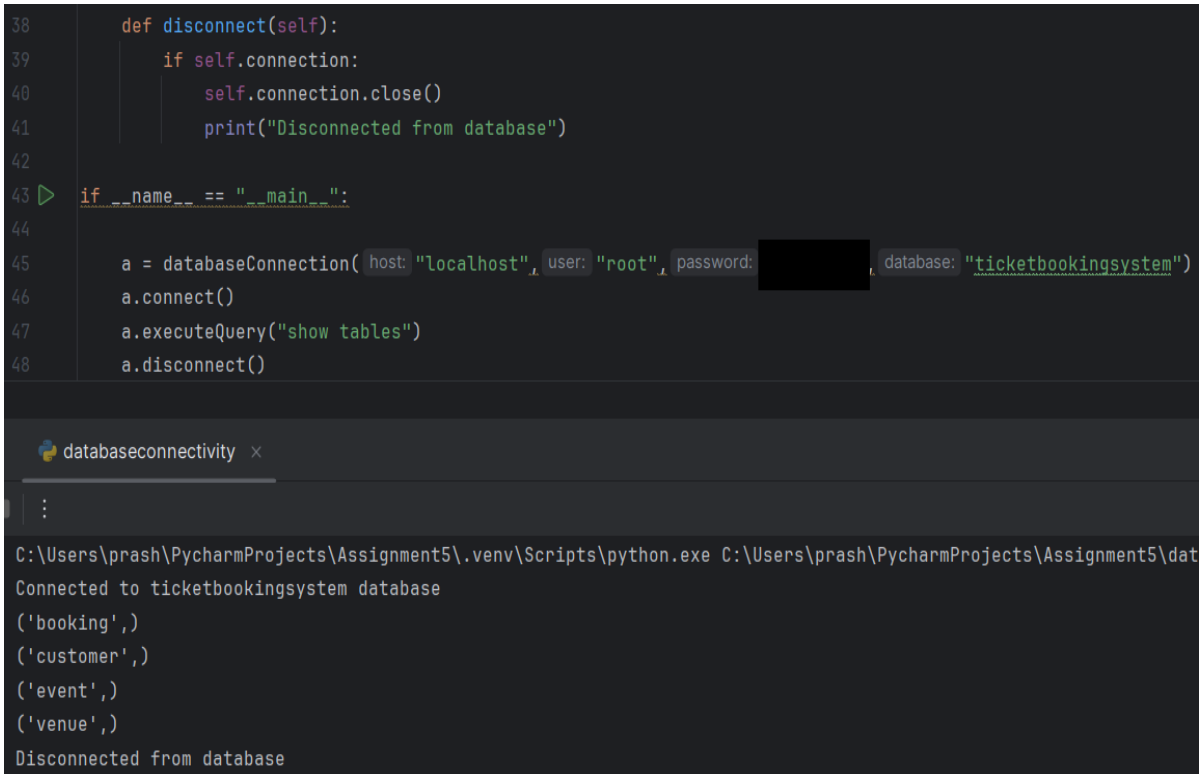
```
import mysql.connector
class DBUtil:
    @staticmethod
    def getDBConn(host,user,password,database):
        try:
            conn = mysql.connector.connect(
                host=host,
                user=user,
                password=password,
                database=database
            )
            print("Database connected")
            return conn
        except:
            print("Error connecting to database")
            return None

```

```python
import mysql.connector
class databaseConnection():
    def __init__(self,host,user,password,database):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None


    def connect(self):
        try:
            self.connection = mysql.connector.connect(
            host = self.host,
            user = self.user,
            password = self.password,
            database = self.database
            )
            print(f"Connected to {self.database} database")
        except:
            print("Could not connect to the database")


    def executeQuery(self,query):
        cursor = self.connection.cursor()
        cursor.execute(query)
        result = cursor.fetchall()
        cursor.close()
        for i in result:
            print(i)
```



7. Place the interface/abstract class in service package and interface implementation class, concrete class in bean package and TicketBookingSystemRepository class in app package.

8. Should throw appropriate exception as mentioned in above task along with handle SQLException.

9. Create TicketBookingSystem class and perform following operations:

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