Ticket Booking System

Task 1

```python
print("Tickets Available :")
availableTicket=int(input())
print("No. of Tickets")
noOfBookingTickets=int(input())

if(noOfBookingTickets<=0):
    print("Please enter a valid number of Tickets")
elif noOfBookingTickets <= availableTicket :
    print("Ticket Available")
else:
    print("Come Back Later")
```

Task 2

```python
def total_cost(num_tickets,category):
    silver = 25
    gold = 50
    diamond = 100

    if(category in ["silver","gold","diamond"]):
        if num_tickets > 0:
            if category == "silver":
                total=num_tickets*silver
            elif category == "gold":
                total=num_tickets*gold
            elif category == "diamond":
                total=num_tickets*diamond
            return total
        else:
            print("Enter number greater than 0")
    else:
        print("Invalid choice")
    return None


category = input("Enter the ticket category (silver,gold,diamond):")
num_tickets= int(input("Enter the number of Tickets :"))

print("Total cost = ",total_cost(num_tickets,category))
```

Task 3

```python
def total_cost(num_tickets,category):
    silver = 25
    gold = 50
    diamond = 100

    if(category in ["silver","gold","diamond"]):
        if num_tickets > 0:
            if category == "silver":
                total=num_tickets*silver
            elif category == "gold":
                total=num_tickets*gold
            elif category == "diamond":
                total=num_tickets*diamond
            return total
        else:
            print("Enter number greater than 0")
    else:
        print("Invalid choice")
    return None


while True:
    category = input("Enter the ticket category (silver,gold,diamond):")
    if category.lower()=='exit':
        print("Thank you!")
        break
    num_tickets = int(input("Enter the number of Tickets :"))

    print("Total cost = ", total_cost(num_tickets, category))
```

Task 4,5,6,7,8,9,10,11

Entity

Event Class

```python
class Event:
    def __init__(self, event_id, event_name, event_date, event_time,
venue_name, total_seats, ticket_price, event_type):
        self.event_id = event_id
        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 = total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    # Getter and Setter methods
    def get_event_id(self):
        return self.event_id

    def set_event_id(self,event_id):
        self.event_id=event_id

    def get_event_name(self):
        return self.event_name

    def set_event_name(self, event_name):
        self.event_name = event_name

    def get_event_date(self):
        return self.event_date

    def set_event_date(self, event_date):
        self.event_date = event_date

    def get_event_time(self):
        return self.event_time

    def set_event_time(self, event_time):
        self.event_time = event_time

    def get_venue_name(self):
        return self.venue_name

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

    def get_total_seats(self):
        return self.total_seats

    def set_total_seats(self, total_seats):
        self.total_seats = total_seats

    def get_available_seats(self):
        return self.available_seats

    def set_available_seats(self, available_seats):
        self.available_seats = available_seats

    def get_ticket_price(self):
        return self.ticket_price

    def set_ticket_price(self, ticket_price):
        self.ticket_price = ticket_price

    def get_event_type(self):
        return self.event_type

    def set_event_type(self, event_type):
        self.event_type = event_type


class Movie(Event):
    def __init__(self, genre,event_id, actress_name, actor_name,
event_name, event_date, event_time, venue_name, total_seats, ticket_price,
event_type):
        super().__init__(event_id,event_name, event_date, event_time,
venue_name, total_seats, ticket_price, "Movie")
        self.genre = genre
        self.Actress_Name = actress_name
        self.Actor_Name = actor_name


    def get_genre(self):
        return self.genre

    def set_genre(self,genre):
        self.genre=genre

    def get_Actress_Name(self):
        return self.Actress_Name
    def set_Actress_Name(self,actress_Name):
        self.Actress_Name=actress_Name
    def get_Actor_Name(self):
        return self.Actor_Name
    def set_Actor_Name(self,actor_Name):
        self.Actor_Name=actor_Name


class Concert(Event):
    def __init__(self,artist,concert_type,event_id,event_name, event_date,
event_time, venue_name, total_seats, ticket_price ):
        super().__init__(event_id,event_name, event_date, event_time,
venue_name, total_seats, ticket_price,"Concert" )
        self.artist=artist
        self.concert_type=concert_type

    def get_artist(self):
        return self.artist
    def set_artist(self,artist):
        self.artist=artist

    def get_concert_type(self):
        return self.concert_type

    def set_concert_type(self, concert_type):
        self.concert_type = concert_type




class Sports(Event):
    def __init__(self,event_id,event_name, event_date, event_time,
venue_name, total_seats, ticket_price,team1,team2,sports_type):
        super().__init__(event_id,event_name, event_date, event_time,
venue_name, total_seats, ticket_price,"Sports")
        self.sports_type=sports_type
        self.team1=team1
        self.team2=team2

    def get_sports_type(self):
        return self.sports_type
    def set_sports_type(self,sports_type):
        self.sports_type=sports_type

    def get_team1(self):
        return self.team1
    def set_team1(self,team1):
        self.team1=team1

    def get_team2(self):
        return self.team2
    def set_team2(self,team2):
        self.team2=team2

    def __str__(self):
        return f'event_name : {self.event_name}'
```

Customer class

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

    def get_customer_id(self):
        return self.customer_id
    def set_customer_id(self,customer_id):
        self.customer_id=customer_id


    def __str__(self):
        return (f"Customer Name : {self.customer_name}\n"
                f"Email : {self.email}\n"
                f"Phone Number : {self.phone_number}\n"
                f"Customer ID: {self.customer_id}")
```

Venue Class

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

    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
```

Booking class

```
from entity.Event import Event
from entity.Customer import Customer

class Booking(Event,Customer):
    def __init__(self):
        super().__init__()
        # self.event = Event
        # self.customer = Customer
        self.num_tickets_booked = 0
        self.total_booking_cost = 0.0
    #     self.booking_id = booking_id
    #     self.booking_name = booking_name
    #
    # def get_booking_id(self):
    #     return self.booking_id
    #
    # def set_booking_id(self,booking_id):
    #     self.booking_id = booking_id
    #
    # def get_booking_name(self):
    #     return self.booking_name
    #
    # def set_booking_name(self,booking_name):
    #     self.booking_name = booking_name
    #
    #
    #
    #
```

TicketBookingSystem Class

```
from entity.Event import Movie,Concert,Sports
class TicketBookingSystem:
    def __init__(self):
        self.events = []

    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue_name):
        if event_type.lower() == "movie":
            event = Movie(event_name, date, time, venue_name, total_seats,
ticket_price, "", "", "")
        elif event_type.lower() == "concert":
            event = Concert(event_name, date, time, venue_name,
total_seats, ticket_price, "", "")
        elif event_type.lower() == "sports":
            event = Sports(event_name, date, time, venue_name, total_seats,
ticket_price, "", "", "")
        else:
            print("Invalid event type.")
            return None

        self.events.append(event)
        return event
```

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

    def book_tickets(self, event, num_tickets):
        if event:
            if num_tickets > 0 and num_tickets <= event.available_seats:
                event.book_tickets(num_tickets)
                total_cost = event.ticket_price * num_tickets
                print(f"Total Cost: ${total_cost}")
                return total_cost
            else:
                print("Invalid number of tickets or not enough available
seats.")
        else:
            print("Event not found.")
        return 0

    def cancel_tickets(self, event, num_tickets):
        if event:
            event.cancel_booking(num_tickets)
        else:
            print("Event not found.")

    def main(self):
        while True:
            print("\nTicket Booking System Menu:")
            print("1. Create Event")
            print("2. Display Event Details")
            print("3. Book Tickets")
            print("4. Cancel Tickets")
            print("5. Exit")

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

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

                self.create_event(event_name, date, time, total_seats,
ticket_price, event_type, venue_name)

            elif choice == "2":
                event_index = int(input("Enter the index of the event to
display details: "))
                if 0 <= event_index < len(self.events):
                    self.display_event_details(self.events[event_index])
                else:
                    print("Invalid event index.")

            elif choice == "3":
                event_index = int(input("Enter the index of the event to
book tickets: "))
                if 0 <= event_index < len(self.events):
                    num_tickets = int(input("Enter the number of tickets to
book: "))
                    self.book_tickets(self.events[event_index],
num_tickets)
                else:
                    print("Invalid event index.")

            elif choice == "4":
                event_index = int(input("Enter the index of the event to
cancel tickets: "))
                if 0 <= event_index < len(self.events):
                    num_tickets = int(input("Enter the number of tickets to
cancel: "))
                    self.cancel_tickets(self.events[event_index],
num_tickets)
                else:
                    print("Invalid event index.")

            elif choice == "5":
                print("Exiting the Ticket Booking System. Thank you!")
                break

            else:
                print("Invalid choice. Please enter a number between 1 and
5.")

# Example usage:
ticket_system = TicketBookingSystem()
ticket_system.main()
```

DAO

EventDao

```
def total_cost(num_tickets,category):
    silver = 25
    gold = 50
    diamond = 100

    if(category in ["silver","gold","diamond"]):
        if num_tickets > 0:
            if category == "silver":
                total=num_tickets*silver
            elif category == "gold":
                total=num_tickets*gold
            elif category == "diamond":
                total=num_tickets*diamond
            return total
        else:
            print("Enter number greater than 0")
    else:
        print("Invalid choice")
    return None
```

```
while True:
    category = input("Enter the ticket category (silver,gold,diamond):")
    if category.lower()=='exit':
        print("Thank you!")
        break
    num_tickets = int(input("Enter the number of Tickets :"))

    print("Total cost = ", total_cost(num_tickets, category))
```

CustomerDao

```
#from entity.Customer import Customer
from entity.booking import Booking

class CustomerDao( Booking):
    def __init__(self):
        super().__init__()

    def perform_customer_actions(self):
        while True:
            print("(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 
0.EXIT")
            ch=int(input("Enter Choice: "))
            if ch==1:
                self.create_customer_table()
            elif ch==2:
                print(self.add_customer())
            elif ch==3:
                print(self.update_customer())
            elif ch==4:
                print(self.delete_customer())
            elif ch==5:
                self.select_customer()
            elif ch==0:
                break
            else:
                print("Invalid Choice")


    def create_customer_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Customer(
            customer_id int primary key,
            customer_name varchar(50) not null,
            email varchar(50),
            phone_number varchar(20) not null
            booking_id int
            )'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print("Event Table created Successfully")
        except Exception as e:
            print(e)


    def add_customer(self):
        try:
            self.open()
            self.customer_id=int(input("Customer ID: "))
            self.customer_name=input("Customer Name : ")
            self.email = input("Email : ")
            self.phone_number = input("Phone Number : ")
            self.booking_id = int(input("Booking ID : "))
            data =
[(self.customer_id,self.customer_name,self.email,self.phone_number,self.boo
king_id)]
            insert_str = '''INSERT into Customer(customer_id
,customer_name,email,phone_number,booking_id)
            values(%s,%s,%s,%s,%s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            self.close()
            return True

        except Exception as e:

            return e

    def update_customer(self):
        try:
            self.open()
            self.customer_id=int(input("Customer id : "))
            self.customer_name = input("Customer Name : ")
            self.email = input("Email : ")
            self.phone_number = input("Phone Number : ")
            self.booking_id = int(input("Booking ID  : "))
            data = [(self.customer_id,self.customer_name, self.email,
self.phone_number,self.booking_id)]
            update_str = '''UPDATE Customer set  customer_id=%s,
            customer_name=%s,email=%s,phone_number=%s,booking_id=%s'''
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            self.close()
            return True

        except Exception as e:

            return e

    def delete_customer(self):
        try:
            self.open()
            customer_id = input("Enter Customer Id to be Deleted : ")
            delete_str = f'''Delete from Customer where customer_id =
{customer_id}'''
            self.stmt.executemany(delete_str)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def select_customer(self):
        try:
            select_str = '''select * from Customer'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            self.close()
            print("Records in Customer Table : ")
            for i in records:
                print(i)
        except Exception as e:
            print(e)

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

Venue Dao

```
from entity.venue import Venue

class VenueDao(Venue):
    def __init__(self):
        super().__init__()

    def perform_venue_actions(self):
        while True:
            print("(Venue) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT
0.EXIT")
            ch = int(input("Enter Choice: "))
            if ch == 1:
                self.create_venue_table()
            elif ch == 2:
                print(self.add_venue())
            elif ch == 3:
                print(self.update_venue())
            elif ch == 4:
                print(self.delete_venue())
            elif ch == 5:
                self.select_venue()
            elif ch == 0:
                break
            else:
                print("Invalid Choice")

    def create_venue_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Venue(
            venue_id int primary key,
            venue_name varchar(100) not null,
            address varchar(100) not null
            )'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print("Event Table created Successfully")
        except Exception as e:
            print(e)

    def add_venue(self):
        try:
            self.open()
            self.venue_id = int(input("Venue ID: "))
            self.venue_name = input("Venue Name : ")
            self.address = input("Address : ")
            data = [(self.venue_id,self.venue_name, self.address)]
            insert_str = '''INSERT into Venue(venue_id,venue_name,address)
values(%s,%s,%s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            self.close()
            return True

        except Exception as e:

            return e

    def update_venue(self):
        try:
            self.open()
            self.venue_id = int(input("Venue ID: "))
            self.venue_name = input("Venue Name : ")
            self.address = input("Address : ")
            data = [(self.venue_id, self.venue_name, self.address)]
            insert_str = '''Update Venue set venue_id=%s , venue_name=%s ,
address=%s'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            self.close()
            return True

        except Exception as e:

            return e

    def delete_venue(self):
        try:
            self.open()
            venue_id = input("Enter Venue Id to be Deleted : ")
            delete_str = f'''Delete from Venue where venue_id =
{venue_id}'''
            self.stmt.executemany(delete_str)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def select_venue(self):
        try:
            select_str = '''select * from Venue'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            self.close()
            print("Records in Venue Table : ")
            for i in records:
                print(i)
        except Exception as e:
            print(e)

    def display_venue_details(self):
        print(f"Venue_Name:{self.venue_name}")
        print(f"Address : {self.address}")
```

BookingDao

```python
import entity.booking

class BookingDao(entity.booking.Booking):
    def __init__(self):
        super().__init__()

    def perform_booking_actions(self):
        while True:
            print("(Booking) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 
0.EXIT")
            ch = int(input("Enter Choice: "))
            if ch == 1:
                self.create_booking_table()
            elif ch == 2:
                print(self.add_booking())
            elif ch == 3:
                print(self.update_booking())
            elif ch == 4:
                print(self.delete_booking())
            elif ch == 5:
                self.select_booking()
            elif ch == 0:
                break
            else:
                print("Invalid Choice")

    def create_booking_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Booking(
            Crete Table Booking(
            booking_id int primary key,
            booking_name varchar(50) not null,
            )
            )'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print("Booking Table created Successfully")
        except Exception as e:
            print(e)

    def add_booking(self):
        try:
            self.open()
            self.booking_id = int(input("Booking ID: "))
            self.booking_name = input("Booking Name : ")
            data = [(self.booking_id,self.booking_name, self.address)]
            insert_str = '''INSERT into Booking(booking_id,booking_name) 
values(%s,%s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            self.close()
            return True

        except Exception as e:

            return e

    def update_booking(self):
        try:
            self.open()
            self.booking_id = int(input("Booking ID: "))
            self.booking_name = input("Booking Name : ")
            data = [(self.booking_id, self.booking_name, self.address)]
            insert_str = '''Update Booking set
booking_id=%s,booking_name=%s'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            self.close()
            return True

        except Exception as e:

            return e

    def delete_booking(self):
        try:
            self.open()
            booking_id = input("Enter Booking Id to be Deleted : ")
            delete_str = f'''Delete from Booking where booking_id =
{booking_id}'''
            self.stmt.executemany(delete_str)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def select_booking(self):
        try:
            select_str = '''select * from Booking'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            self.close()
            print("Records in Booking Table : ")
            for i in records:
                print(i)
        except Exception as e:
            print(e)

    def calculate_booking_cost(self, num_tickets):
        self.num_tickets_booked = num_tickets
        self.total_booking_cost = num_tickets * self.event.ticket_price

    def book_tickets(self, num_tickets):
        if num_tickets <= self.event.available_seats:
            self.event.book_tickets(num_tickets)
            self.calculate_booking_cost(num_tickets)
            print(f"{num_tickets} tickets booked successfully.")
        else:
            print("Not enough available seats to book the requested number
of tickets.")

    def cancel_booking(self):
        self.event.cancel_booking(self.num_tickets_booked)
        print(f"{self.num_tickets_booked} tickets canceled.")
        self.num_tickets_booked = 0
        self.total_booking_cost = 0.0
    def get_available_no_of_tickets(self):
        return self.event.available_seats

    def get_event_details(self):
        return self.event.display_event_details()
```

Event Abstract

```
from abc import ABC,abstractmethod

class Event(ABC):
    def
__init__(self,event_name,event_date,event_time,venue_name,total_seats,avail
able_seats,ticket_price):
        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 = total_seats
        self.ticket_price = ticket_price



    @abstractmethod
    def display_event_details(self):
        pass

    @abstractmethod
    def book_tickets(self,num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self,num_tickets):
        pass

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

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

class Movie(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, genre, actor_name, actress_name):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def display_event_details(self):
        print(f"Event Name: {self.event_name}")
        print(f"Genre: {self.genre}")
        print(f"Actor: {self.actor_name}")
        print(f"Actress: {self.actress_name}")
        print(f"Date: {self.event_date}, Time: {self.event_time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}, Available Seats:
{self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}\n")

    def book_tickets(self, num_tickets):
        if num_tickets > 0 and num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}")
        else:
            print("Invalid number of tickets or not enough available
seats.")

    def cancel_booking(self, num_tickets):
        if num_tickets > 0 and num_tickets <=
self.get_booked_no_of_tickets():
            self.available_seats += num_tickets
            print(f"{num_tickets} tickets canceled for {self.event_name}")
        else:
            print("Invalid number of tickets to cancel.")

class Concert(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, artist, concert_type):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price)
        self.artist = artist
        self.concert_type = concert_type

    def display_event_details(self):
        print(f"Event Name: {self.event_name}")
        print(f"Artist: {self.artist}")
        print(f"Concert Type: {self.concert_type}")
        print(f"Date: {self.event_date}, Time: {self.event_time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}, Available Seats:
{self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}\n")

    def book_tickets(self, num_tickets):
        if num_tickets > 0 and num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}")
        else:
            print("Invalid number of tickets or not enough available
seats.")

    def cancel_booking(self, num_tickets):
        if num_tickets > 0 and num_tickets <=
self.get_booked_no_of_tickets():
            self.available_seats += num_tickets
            print(f"{num_tickets} tickets canceled for {self.event_name}")
        else:
            print("Invalid number of tickets to cancel.")

class Sports(Event):
    def __init__(self, event_name, event_date, event_time, venue_name,
total_seats, ticket_price, sport_type, team1, team2):
        super().__init__(event_name, event_date, event_time, venue_name,
total_seats, ticket_price)
        self.sport_type = sport_type
        self.team1 = team1
        self.team2 = team2

    def display_event_details(self):
        print(f"Event Name: {self.event_name}")
        print(f"Sport Type: {self.sport_type}")
        print(f"Teams: {self.team1} vs {self.team2}")
        print(f"Date: {self.event_date}, Time: {self.event_time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}, Available Seats:
{self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}\n")

    def book_tickets(self, num_tickets):
        if num_tickets > 0 and num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"{num_tickets} tickets booked for {self.event_name}")
        else:
            print("Invalid number of tickets or not enough available
seats.")

    def cancel_booking(self, num_tickets):
        if num_tickets > 0 and num_tickets <=
self.get_booked_no_of_tickets():
            self.available_seats += num_tickets
            print(f"{num_tickets} tickets canceled for {self.event_name}")
        else:
            print("Invalid number of tickets to cancel.")
```

Exceptions

```
class InvalidBookingIDException(Exception):
    def __init__(self, booking_id):
        super().__init__(f"Invalid Booking ID: {booking_id}.")
```

```
class EventNotFoundException(Exception):
    def __init__(self, event_id):
        super().__init__(f"Event '{event_id}' not found.")
```

```
from exception.EventException import EventNotFoundException
from exception.InvalidBookingException import InvalidBookingIDException
from main.IBookingSystemRepository import IBookingSystemRepository


class TicketBookingSystem:
    def __init__(self, booking_system_repository:
IBookingSystemRepository):
        self.booking_system_repository = booking_system_repository

    def book_tickets(self, event_name, num_tickets):
        try:
            event = self.get_event_by_name(event_name)
            if event:
                # Attempt to book tickets
                event.book_tickets(num_tickets)
                # Save the updated event details to the database
                self.booking_system_repository.book_tickets(event_name,
num_tickets, [])
            else:
                raise EventNotFoundException(event_name)
        except EventNotFoundException as e:
            print(f"Error: {e}")

    def get_event_by_name(self, event_name):
        events = self.booking_system_repository.get_event_details()
        for event in events:
            if event.event_name == event_name:
                return event
        return None

    def cancel_booking(self, booking_id):
        try:
            # Attempt to cancel booking
            self.booking_system_repository.cancel_booking(booking_id)
        except InvalidBookingIDException as e:
            print(f"Error: {e}")

    def view_booking_details(self, booking_id):
        try:
            # Attempt to get booking details
            booking_details =
self.booking_system_repository.get_booking_details(booking_id)
            print(f"Booking Details: {booking_details}")
        except InvalidBookingIDException as e:
            print(f"Error: {e}")


# Main program
def main():
    try:
        # Initialize the ticket booking system with a repository
        ticket_system = TicketBookingSystem(BookingSystemRepositoryImpl())

        # Example usage
        ticket_system.book_tickets("Movie Event", 3)
        ticket_system.view_booking_details(123)
        ticket_system.cancel_booking(123)

    except Exception as e:
        print(f"Unhandled Exception: {e}")


if __name__ == "__main__":
    main()
```

Main

BookingSystemRepositoryImpl

```
from main.IBookingSystemRepository import IBookingSystemRepository
from dao.bookingdao import BookingDao
from util.DButil import DButil
from dao.Eventdao import EventDao
from dao.Venuedao import VenueDao
from dao.Customerdao import CustomerDao
from exception.EventException import EventNotFoundException
from exception.InvalidBookingException import InvalidBookingIDException


class BookingSystemRepositoryImpl(IBookingSystemRepository):
    def __init__(self):
        self.connection = DButil.getDBConn()
        self.create_tables()

    def create_tables(self):
        b = BookingDao()
        b.create_booking_table()
        e = EventDao()
        e.create_event_table()
        c = CustomerDao()
        c.create_customer_table()
        v = VenueDao()
        v.create_venue_table()

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

        e = EventDao()
        e.add_event()

    def get_event_details(self):
        e=EventDao()
        e.display_event_details()

    def get_available_no_of_tickets(self):
        b = BookingDao()
        b.get_available_no_of_tickets()

    def calculate_booking_cost(self, num_tickets):
        pass

    def book_tickets(self, event_name, num_tickets, list_of_customers):
        pass

    def cancel_booking(self, booking_id):
        pass

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

IBookingServiceSystemProvider

```
from dao.Eventdao import EventDao
from dao.bookingdao import BookingDao
from entity.Event import Event
from exception.EventException import EventNotFoundException
from exception.InvalidBookingException import InvalidBookingIDException


class IBookingSystemServiceProvider(EventDao,BookingDao):
    def calculate_booking_cost(self, num_tickets):
        # pass
        self.num_tickets_booked = num_tickets
        self.total_booking_cost = num_tickets * 
Event.get_ticket_price(self)



    def book_tickets(self, num_tickets):
        if num_tickets <= Event.get_available_seats(self):
            EventDao.book_tickets(num_tickets)
            self.calculate_booking_cost(num_tickets)
            print(f"{num_tickets} tickets booked successfully.")
        else:
            print("Not enough available seats to book the requested number 
of tickets.")



    def cancel_booking(self, booking_id):
        pass

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

IBookingServiceRepository

```
from abc import ABC, abstractmethod
from dao.Eventdao import EventDao


class IBookingSystemRepository(ABC):
    @abstractmethod
    def create_event(self, event_name, date, time, total_seats, 
ticket_price, event_type, venue):
        pass

    @abstractmethod
    def get_event_details(self):
        pass

    @abstractmethod
    def get_available_no_of_tickets(self):
        pass

    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        pass

    @abstractmethod
    def book_tickets(self, event_name, num_tickets, list_of_customers):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

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

IEventServiceProvider

```
from abc import ABC, abstractmethod
from entity.Event import Event

class IEventServiceProvider(ABC):
    @abstractmethod
    def create_event(self, event_name, date, time, total_seats,
ticket_price, event_type, venue):
        pass

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

Main

```
from dao.bookingdao import BookingDao
from dao.Eventdao import EventDao
from dao.Venuedao import VenueDao
from dao.Customerdao import CustomerDao
from main.IBookingServiceSystemProvider import
IBookingSystemServiceProvider
from exception.EventException import EventNotFoundException
from exception.InvalidBookingException import InvalidBookingIDException
from util.DBconnection import DBConnection

def main():

    dbconnection = DBConnection()

    try:
        dbconnection.open()
        print("Database Is Connected:")
    except Exception as e:
        print(e)

    try:
        print("_" * 30)
        print("Ticket Booking System")
        print("_" * 30)
        print("Welcome to Ticket Booking  System!")

        ticket_booking_system = IBookingSystemServiceProvider

        while True:
            print("1.Event 2.Customer 3.Venue 4.Booking  0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                ev = EventDao()
                ev.perform_event_actions()
            elif ch == 2:
                c = CustomerDao()
                c.perform_customer_actions()
            elif ch == 3:
                v = VenueDao()
                v.perform_venue_actions()
            elif ch == 4:
                b = BookingDao()
                b.perform_booking_actions()

            elif ch == 0:
                break
            else:
                print("Invalid choice")

        while True:
            print("=" * 10)
            print("---Main Menu---")
            print("=" * 10)
            print("1.Booking Cost\n2.Number of Tickets\n3.Cancel
Booking\n4.Booking Details\n0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                print(f'Total Cost
{ticket_booking_system.calculate_booking_cost()}')
            elif ch == 2:
                print(ticket_booking_system.book_tickets(int(input('Enter
number of tickets '))))
            elif ch == 3:
                print(ticket_booking_system.cancel_booking(int(input('Enter
Booking id to be Cancelled: '))))
            elif ch == 4:
                print(ticket_booking_system.get_booking_details())
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    except EventNotFoundException as e:
        print(e)

    except InvalidBookingIDException as e:
        print(e)

    except Exception as e:
        print(e)

    finally:
        dbconnection.close()
        print("Thankyou !")
        print("Disconnected:")


if __name__ == "__main__":
    main()
```

Util

DBConnection

```
import sys
import mysql.connector as sql
from util.DButil import DButil
```

```
class DBConnection:
    def open(self):
        try:
            connection_properties=DButil.getDBConn()
            self.conn=sql.connect(**connection_properties)
            self.stmt=self.conn.cursor()
        except Exception as e:
            print(str(e) + 'Database not Connected:')
            sys.exit(1)

    def close(self):
        self.conn.close()
```

DButil

```
class DButil:
    connection_properties = None

    @staticmethod
    def getDBConn():
        if DButil.connection_properties is None:
            host = 'localhost'
            database = 'TicketBookingSystem'
            port = '3306'
            user = 'root'
            password = 'root'
            DButil.connection_properties = {'host': host,'database':
database,'port':port,'user': user,'password': password}
        return DButil.connection_properties
```

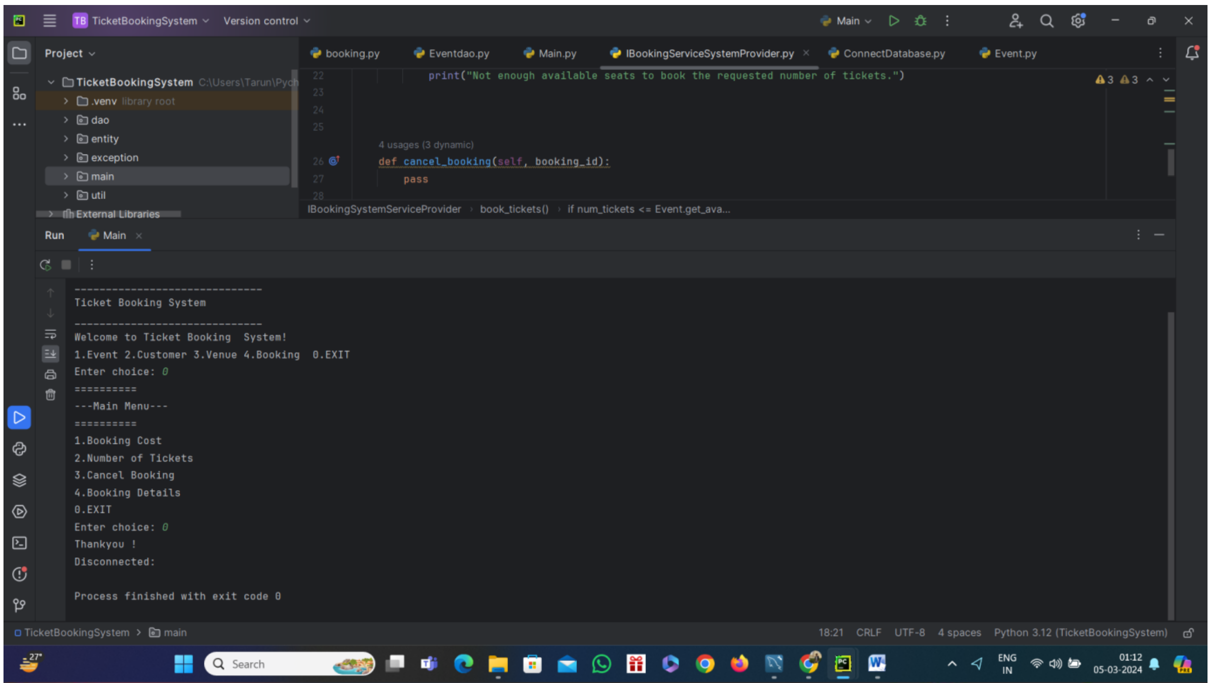
Other Way

Connect database

```
import mysql.connector
con = mysql.connector.connect(
    host="localhost",
    user="root",
    password="root",
    port="3306",
    database="TicketBookingSystem")

cursor = con.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS Booking(
            Crete Table Booking(
            booking_id int primary key,
            booking_name varchar(50) not null,
            )
            )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS Venue(
            venue_id int primary key,
            venue_name varchar(100) not null,
            address varchar(100) not null
            )''')
```

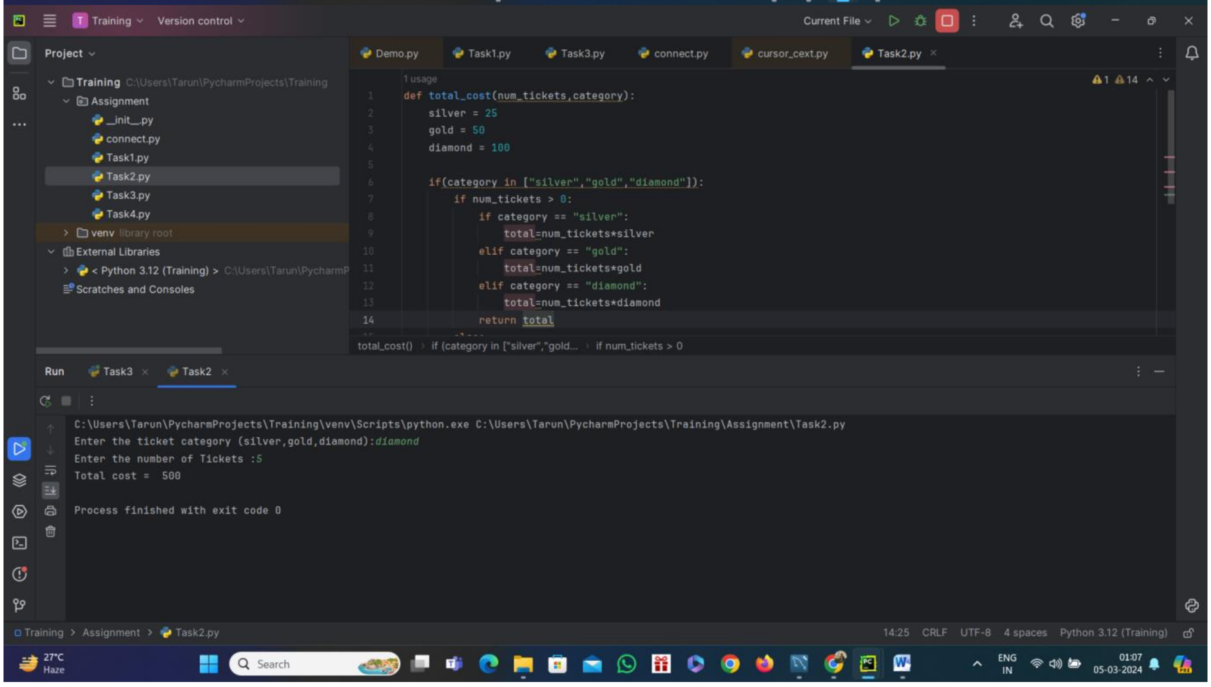
```
cursor.execute('''CREATE TABLE IF NOT EXISTS Event(
            event_id int primary key,
            event_name varchar(255) not null,
            event_date date not null,
            event_time time not null,
            venue_id int,
            total_seats int not null,
            available_seats int not null,
            ticket_price decimal(10,2) not null,
            event_type enum('Movie','Sports','Concert') not null,
            booking_id int,
            foreign key (venue_id) references venue(venue_id)
            )''')



cursor.execute('''CREATE TABLE IF NOT EXISTS Customer(
            customer_id int primary key,
            customer_name varchar(50) not null,
            email varchar(50),
            phone_number varchar(20) not null
            booking_id int
            )''')
con.commit()
cursor.close()
con.close()
```

SS



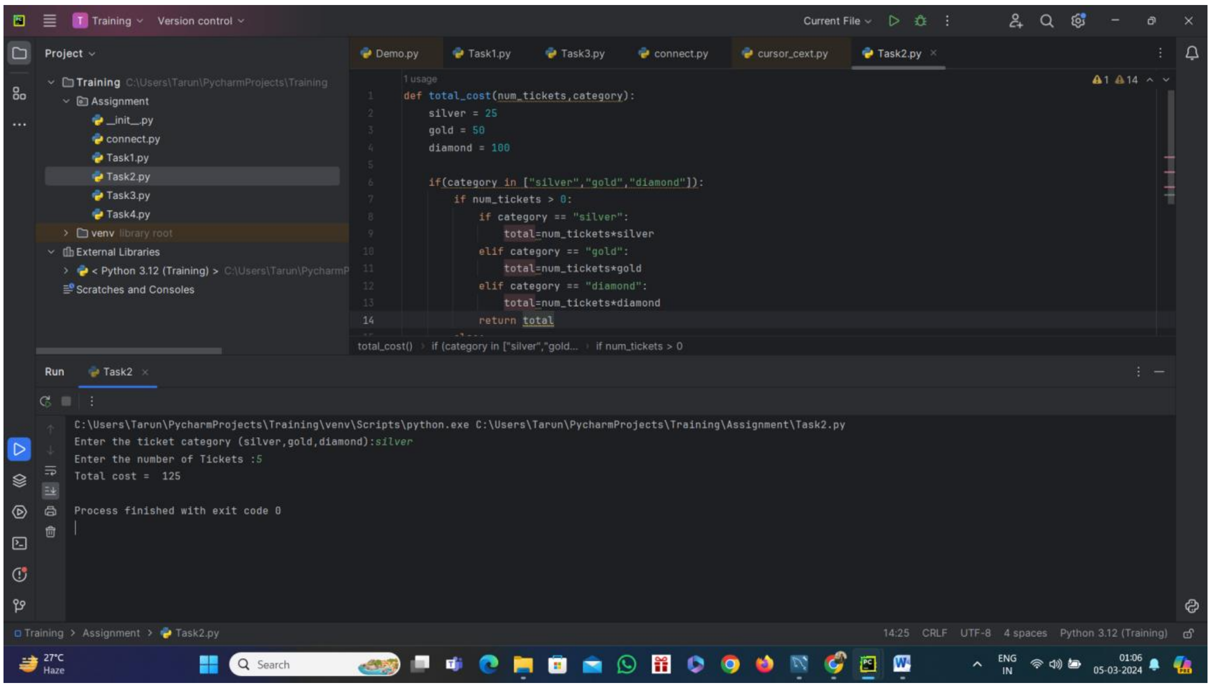
```python
        print("Not enough available seats to book the requested number of tickets.")



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

```
------------------------------
Ticket Booking System

------------------------------
Welcome to Ticket Booking  System!
1.Event 2.Customer 3.Venue 4.Booking  0.EXIT
Enter choice: 0
==========
---Main Menu---
==========
1.Booking Cost
2.Number of Tickets
3.Cancel Booking
4.Booking Details
0.EXIT
Enter choice: 0
Thankyou !
Disconnected:

Process finished with exit code 0
```



```python
def total_cost(num_tickets,category):
    silver = 25
    gold = 50
    diamond = 100

    if(category in ["silver","gold","diamond"]):
        if num_tickets > 0:
            if category == "silver":
                total=num_tickets*silver
            elif category == "gold":
                total=num_tickets*gold
            elif category == "diamond":
                total=num_tickets*diamond
            return total
```

```
C:\Users\Tarun\PycharmProjects\Training\venv\Scripts\python.exe C:\Users\Tarun\PycharmProjects\Training\Assignment\Task2.py
Enter the ticket category (silver,gold,diamond):diamond
Enter the number of Tickets :5
Total cost =  500

Process finished with exit code 0
```



```python
def total_cost(num_tickets,category):
    silver = 25
    gold = 50
    diamond = 100

    if(category in ["silver","gold","diamond"]):
        if num_tickets > 0:
            if category == "silver":
                total=num_tickets*silver
            elif category == "gold":
                total=num_tickets*gold
            elif category == "diamond":
                total=num_tickets*diamond
            return total
```

```
C:\Users\Tarun\PycharmProjects\Training\venv\Scripts\python.exe C:\Users\Tarun\PycharmProjects\Training\Assignment\Task2.py
Enter the ticket category (silver,gold,diamond):silver
Enter the number of Tickets :5
Total cost =  125

Process finished with exit code 0
```