TicketBookingSystem

Assignment 5

SupersetID: 5270707

Name: Shardul Satish Kulkarni

Directory Structure

The project followed a structured directory organization:

1. **Entity/Bean:** Contains entity classes (e.g., `Event`, `Booking`, `Customer`, `Venue`, `Movie`, `Concert`, `Sport`) representing data structures without business logic. Packages seen: `Entity` (Tasks 4-10), `bean` (Task 11).
2. **DAO/Service/Repository:** Includes service interfaces (e.g., `IEventServiceProvider`, `IBookingServiceProvider`, `IBookingSystemRepository`) defining functionalities and implementation classes handling business logic and database interactions. Packages seen: `dao` (Tasks 8-10), `service` and `repository` (Task 11).
3. **Util:** Contains utility classes for database connectivity (`DBConnUtil`, `DBPropertyUtil`) and other helpers like comparators (`EventComparator`). Package seen: `util` (Tasks 10, 11).
4. **Main/App:** Holds the main application class (`TicketBookingSystem`) for user interaction and demonstrating functionalities. Packages seen: `Main` (Tasks 1, 3, 5-10), `app` (Task 11).
5. **Exception:** Contains custom exception classes (`EventNotFoundException`, `InvalidBookingException`). Package seen: `exception` (Tasks 9, 10, 11).

SQL Tasks (Tasks 1-4)

These tasks focused on database design and querying, implemented primarily through SQL scripts.

- **Task 1: Database Design:**
  - Created the `TicketBookingSystem` database (Verified in `Answers Assignment SQL OOPS TicketBookingSystem.pdf`).
  - Developed SQL scripts (`01_create_tables.sql`) to create the `Venue`, `Event`, `Customer`, and `Booking` tables with specified columns, data types, and

initial constraints.

- Established relationships using Primary and Foreign Keys in the table creation scripts (Verified in `01_create_tables.sql`).
- Designed an Entity Relationship Diagram (ERD) conceptually to visualize table relationships.
- Added necessary constraints like `NOT NULL` and `UNIQUE` using `ALTER TABLE` commands in the SQL script (`01_create_tables.sql`) to ensure data integrity.

- **Task 2: Select, Where, Between, AND, LIKE:**
  - Inserted sample records (at least 10) into each table using `02_insert_data.sql`.
  - Implemented various SQL SELECT queries in `03_queries.sql` to:
    - List all events.
    - Select events with available tickets.
    - Find events with names containing "cup" using `LIKE`.
    - Filter events by ticket price range using `BETWEEN`.
    - Filter events by date range using `BETWEEN`.
    - Combine conditions using `AND` to find available "Concert" events.
    - Retrieve users in batches using `LIMIT` and `OFFSET`.
    - Find bookings with more than 4 tickets.
    - Retrieve customers with phone numbers ending in '000' using `LIKE`.
    - Filter events by seat capacity.
    - Select events whose names do not start with 'x', 'y', or 'z' using `NOT LIKE`.
- **Task 3: Aggregate functions, Having, Order By, GroupBy and Joins:**
  - Implemented SQL queries in `03_queries.sql` using aggregate functions, joins, and grouping/ordering clauses to:
    - Calculate average ticket prices per event (`AVG`, `GROUP BY`).
    - Calculate total revenue per event (`SUM`, `JOIN`, `GROUP BY`).

        - Find the event with the highest ticket sales (`SUM` , `JOIN` , `GROUP BY` , `ORDER BY` , `LIMIT`).
        - Calculate total tickets sold per event (`SUM` , `JOIN` , `GROUP BY`).
        - Find events with no ticket sales (using `IS NULL` or checking `booking_id` , potentially using `LEFT JOIN` or subquery).
        - Find the user who booked the most tickets (`SUM` , `JOIN` , `GROUP BY` , `ORDER BY` , `LIMIT`).
        - List events and total tickets sold per month (`SUM` , `JOIN` , `GROUP BY` , `DATE_FORMAT`).
        - Calculate average ticket price per venue (`AVG` , `JOIN` , `GROUP BY`).
        - Calculate total tickets sold per event type (`SUM` , `JOIN` , `GROUP BY`).
        - Calculate total revenue per event per year (`SUM` , `JOIN` , `GROUP BY` , `DATE_FORMAT`).
        - List users who booked multiple distinct events (`COUNT` , `DISTINCT` , `JOIN` , `GROUP BY` , `HAVING`).
        - Calculate total revenue generated per user (`SUM` , `JOIN` , `GROUP BY`).
        - Calculate average ticket price per event category and venue (`AVG` , `JOIN` , `GROUP BY`).
        - List users and their total tickets purchased in the last 30 days (`SUM` , `JOIN` , `WHERE` , `GROUP BY` , date functions).
- **Task 4: Subquery and its types:**
    - Implemented SQL queries in `03_queries.sql` and `practice.sql` using various subquery types:
        - Calculated average ticket price per venue (Scalar Subquery in SELECT).
        - Found events with over 50% tickets sold (Scalar Subquery in WHERE).
        - Calculated total tickets sold per event (Scalar Subquery in SELECT).
        - Found users with no bookings (`NOT EXISTS`).
        - Listed events with no sales (`NOT IN`).
        - Calculated total tickets sold per event type (Subquery in FROM).

- Found events with prices above average (Scalar Subquery in WHERE).
- Calculated total revenue per user (Correlated Subquery in SELECT).
- Listed users booking tickets for a specific venue (`IN` with Subquery in WHERE).
- Calculated total tickets sold per category (used JOIN and GROUP BY, fulfills intent of).
- Found users' bookings per month (used JOIN and GROUP BY, fulfills intent of).

Java Tasks (Tasks 1-11)

These tasks involved implementing the Ticket Booking System logic in Java, progressively adding features like classes, inheritance, abstraction, collections, exceptions, and database connectivity.

- **Task 1 (Control Structure - Conditionals):**
  - Created a basic program (`Task 1/Task1/src/Main/Main.java`) that takes available tickets and requested tickets as input.
  - Used `if-else` statements to check ticket availability and display appropriate messages (`Booking Successful!` or `Ticket Unavailable!`).

```
package Main;

import java.util.Scanner;
public class Main {

    //program to display tickets
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of available tickets: ");
        int availableTickets = scanner.nextInt();

        System.out.print("Enter number of tickets to book: ");
        int noOfBookingTicket = scanner.nextInt();
```

```
        if (availableTickets >= noOfBookingTicket) {
            int remainingTickets = availableTickets - noOfBookingTicket;
            System.out.println("Booking Successful!");
            System.out.println("Remaining Tickets: " + remainingTickets);
        } else {
            System.out.println("Ticket Unavailable! Not enough tickets to fulf
        }

        scanner.close();
    }
}
```

- **Task 2 (Control Structure - Nested Conditionals):**
  - Developed a program (`Task2/Task2/src/TicketBookingNested.java`) to calculate ticket costs based on category ("Silver", "Gold", "Diamond").
  - Used nested `if-else if-else` statements to determine the price based on the selected ticket type and calculate the total cost.

```
import java.util.Scanner;

public class TicketBookingNested {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Available Ticket Categories:");
        System.out.println("1. Silver - ₹300");
        System.out.println("2. Gold   - ₹500");
        System.out.println("3. Diamond - ₹800");

        System.out.print("Enter ticket category (Silver/Gold/Diamond): ");
        String ticketType = sc.nextLine().toLowerCase();

        System.out.print("Enter number of tickets: ");
```

```
        int numberOfTickets = sc.nextInt();

        int base = 0;

        if (numberOfTickets > 0) {
            if (ticketType.equals("silver")) {
                base = 300;
            } else if (ticketType.equals("gold")) {
                base = 500;
            } else if (ticketType.equals("diamond")) {
                base = 800;
            } else {
                System.out.println("Invalid ticket category selected!");
                return;
            }

            int totalCost = base * numberOfTickets;
            System.out.println("Total cost for " + numberOfTickets + " " + ticke
        } else {
            System.out.println("Invalid number of tickets.");
        }

        sc.close();
    }
}
```

- **Task 3 (Control Structure - Looping):**
  - Modified the booking program (`Task3/src/Main.java`) to run repeatedly using a `while(true)` loop.
  - Allowed users to book tickets for different seat types ("Silver", "Gold", "Diamond") multiple times until they enter "quit" to exit.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
        String ticketType;

        System.out.println("Welcome to the Movie Ticket Reservation Syste
        System.out.println("Available Seating Options:");
        System.out.println("1. Silver  - ₹200");
        System.out.println("2. Gold - ₹400");
        System.out.println("3. Diamond     - ₹700");

        while (true) {
            System.out.print("\nSelect seat type (Regular/Premium/VIP) or ty
            ticketType = input.nextLine().toLowerCase();

            if (ticketType.equals("quit")) {
                System.out.println("Thanks for using the Movie Ticket Reserva
                break;
            }

            int ticketPrice;

            if (ticketType.equals("Silver")) {
                ticketPrice = 200;
            } else if (ticketType.equals("Gold")) {
                ticketPrice = 400;
            } else if (ticketType.equals("Diamond")) {
                ticketPrice = 700;
            } else {
                System.out.println("Invalid seat type. Please try again.");
                continue;
            }

            System.out.print("How many seats would you like to reserve? ");
            int seats;

            try {
                seats = Integer.parseInt(input.nextLine());

                if (seats <= 0) {
```

```
                System.out.println("Seat count must be a positive number.")
                continue;
            }

            int totalAmount = ticketPrice * seats;
            System.out.println("Reservation Confirmed!");
            System.out.println("Total amount for " + seats + " " + ticketTyp

        } catch (NumberFormatException e) {
            System.out.println("Invalid input. Please enter a numeric value
        }
    }

    input.close();
  }
}
```

- **Task 4: Class & Object:**
  - Created initial entity classes: `Event`, `Venue`, `Customer`, and `Booking` in the `Entity` package (`Task4/src/Entity/`).
  - Implemented constructors, getters, setters, and display methods (`display_event_details`, `display_venue_details`, `display_customer_details`) for these classes.
  - Included methods in `Event` for `calculate_total_revenue`, `getBookedNoOfTickets`, `book_tickets`, and `cancel_booking`.
  - The `Booking` class included methods like `calculate_booking_cost`, `book_tickets`, `cancel_booking`, `getAvailableNoOfTickets`, and `getEventDetails`, operating on an associated `Event` object. A `main` method within the `Booking` class demonstrated these operations.

```
package Entity;
class Venue {
    private String venue_name;
    private String address;

    public Venue() { }
```

```
    public Venue(String venue_name, String address) {
        this.venue_name = venue_name;
        this.address = address;
    }

    public String getVenue_name() {
        return venue_name;
    }

    public void setVenue_name(String venue_name) {
        this.venue_name = venue_name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public void display_venue_details() {
        System.out.println("Venue Name: " + venue_name);
        System.out.println("Address: " + address);
    }
}


package Entity;
import java.time.LocalDate;
import java.time.LocalTime;

enum EventType {
    Movie, Sports, Concert
}

class Event {
    private String event_name;
```

```
    private LocalDate event_date;
    private LocalTime event_time;
    private String venue_name;
    private int total_seats;
    private int available_seats;
    private double ticket_price;
    private EventType event_type;

    public Event() { }

    public Event(String event_name, LocalDate event_date, LocalTime even
        this.event_name = event_name;
        this.event_date = event_date;
        this.event_time = event_time;
        this.venue_name = venue_name;
        this.total_seats = total_seats;
        this.available_seats = total_seats;
        this.ticket_price = ticket_price;
        this.event_type = event_type;
    }

    public String getEvent_name() {
        return event_name;
    }

    public void setEvent_name(String event_name) {
        this.event_name = event_name;
    }

    public LocalDate getEvent_date() {
        return event_date;
    }

    public void setEvent_date(LocalDate event_date) {
        this.event_date = event_date;
    }

    public LocalTime getEvent_time() {
```

```
    return event_time;
}

public void setEvent_time(LocalTime event_time) {
    this.event_time = event_time;
}

public String getVenue_name() {
    return venue_name;
}

public void setVenue_name(String venue_name) {
    this.venue_name = venue_name;
}

public int getTotal_seats() {
    return total_seats;
}

public void setTotal_seats(int total_seats) {
    this.total_seats = total_seats;
}

public int getAvailable_seats() {
    return available_seats;
}

public void setAvailable_seats(int available_seats) {
    this.available_seats = available_seats;
}

public double getTicket_price() {
    return ticket_price;
}

public void setTicket_price(double ticket_price) {
    this.ticket_price = ticket_price;
}
```

```
public EventType getEvent_type() {
    return event_type;
}

public void setEvent_type(EventType event_type) {
    this.event_type = event_type;
}

public double calculate_total_revenue() {
    int ticketsSold = total_seats - available_seats;
    return ticketsSold * ticket_price;
}

public int getBookedNoOfTickets() {
    return total_seats - available_seats;
}

public boolean book_tickets(int num_tickets) {
    if(num_tickets <= available_seats) {
        available_seats -= num_tickets;
        return true;
    }
    return false;
}

public boolean cancel_booking(int num_tickets) {
    if(available_seats + num_tickets <= total_seats) {
        available_seats += num_tickets;
        return true;
    }
    return false;
}

public void display_event_details() {
    System.out.println("Event Name: " + event_name);
    System.out.println("Event Date: " + event_date);
    System.out.println("Event Time: " + event_time);
```

```
        System.out.println("Venue Name: " + venue_name);
        System.out.println("Total Seats: " + total_seats);
        System.out.println("Available Seats: " + available_seats);
        System.out.println("Ticket Price: " + ticket_price);
        System.out.println("Event Type: " + event_type);
    }
}



package Entity;
class Customer {
    private String customer_name;
    private String email;
    private String phone_number;

    public Customer() { }

    public Customer(String customer_name, String email, String phone_num
        this.customer_name = customer_name;
        this.email = email;
        this.phone_number = phone_number;
    }

    public String getCustomer_name() {
        return customer_name;
    }

    public void setCustomer_name(String customer_name) {
        this.customer_name = customer_name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
```

```
    }

    public String getPhone_number() {
        return phone_number;
    }

    public void setPhone_number(String phone_number) {
        this.phone_number = phone_number;
    }

    public void display_customer_details() {
        System.out.println("Customer Name: " + customer_name);
        System.out.println("Email: " + email);
        System.out.println("Phone Number: " + phone_number);
    }
}

package Entity;

import java.time.LocalDate;
import java.time.LocalTime;

class Booking
{
    private Event event;
    private double total_cost;

    public Booking() { }

    public Booking(Event event) {
        this.event = event;
    }

    public Event getEvent() {
        return event;
    }

    public void setEvent(Event event) {
```

```
    this.event = event;
}

public double getTotal_cost() {
    return total_cost;
}

public void setTotal_cost(double total_cost) {
    this.total_cost = total_cost;
}

public double calculate_booking_cost(int num_tickets) {
    total_cost = num_tickets * event.getTicket_price();
    return total_cost;
}

public boolean book_tickets(int num_tickets) {
    return event.book_tickets(num_tickets);
}

public boolean cancel_booking(int num_tickets) {
    return event.cancel_booking(num_tickets);
}

public int getAvailableNoOfTickets() {
    return event.getAvailable_seats();
}

public void getEventDetails() {
    event.display_event_details();
}

public static void main(String[] args) {
    Event sampleEvent = new Event("Rock Concert", LocalDate.of(2025,
    Venue sampleVenue = new Venue("Stadium Arena", "123 Main St, Cit
    Customer sampleCustomer = new Customer("Alice Johnson", "alice@
    Booking bookingSystem = new Booking(sampleEvent);
```

```
        sampleEvent.display_event_details();
        sampleVenue.display_venue_details();
        sampleCustomer.display_customer_details();

        int ticketsToBook = 50;
        if(bookingSystem.book_tickets(ticketsToBook)) {
            System.out.println("Booked " + ticketsToBook + " tickets.");
        } else {
            System.out.println("Booking failed. Not enough tickets available.");
        }

        System.out.println("Available tickets: " + bookingSystem.getAvailable
        System.out.println("Booking cost: " + bookingSystem.calculate_booki
        System.out.println("Total revenue: " + sampleEvent.calculate_total_re

        int ticketsToCancel = 10;
        if(bookingSystem.cancel_booking(ticketsToCancel)) {
            System.out.println("Cancelled " + ticketsToCancel + " tickets.");
        } else {
            System.out.println("Cancellation failed.");
        }

        System.out.println("Available tickets after cancellation: " + bookingSy
        bookingSystem.getEventDetails();
    }
}
```

- **Task 5: Inheritance and Polymorphism:**
  - Created subclasses `Movie`, `Concert`, and `Sports` inheriting from the `Event` class in the `Entity` package (`Task5/src/Entity/`).
  - Added specific attributes (e.g., `Genre`, `ActorName` for Movie; `artist`, `concertType` for Concert; `sportName`, `teamsName` for Sports) and overridden `display_event_details` methods in subclasses.
  - Implemented a `TicketBookingSystem` class (`Task5/src/TicketBookingSystem.java`) with methods:
    - `create_event`: Instantiated the correct subclass based on `event_type`.

- `display_event_details`: Called the appropriate `display_event_details` method polymorphically.
- `book_tickets`: Checked availability and updated seats in the event object.
- `cancel_tickets`: Updated available seats in the event object.

- The `main` method provided a menu-driven interface for user interaction.

```
package Entity;

import java.time.LocalDate;
import java.time.LocalTime;

public class Movie extends Event {

    private String Genre;
    private String ActorName;
    private String ActressName;

    public Movie(){}

    public Movie(String event_name, LocalDate event_date, LocalTime even
        super(event_name, event_date, event_time, venue_name, total_seats,
        this.Genre = Genre;
        this.ActorName = ActorName;
        this.ActressName = ActressName;
    }

    public String getGenre() {
        return Genre;
    }

    public void setGenre(String genre) {
        Genre = genre;
    }

    public String getActorName() {
        return ActorName;
```

```
    }

    public void setActorName(String actorName) {
        ActorName = actorName;
    }

    public String getActressName() {
        return ActressName;
    }

    public void setActressName(String actressName) {
        ActressName = actressName;
    }

    public void display_Movie_details(){
        super.display_event_details();
        System.out.println("Genre: " + Genre);
        System.out.println("ActorName: " + ActorName);
        System.out.println("ActressName: " + ActressName);
    }
}

package Entity;


import java.time.LocalDate;
import java.time.LocalTime;

public class Sports extends Event {
    private String sportName;
    private String teamsName;

    public Sports(){}

    public Sports(String event_name, LocalDate event_date, LocalTime even
        super(event_name, event_date, event_time, venue_name, total_seats,
        this.sportName = sportName;
        this.teamsName = teamsName;
```

```
    }

    public void setTeamsName(String teamsName) {
        this.teamsName = teamsName;
    }

    public String getTeamsName() {
        return teamsName;
    }

    public void setSportName(String sportName) {
        this.sportName = sportName;
    }

    public String getSportName() {
        return sportName;
    }

    public void display_sport_details(){
        super.display_event_details();
        System.out.println("SportName: "+ sportName);
        System.out.println("TeamsName: "+ teamsName);
    }
}

package Entity;

import java.time.LocalDate;
import java.time.LocalTime;

public class Concert extends Event {
    private String artist;
    private String concertType;

    public Concert(){}

    public Concert(String event_name, LocalDate event_date, LocalTime ev
        super(event_name, event_date, event_time, venue_name, total_seats,
```

```
        this.artist = artist;
        this.concertType = concertType;
    }

    public void setArtist(String artist) {
        this.artist = artist;
    }

    public String getArtist() {
        return artist;
    }

    public void setConcertType(String concertType) {
        this.concertType = concertType;
    }

    public String getConcertType() {
        return concertType;
    }

    public void display_concert_details(){
        super.display_event_details();
        System.out.println("Artist: " + artist);
        System.out.println("ConcertType: " + concertType);
    }
}
```

```
import Entity.*;
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TicketBookingSystem {

   List<Event> events = new ArrayList<>();
    Scanner sc = new Scanner(System.in);
```

```
public Event create_event(String event_name, LocalDate event_date, LocalT

    Event event = null;

    if(event_type.equalsIgnoreCase("Movie")){
        System.out.println("Enter Movie Genre: ");
        String genre = sc.next();
        sc.nextLine();
        System.out.println("Enter Actor Name: ");
        String ActorName = sc.nextLine();
        System.out.println("Enter ActressName: ");
        String ActressName = sc.nextLine();

      event = new Movie(event_name, event_date, event_time, venue_name, t
    }

    if(event_type.equalsIgnoreCase("Concert")){
        System.out.println("Enter Artist: ");
        String artist = sc.nextLine();
        System.out.println("Enter ConcertType");
        String concertType = sc.nextLine();

        event = new Concert(event_name, event_date, event_time, venue_nam
    }

    if(event_type.equalsIgnoreCase("Sports"))
    {
        System.out.println("Enter SportName: ");
        String sportName = sc.nextLine();
        System.out.println("Enter TeamsName: ");
        String teamsName = sc.nextLine();

        event = new Sports(event_name, event_date, event_time, venue_name,

    }
    events.add(event);
    return event;
```

```
    }

    public void display_event_details(Event event){
        event.display_event_details();
    }

    public double book_tickets(Event event, int num_tickets){
        double totalCost = 0.00;
        if(event.getAvailable_seats() < num_tickets) {
            System.out.println("Sorry the REMAINING TICKETS are only: "+ event.g
        }
        else{
            event.setAvailable_seats(event.getAvailable_seats() - num_tickets);
            totalCost = event.getTicket_price() * num_tickets;
            System.out.println("Booked Successfully");
        }

        return totalCost;

    }

    public void cancel_tickets(Event event, int num_tickets){
        if(num_tickets > event.getAvailable_seats())
        {
            System.out.println("ERROR");
        }
        else{
            event.setAvailable_seats(event.getAvailable_seats() + num_tickets);
            System.out.println("Canceled Successfully");
        }
    }

    private static LocalDate ConvertDate(int year, int month, int day){
        return LocalDate.of(year, month, day);
    }

    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();
```

```
        Scanner sc = system.sc;

        while (true) {
            System.out.println("\n1. Create Event");
            System.out.println("2. View Event Details");
            System.out.println("3. Book Tickets");
            System.out.println("4. Cancel Tickets");
            System.out.println("0. Exit");
            System.out.print("Enter choice: ");
            int choice = Integer.parseInt(sc.nextLine());

            switch (choice) {
                case 1 -> {
                    System.out.print("Enter event name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter date (yyyy-mm-dd): ");
                    System.out.println("Enter Year: ");
                    int year = sc.nextInt();
                    System.out.println("Enter Month: ");
                    int month = sc.nextInt();
                    System.out.println("Enter Day: ");
                    int day = sc.nextInt();
                    LocalDate date = ConvertDate(year, month, day);
                    System.out.print("Enter time (HH:mm): ");
                    System.out.println("Enter Hour: ");
                    int Hours = sc.nextInt();
                    System.out.println("Enter Minutes:");
                    int Minutes = sc.nextInt();
                    LocalTime time = LocalTime.of(Hours, Minutes);
                    sc.nextLine();
                    System.out.print("Enter total seats: ");
                    int seats = sc.nextInt();
                    System.out.print("Enter ticket price: ");
                    double price = sc.nextDouble();
                    sc.nextLine();
                    System.out.print("Enter event type (Movie, Concert, Sports): ");
                    String type = sc.nextLine();
                    System.out.println("Enter Venue Name: ");
```

```
            String venue = sc.nextLine();
            sc.nextLine();
            system.create_event(name, date, time, seats, price, type, venue);

        }
        case 2 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEvent_
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {
                system.display_event_details(system.events.get(idx - 1));
            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 3 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEvent_
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {
                System.out.print("Enter number of tickets to book: ");
                int tickets = Integer.parseInt(sc.nextLine());
                system.book_tickets(system.events.get(idx - 1), tickets);
            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 4 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEvent_
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {
```

```
                    System.out.print("Enter number of tickets to cancel: ");
                    int tickets = Integer.parseInt(sc.nextLine());
                    system.cancel_tickets(system.events.get(idx - 1), tickets);
                } else {
                    System.out.println("Invalid selection.");
                }
            }
            case 0 -> {
                System.out.println("Exiting...");
                return;
            }
            default -> System.out.println("Invalid choice.");
        }
    }
}

}
```

- **Task 6: Abstraction:**
  - Converted the `Event` class (`Task6/src/Entity/Event.java`) into an abstract class with an abstract `display_event_details` method.
  - Concrete subclasses (`Movie`, `Concert`, `Sport` in `Task6/src/Entity/`) inherited from the abstract `Event` and provided implementations for `display_event_details`.
  - Created an abstract `BookingSystem` class (`Task6/src/Main/BookingSystem.java`) defining abstract methods like `create_event`, `display_event_details`, `book_tickets`, `cancel_tickets`.
  - The concrete `TicketBookingSystem` class (`Task6/src/Main/TicketBookingSystem.java`) extended `BookingSystem`, implemented its abstract methods, and managed a list of `Event` objects.
  - A `main` method provided the user interface.

```
package Entity;
import java.time.LocalDate;
import java.time.LocalTime;
```

```
public abstract class Event {
    private String event_name;
    private LocalDate event_date;
    private LocalTime event_time;
    private String venue_name;
    private int total_seats;
    private int available_seats;
    private double ticket_price;
    private String event_type;

    public Event() { }

    public Event(String event_name, LocalDate event_date, LocalTime even
        this.event_name = event_name;
        this.event_date = event_date;
        this.event_time = event_time;
        this.venue_name = venue_name;
        this.total_seats = total_seats;
        this.available_seats = total_seats;
        this.ticket_price = ticket_price;
        this.event_type = event_type;
    }

    public String getEvent_name() {
        return event_name;
    }

    public void setEvent_name(String event_name) {
        this.event_name = event_name;
    }

    public LocalDate getEvent_date() {
        return event_date;
    }

    public void setEvent_date(LocalDate event_date) {
```

```
    this.event_date = event_date;
}

public LocalTime getEvent_time() {
    return event_time;
}

public void setEvent_time(LocalTime event_time) {
    this.event_time = event_time;
}

public String getVenue_name() {
    return venue_name;
}

public void setVenue_name(String venue_name) {
    this.venue_name = venue_name;
}

public int getTotal_seats() {
    return total_seats;
}

public void setTotal_seats(int total_seats) {
    this.total_seats = total_seats;
}

public int getAvailable_seats() {
    return available_seats;
}

public void setAvailable_seats(int available_seats) {
    this.available_seats = available_seats;
}

public double getTicket_price() {
    return ticket_price;
}
```

```
public void setTicket_price(double ticket_price) {
    this.ticket_price = ticket_price;
}

public String getEvent_type() {
    return event_type;
}

public void setEvent_type(String event_type) {
    this.event_type = event_type;
}



public double calculate_total_revenue() {
    int ticketsSold = total_seats - available_seats;
    return ticketsSold * ticket_price;
}

public int getBookedNoOfTickets() {
    return total_seats - available_seats;
}

public boolean book_tickets(int num_tickets) {
    if(num_tickets <= available_seats) {
        available_seats -= num_tickets;
        return true;
    }
    return false;
}

public boolean cancel_booking(int num_tickets) {
    if(available_seats + num_tickets <= total_seats) {
        available_seats += num_tickets;
        return true;
    }
    return false;
```

```
    }

    public abstract void display_event_details();
}


package Entity;

import java.time.LocalDate;
import java.time.LocalTime;

public class Movie extends Event {

    private String Genre;
    private String ActorName;
    private String ActressName;

    public Movie(){}

    public Movie(String event_name, LocalDate event_date, LocalTime even
        super(event_name, event_date, event_time, venue_name, total_seats,
        this.Genre = Genre;
        this.ActorName = ActorName;
        this.ActressName = ActressName;
    }

    public String getGenre() {
        return Genre;
    }

    public void setGenre(String genre) {
        Genre = genre;
    }

    public String getActorName() {
        return ActorName;
    }
```

```
    public void setActorName(String actorName) {
        ActorName = actorName;
    }

    public String getActressName() {
        return ActressName;
    }

    public void setActressName(String actressName) {
        ActressName = actressName;
    }

    @Override
    public void display_event_details(){
        System.out.println("Event Name: " + getEvent_name());
        System.out.println("Event Date: " + getEvent_date());
        System.out.println("Event Time: " + getEvent_time());
        System.out.println("Venue Name: " + getVenue_name());
        System.out.println("Total Seats: " + getTotal_seats());
        System.out.println("Available Seats: " + getAvailable_seats());
        System.out.println("Ticket Price: " + getTicket_price());
        System.out.println("Event Type: " + getEvent_type());
        System.out.println("Genre: " + Genre);
        System.out.println("ActorName: " + ActorName);
        System.out.println("ActressName: " + ActressName);
    }
}

package Entity;


import java.time.LocalDate;
import java.time.LocalTime;

public class Sport extends Event {
    private String sportName;
    private String teamsName;
```

```
    public Sport(){}

    public Sport(String event_name, LocalDate event_date, LocalTime event
        super(event_name, event_date, event_time, venue_name, total_seats,
        this.sportName = sportName;
        this.teamsName = teamsName;
    }

    public void setTeamsName(String teamsName) {
        this.teamsName = teamsName;
    }

    public String getTeamsName() {
        return teamsName;
    }

    public void setSportName(String sportName) {
        this.sportName = sportName;
    }

    public String getSportName() {
        return sportName;
    }

    @Override
    public void display_event_details(){
        System.out.println("Event Name: " + getEvent_name());
        System.out.println("Event Date: " + getEvent_date());
        System.out.println("Event Time: " + getEvent_time());
        System.out.println("Venue Name: " + getVenue_name());
        System.out.println("Total Seats: " + getTotal_seats());
        System.out.println("Available Seats: " + getAvailable_seats());
        System.out.println("Ticket Price: " + getTicket_price());
        System.out.println("Event Type: " + getEvent_type());
        System.out.println("SportName: "+ sportName);
        System.out.println("TeamsName: "+ teamsName);
    }
}
```

```
package Entity;

import java.time.LocalDate;
import java.time.LocalTime;

public class Concert extends Event {
    private String artist;
    private String concertType;

    public Concert(){}

    public Concert(String event_name, LocalDate event_date, LocalTime ev
        super(event_name, event_date, event_time, venue_name, total_seats,
        this.artist = artist;
        this.concertType = concertType;
    }

    public void setArtist(String artist) {
        this.artist = artist;
    }

    public String getArtist() {
        return artist;
    }

    public void setConcertType(String concertType) {
        this.concertType = concertType;
    }

    public String getConcertType() {
        return concertType;
    }

    @Override
    public void display_event_details(){
        System.out.println("Event Name: " + getEvent_name());
        System.out.println("Event Date: " + getEvent_date());
```

```
        System.out.println("Event Time: " + getEvent_time());
        System.out.println("Venue Name: " + getVenue_name());
        System.out.println("Total Seats: " + getTotal_seats());
        System.out.println("Available Seats: " + getAvailable_seats());
        System.out.println("Ticket Price: " + getTicket_price());
        System.out.println("Event Type: " + getEvent_type());
        System.out.println("Artist: " + artist);
        System.out.println("ConcertType: " + concertType);
    }
}
```

```
package Main;

import Entity.Event;

import java.time.LocalDate;
import java.time.LocalTime;

public abstract class BookingSystem {

    public abstract Event create_event(String event_name, LocalDate event_dat
    public abstract void display_event_details(Event event);
    public abstract double book_tickets(Event event, int num_tickets);
    public abstract void cancel_tickets(Event event, int num_tickets);


}

package Main;
import Entity.*;



import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
```

```
public class TicketBookingSystem extends BookingSystem {

    List<Event> events = new ArrayList<>();
    Scanner sc = new Scanner(System.in);

    @Override
    public Event create_event(String event_name, LocalDate event_date, LocalT

        Event event = null;

        if(event_type.equalsIgnoreCase("Movie")){
            System.out.println("Enter Movie Genre: ");
            String genre = sc.next();
            sc.nextLine();
            System.out.println("Enter Actor Name: ");
            String ActorName = sc.nextLine();
            System.out.println("Enter ActressName: ");
            String ActressName = sc.nextLine();

            event = new Movie(event_name, event_date, event_time, venue_name,
        }

        if(event_type.equalsIgnoreCase("Concert")){
            System.out.println("Enter Artist: ");
            String artist = sc.nextLine();
            System.out.println("Enter ConcertType");
            String concertType = sc.nextLine();

            event = new Concert(event_name, event_date, event_time, venue_nam
        }

        if(event_type.equalsIgnoreCase("Sports"))
        {
            System.out.println("Enter SportName: ");
            String sportName = sc.nextLine();
            System.out.println("Enter TeamsName: ");
            String teamsName = sc.nextLine();
```

```
        event = new Sport(event_name, event_date, event_time, venue_name, t

    }
    events.add(event);
    return event;
}

@Override
public void display_event_details(Event event){


    event.display_event_details();




}

@Override
public double book_tickets(Event event, int num_tickets){
    double totalCost = 0.00;
    if(event.getAvailable_seats() < num_tickets) {
        System.out.println("Sorry the REMAINING TICKETS are only: "+ event.g
    }
    else{
        event.setAvailable_seats(event.getAvailable_seats() - num_tickets);
        totalCost = event.getTicket_price() * num_tickets;
        System.out.println("Booked Successfully");
    }

    return totalCost;

}

@Override
public void cancel_tickets(Event event, int num_tickets){
    if(num_tickets > event.getAvailable_seats())
    {
```

```
            System.out.println("ERROR");
        }
        else{
            event.setAvailable_seats(event.getAvailable_seats() + num_tickets);
            System.out.println("Canceled Successfully");
        }
    }

    private static LocalDate ConvertDate(int year, int month, int day){
        return LocalDate.of(year, month, day);
    }

    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();
        Scanner sc = system.sc;

        while (true) {
            System.out.println("\n1. Create Event");
            System.out.println("2. View Event Details");
            System.out.println("3. Book Tickets");
            System.out.println("4. Cancel Tickets");
            System.out.println("0. Exit");
            System.out.print("Enter choice: ");
            int choice = Integer.parseInt(sc.nextLine());

            switch (choice) {
                case 1 -> {
                    System.out.print("Enter event name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter date (yyyy-mm-dd): ");
                    System.out.println("Enter Year: ");
                    int year = sc.nextInt();
                    System.out.println("Enter Month: ");
                    int month = sc.nextInt();
                    System.out.println("Enter Day: ");
                    int day = sc.nextInt();
                    LocalDate date = ConvertDate(year, month, day);
                    System.out.print("Enter time (HH:mm): ");
```

```
            System.out.println("Enter Hour: ");
            int Hours = sc.nextInt();
            System.out.println("Enter Minutes:");
            int Minutes = sc.nextInt();
            LocalTime time = LocalTime.of(Hours, Minutes);
            sc.nextLine();
            System.out.print("Enter total seats: ");
            int seats = sc.nextInt();
            System.out.print("Enter ticket price: ");
            double price = sc.nextDouble();
            sc.nextLine();
            System.out.print("Enter event type (Movie, Concert, Sports): ");
            String type = sc.nextLine();
            System.out.println("Enter Venue Name: ");
            String venue = sc.nextLine();
            sc.nextLine();
            system.create_event(name, date, time, seats, price, type, venue);

        }
        case 2 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEvent_
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {
                system.display_event_details(system.events.get(idx - 1));
            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 3 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEvent_
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {
```

```
                System.out.print("Enter number of tickets to book: ");
                int tickets = Integer.parseInt(sc.nextLine());
                system.book_tickets(system.events.get(idx - 1), tickets);
            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 4 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEvent_
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {
                System.out.print("Enter number of tickets to cancel: ");
                int tickets = Integer.parseInt(sc.nextLine());
                system.cancel_tickets(system.events.get(idx - 1), tickets);
            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 0 -> {
            System.out.println("Exiting...");
            return;
        }
        default -> System.out.println("Invalid choice.");
      }
    }
  }

}
```

- **Task 7: Has-A Relation / Association:**
  - Modified the `Event` class (`Task7/src/Entity/Event.java`) to contain a `Venue` object reference (Composition/Aggregation) instead of just the venue name. Created `Venue` class (`Task7/src/Entity/Venue.java`).

- Updated the `Booking` class (`Task7/src/Entity/Booking.java`) to hold a `List<Customer>` and an `Event` reference. Created `Customer` class (`Task7/src/Entity/Customer.java`).
- Modified the `TicketBookingSystem` class (`Task7/src/Main/TicketBookingSystem.java`):
  - `create_event` now accepted a `Venue` object.
  - `book_tickets` now accepted a `List<Customer>`. It created `Booking` objects and managed customer details.
- Implemented `cancel_tickets` based on the event and number of tickets (as per code, though assignment task 8 mentions by ID).
- The `main` method provided the user interface.

```
package Entity;
import java.time.LocalDate;
import java.time.LocalTime;



public abstract class Event {
    private String event_name;
    private LocalDate event_date;
    private LocalTime event_time;
    private Venue venue;
    private int total_seats;
    private int available_seats;
    private double ticket_price;
    private String event_type;

    public Event() { }

    public Event(String event_name, LocalDate event_date, LocalTime even
        this.event_name = event_name;
        this.event_date = event_date;
        this.event_time = event_time;
        this.venue = venue;
        this.total_seats = total_seats;
        this.available_seats = total_seats;
        this.ticket_price = ticket_price;
```

```
    this.event_type = event_type;
}

public String getEvent_name() {
    return event_name;
}

public void setEvent_name(String event_name) {
    this.event_name = event_name;
}

public LocalDate getEvent_date() {
    return event_date;
}

public void setEvent_date(LocalDate event_date) {
    this.event_date = event_date;
}

public LocalTime getEvent_time() {
    return event_time;
}

public void setEvent_time(LocalTime event_time) {
    this.event_time = event_time;
}

public Venue getVenue() {
    return venue;
}

public void setVenue(Venue venue) {
    this.venue = venue;
}

public int getTotal_seats() {
    return total_seats;
}
```

```
public void setTotal_seats(int total_seats) {
    this.total_seats = total_seats;
}

public int getAvailable_seats() {
    return available_seats;
}

public void setAvailable_seats(int available_seats) {
    this.available_seats = available_seats;
}

public double getTicket_price() {
    return ticket_price;
}

public void setTicket_price(double ticket_price) {
    this.ticket_price = ticket_price;
}

public String getEvent_type() {
    return event_type;
}

public void setEvent_type(String event_type) {
    this.event_type = event_type;
}



public double calculate_total_revenue() {
    int ticketsSold = total_seats - available_seats;
    return ticketsSold * ticket_price;
}

public int getBookedNoOfTickets() {
    return total_seats - available_seats;
```

```
    }

    public boolean book_tickets(int num_tickets) {
        if(num_tickets <= available_seats) {
            available_seats -= num_tickets;
            return true;
        }
        return false;
    }

    public boolean cancel_booking(int num_tickets) {
        if(available_seats + num_tickets <= total_seats) {
            available_seats += num_tickets;
            return true;
        }
        return false;
    }

    public abstract void display_event_details();
}


package Entity;

import Entity.Customer;
import Entity.Event;

import java.time.LocalDate;
import java.util.List;

public class Booking {
    private static int nextBookingId = 1;
    private List<Customer> customers;
    private int bookingId;
    private Event event;
    private int num_tickets;
    private double total_cost;
    private LocalDate booking_date;
```

```
public Booking(){

}
public Booking(Event event, List<Customer> customers, int num_tickets
   this.bookingId = nextBookingId++;
   this.event = event;
   this.customers = customers;
   this.num_tickets = num_tickets;
   this.total_cost = total_cost;
   this.booking_date = booking_date;
}

public void setBookingId(int bookingId) {

   this.bookingId = bookingId;
}

public void setCustomers(List<Customer> customers) {
   this.customers = customers;
}

public static void setNextBookingId(int nextBookingId) {

   Booking.nextBookingId = nextBookingId;
}

public void setBooking_date(
       LocalDate booking_date) {
   this.booking_date = booking_date;
}

public void setEvent(Event event) {

   this.event = event;
}

public void setNum_tickets(int num_tickets) {
```

```
    this.num_tickets = num_tickets;
}

public void setTotal_cost(double total_cost) {

    this.total_cost = total_cost;
}

public Event getEvent() {

    return event;
}

public int getNum_tickets() {

    return num_tickets;
}

public double getTotal_cost() {

    return total_cost;
}

public LocalDate getBooking_date() {

    return booking_date;
}

public void display_booking_details(){

    if(customers.isEmpty()){
        System.out.println("No bookings");
    }
    else{
        System.out.println("Customers: ");
        for(Customer c: customers) {
            System.out.println(c);
        }
```

```
        }

        if(event == null){
            System.out.println("No events");
        }
        else{
            System.out.println(event);
        }

        System.out.println("Num Tickets: " + num_tickets);
        System.out.println("TotalCOst: " + total_cost);
        System.out.println("BookingDate: "+ booking_date);

    }

    @Override
    public String toString(){
        StringBuilder sb = new StringBuilder();
        if(customers == null || customers.isEmpty()){
            sb.append("No bookings\n");
        } else {
            sb.append("Customers:\n");
            for(Customer c : customers){
                sb.append(c.toString()).append("\n");
            }
        }
        if(event == null){
            sb.append("No events\n");
        } else {
            sb.append(event.toString()).append("\n");
        }
        sb.append("Num Tickets: ").append(num_tickets).append("\n");
        sb.append("Total Cost: ").append(total_cost).append("\n");
        sb.append("BookingDate: ").append(booking_date);
        return sb.toString();
    }

}
```

```
package Main;
import Entity.*;



import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TicketBookingSystem extends BookingSystem {

    List<Event> events = new ArrayList<>();
    List<Booking> bookings = new ArrayList<>();

    Scanner sc = new Scanner(System.in);
    static int bookingCounter = 1;
    @Override
    public Event create_event(String event_name, LocalDate event_date, Lo

        Event event = null;

        if(event_type.equalsIgnoreCase("Movie")){
            System.out.println("Enter Movie Genre: ");
            String genre = sc.next();
            sc.nextLine();
            System.out.println("Enter Actor Name: ");
            String ActorName = sc.nextLine();
            System.out.println("Enter ActressName: ");
            String ActressName = sc.nextLine();

            event = new Movie(event_name, event_date, event_time, venue, tot
        }

        if(event_type.equalsIgnoreCase("Concert")){
            System.out.println("Enter Artist: ");
            String artist = sc.nextLine();
```

```
            System.out.println("Enter ConcertType");
            String concertType = sc.nextLine();

            event = new Concert(event_name, event_date, event_time, venue, t
        }

        if(event_type.equalsIgnoreCase("Sports"))
        {
            System.out.println("Enter SportName: ");
            String sportName = sc.nextLine();
            System.out.println("Enter TeamsName: ");
            String teamsName = sc.nextLine();

            event = new Sport(event_name, event_date, event_time, venue, tota

        }
        events.add(event);
        return event;
    }

    @Override
    public void display_event_details(Event event){


        event.display_event_details();




    }

    @Override
    public double book_tickets(Event event, int num_tickets, List<Customer>
        double totalCost = 0.00;

        if(event.getAvailable_seats() < num_tickets) {
            System.out.println("Sorry the REMAINING TICKETS are only: "+ eve
        }
        else{
```

```
            event.setAvailable_seats(event.getAvailable_seats() - num_tickets);
            totalCost = event.getTicket_price() * num_tickets;
            Booking booking = new Booking();
            booking.setCustomers(customers);
            booking.setBooking_date(LocalDate.now());
            booking.setBookingId(bookingCounter++);
            booking.setTotal_cost(totalCost);
            booking.setNum_tickets(num_tickets);
            bookings = new ArrayList<>();
            bookings.add(booking);
            System.out.println("Booking successful");
            return totalCost;
        }

        return totalCost;

    }

    @Override
    public void cancel_tickets(Event event, int num_tickets){
        if(num_tickets > event.getAvailable_seats())
        {
            System.out.println("ERROR");
        }
        else{
            event.setAvailable_seats(event.getAvailable_seats() + num_tickets)
            System.out.println("Canceled Successfully");
        }
    }

    private static LocalDate ConvertDate(int year, int month, int day){
        return LocalDate.of(year, month, day);
    }

    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();
        Scanner sc = system.sc;
```

```
        while (true) {
            System.out.println("\n1. Create Event");
            System.out.println("2. View Event Details");
            System.out.println("3. Book Tickets");
            System.out.println("4. Cancel Tickets");
            System.out.println("0. Exit");
            System.out.print("Enter choice: ");
            int choice = Integer.parseInt(sc.nextLine());

            switch (choice) {
                case 1 -> {
                    System.out.print("Enter event name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter date (yyyy-mm-dd): ");
                    System.out.println("Enter Year: ");
                    int year = sc.nextInt();
                    System.out.println("Enter Month: ");
                    int month = sc.nextInt();
                    System.out.println("Enter Day: ");
                    int day = sc.nextInt();
                    LocalDate date = ConvertDate(year, month, day);
                    System.out.print("Enter time (HH:mm): ");
                    System.out.println("Enter Hour: ");
                    int Hours = sc.nextInt();
                    System.out.println("Enter Minutes:");
                    int Minutes = sc.nextInt();
                    LocalTime time = LocalTime.of(Hours, Minutes);
                    sc.nextLine();
                    System.out.print("Enter total seats: ");
                    int seats = sc.nextInt();
                    System.out.print("Enter ticket price: ");
                    double price = sc.nextDouble();
                    sc.nextLine();
                    System.out.print("Enter event type (Movie, Concert, Sports): "
                    String type = sc.nextLine();
                    System.out.println("Enter Venue Name: ");
                    String venueName = sc.nextLine();
                    sc.nextLine();
```

```
            System.out.println("Enter Venue Address: ");
            String addressV = sc.nextLine();
            Venue venue = new Venue(venueName, addressV);
            system.create_event(name, date, time, seats, price, type, venu

        }
        case 2 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEv
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {
                system.display_event_details(system.events.get(idx - 1));
            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 3 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEv
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());

            if (idx >= 1 && idx <= system.events.size()) {
                System.out.print("Enter number of tickets to book: ");
                int tickets = Integer.parseInt(sc.nextLine());
                List<Customer> customers = new ArrayList<>();
                for(int i=0; i<tickets; i++)
                {
                    System.out.println("Enter Your name: ");
                    String name = sc.nextLine();
                    System.out.println("Enter email: ");
                    String email = sc.nextLine();
                    System.out.println("Enter Phone Number: ");
                    String phone = sc.next();
                    sc.nextLine();
```

```
                    Customer customer = new Customer(name, email, phone
                    customers.add(customer);
                }

                system.book_tickets(system.events.get(idx - 1), tickets, cus
            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 4 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEv
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {
                System.out.print("Enter number of tickets to cancel: ");
                int tickets = Integer.parseInt(sc.nextLine());
                system.cancel_tickets(system.events.get(idx - 1), tickets);
            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 0 -> {
            System.out.println("Exiting...");
            return;
        }
        default -> System.out.println("Invalid choice.");
      }
    }
  }

}
```

- **Task 8: Interface/Abstract Class, Single Inheritance, Static Variable:**
  - Defined interfaces `IEventServiceProvider` and `IBookingServiceProvider` in the `dao` package (`Task8/src/dao/`) specifying the contract for event and booking operations.

- Implemented these interfaces with `EventServiceProviderImpl` and `BookingSystemServiceProviderImpl` classes in the `dao` package (`Task8/src/dao/`). (Note: Code showed separate implementations, not inheritance between them as suggested in the assignment).
- The `TicketBookingSystem` main class (`Task8/src/Main/TicketBookingSystem.java`) utilized instances of these service implementation classes to perform operations. Static variables like `nextBookingId` were used in `Booking.java`.
- Followed the specified package structure (Entity, dao, Main).
- Included user feedback messages for success or failure.

```
package dao;

import Entity.Event;
import Entity.Venue;

import java.time.LocalDate;
import java.time.LocalTime;
import java.util.List;

public interface IEventServiceProvider {
    public Event create_event(String event_name, LocalDate event_date, Lo
    public void getEventDetails(Event event);
    public int getAvailableNoOfTickets(Event event);
}

package dao;

import Entity.Booking;
import Entity.Customer;
import Entity.Event;

import java.util.List;

public interface IBookingServiceProvider {

    public void calculate_booking_cost(Event event, int num_tickets, Bookin
    public Booking book_tickets(Event event, int num_tickets, List<Custome
```

```
    public void cancel_tickets(int booking_id);
    public void get_booking_details(int booking_id);



}

package dao;

import Entity.*;

import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class EventServiceProviderImpl implements IEventServiceProvider
    Scanner sc = new Scanner(System.in);

    @Override
    public Event create_event(String event_name, LocalDate event_date, Lo

        Event event = null;

        if(event_type.equalsIgnoreCase("Movie")){
            System.out.println("Enter Movie Genre: ");
            String genre = sc.next();
            sc.nextLine();
            System.out.println("Enter Actor Name: ");
            String ActorName = sc.nextLine();
            System.out.println("Enter ActressName: ");
            String ActressName = sc.nextLine();

            event = new Movie(event_name, event_date, event_time, venue, tot
        }

        if(event_type.equalsIgnoreCase("Concert")){
```

```
            System.out.println("Enter Artist: ");
            String artist = sc.nextLine();
            System.out.println("Enter ConcertType");
            String concertType = sc.nextLine();

            event = new Concert(event_name, event_date, event_time, venue, t
        }

        if(event_type.equalsIgnoreCase("Sports"))
        {
            System.out.println("Enter SportName: ");
            String sportName = sc.nextLine();
            System.out.println("Enter TeamsName: ");
            String teamsName = sc.nextLine();

            event = new Sport(event_name, event_date, event_time, venue, tota

        }

        return event;
    }

    @Override
    public void getEventDetails(Event event){

      event.display_event_details();



    }

    @Override
    public int getAvailableNoOfTickets(Event event){
        return event.getAvailable_seats();
    }

}

package dao;
```

```
import Entity.Booking;
import Entity.Customer;
import Entity.Event;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class BookingSystemServiceProviderImpl implements IBookingServ
    List<Event> events = new ArrayList<>();
    List<Booking> bookings = new ArrayList<>();

   @Override
   public void calculate_booking_cost(Event event, int num_tickets, Bookin
      double totalCost = event.getTicket_price() * num_tickets;
      booking.setTotal_cost(totalCost);
      booking.setNum_tickets(num_tickets);
   }
   @Override
   public Booking book_tickets(Event event, int num_tickets, List<Custome

      double totalCost = 0.00;
      Booking booking = null;
      events.add(event);


      if(event == null){
         System.out.println("NO EVENT SELECTED");
         return booking;
      }
      if (customers == null || customers.size() != num_tickets) {
         System.out.println("Customer count does not match number of tick
         return null;
      }

      if(event.getAvailable_seats() < num_tickets)
      {
```

```
            System.out.println("Not enough tickets");
            return booking;
        }
        else{
            totalCost = num_tickets * event.getTicket_price();
            booking = new Booking(
                    event,
                    customers,
                    num_tickets,
                    totalCost,
                    LocalDate.now()
            );
            event.setAvailable_seats(event.getAvailable_seats() - num_tickets);
            bookings.add(booking);
            System.out.println("Booking Successful, your booking Id is: " + boo
        }
        return booking;
    }
    @Override
    public void cancel_tickets(int booking_id){

        Booking booking = null;

        for(Booking b: bookings){
            if(b.getBookingId()==booking_id){
                booking = b;
                break;
            }
        }

        if(booking != null){
            Event e = booking.getEvent();
            e.setAvailable_seats(e.getAvailable_seats() + booking.getNum_tick
            System.out.println("Canceled Successfully");
            bookings.remove(booking);
        }
        else{
            System.out.println("Booking NOT FOUND");
```

```
        }



    }
    @Override
    public void get_booking_details(int booking_id){

        Booking booking = null;

        for(Booking b: bookings){
            if(b.getBookingId() == booking_id){
                booking = b;
                break;
            }

        }

        if(booking != null){
            booking.display_booking_details();
        }
        else{
            System.out.println("BOOKING NOT FOUND");
        }

    }
}
```

```
package Main;
import Entity.*;
import dao.BookingSystemServiceProviderImpl;
import dao.EventServiceProviderImpl;
import dao.IBookingServiceProvider;
import dao.IEventServiceProvider;


import java.time.LocalDate;
import java.time.LocalTime;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TicketBookingSystem {

    List<Event> events = new ArrayList<>();
    List<Booking> bookings = new ArrayList<>();
    IEventServiceProvider e = new EventServiceProviderImpl();
    IBookingServiceProvider b = new BookingSystemServiceProviderImpl();
    Scanner sc = new Scanner(System.in);


    private static LocalDate ConvertDate(int year, int month, int day){
        return LocalDate.of(year, month, day);
    }

    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();
        Scanner sc = system.sc;

        while (true) {
            System.out.println("\n1. Create Event");
            System.out.println("2. View Event Details");
            System.out.println("3. Book Tickets");
            System.out.println("4. Cancel Tickets");
            System.out.println("5. Get Booking Details");
            System.out.println("0. Exit");
            System.out.print("Enter choice: ");
            int choice = Integer.parseInt(sc.nextLine());

            switch (choice) {
                case 1 -> {
                    System.out.print("Enter event name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter date (yyyy-mm-dd): ");
                    System.out.println("Enter Year: ");
                    int year = sc.nextInt();
```

```
            System.out.println("Enter Month: ");
            int month = sc.nextInt();
            System.out.println("Enter Day: ");
            int day = sc.nextInt();
            LocalDate date = ConvertDate(year, month, day);
            System.out.print("Enter time (HH:mm): ");
            System.out.println("Enter Hour: ");
            int Hours = sc.nextInt();
            System.out.println("Enter Minutes:");
            int Minutes = sc.nextInt();
            LocalTime time = LocalTime.of(Hours, Minutes);
            sc.nextLine();
            System.out.print("Enter total seats: ");
            int seats = sc.nextInt();
            System.out.print("Enter ticket price: ");
            double price = sc.nextDouble();
            sc.nextLine();
            System.out.print("Enter event type (Movie, Concert, Sports): "
            String type = sc.nextLine();
            System.out.println("Enter Venue Name: ");
            String venueName = sc.nextLine();
            sc.nextLine();
            System.out.println("Enter Venue Address: ");
            String addressV = sc.nextLine();
            Venue venue = new Venue(venueName, addressV);

            Event newEvent = system.e.create_event(name, date, time, se
            system.events.add(newEvent);

        }
        case 2 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEv
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {
```

```
                system.e.getEventDetails(system.events.get(idx - 1));

            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 3 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEv
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());

            if (idx >= 1 && idx <= system.events.size()) {
                System.out.print("Enter number of tickets to book: ");
                int tickets = Integer.parseInt(sc.nextLine());
                List<Customer> customers = new ArrayList<>();
                for(int i=0; i<tickets; i++)
                {
                    System.out.println("Enter Your name: ");
                    String name = sc.nextLine();
                    System.out.println("Enter email: ");
                    String email = sc.nextLine();
                    System.out.println("Enter Phone Number: ");
                    String phone = sc.next();
                    sc.nextLine();
                    Customer customer = new Customer(name, email, phone
                    customers.add(customer);
                }

                Booking booking = system.b.book_tickets(system.events.ge
                system.bookings.add(booking);

            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 4 -> {
```

```
                for (int i = 0; i < system.events.size(); i++) {
                    System.out.println((i + 1) + ". " + system.events.get(i).getEv
                }
                System.out.print("Enter Booking Id: ");
                int bookingId = Integer.parseInt(sc.nextLine());

                if (bookingId >= 1 && bookingId <= system.bookings.size()) {
                    system.b.cancel_tickets(bookingId);
                }
                else {
                    System.out.println("Booking NOT FOUND");
                }
            }

            case 5→{
                for (int i = 0; i < system.events.size(); i++) {
                    System.out.println((i + 1) + ". " + system.events.get(i).getEv
                }
                System.out.print("Enter BookingID: ");
                int bookingId = Integer.parseInt(sc.nextLine());

                if (bookingId >= 1 && bookingId <= system.bookings.size()) {
                    system.b.get_booking_details(bookingId);
                } else {
                    System.out.println("Invalid Booking Id");
                }
            }


            case 0 → {
                System.out.println("Exiting...");
                return;
            }
            default → System.out.println("Invalid choice.");
        }
    }
}
```

```
}
```

- **Task 9: Exception Handling:**
  - Created custom exceptions `EventNotFoundException` and `InvalidBookingException` in the `exception` package (`Task9/src/exception/`).
  - Modified service implementation methods (`Task9/src/dao/`) to throw these custom exceptions under appropriate conditions (e.g., event not found during booking, invalid booking ID for cancellation/details).
  - Updated the `TicketBookingSystem` main method (`Task9/src/Main/TicketBookingSystem.java`) to include `try-catch` blocks to handle these custom exceptions and potential `NullPointerException`, displaying informative error messages to the user.

```
package exception;

public class EventNotFoundException extends Exception {

   public EventNotFoundException(String message){
      super(message);
   }
}

package exception;

public class InvalidBookingException extends Exception {

   public InvalidBookingException(String message){
      super(message);
   }

}

package dao;

import Entity.*;
import exception.EventNotFoundException;
```

```
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class EventServiceProviderImpl implements IEventServiceProvider {
    Scanner sc = new Scanner(System.in);

    @Override
    public Event create_event(String event_name, LocalDate event_date, Lo

        Event event = null;

        if(event_type.equalsIgnoreCase("Movie")){
            System.out.println("Enter Movie Genre: ");
            String genre = sc.next();
            sc.nextLine();
            System.out.println("Enter Actor Name: ");
            String ActorName = sc.nextLine();
            System.out.println("Enter ActressName: ");
            String ActressName = sc.nextLine();

            event = new Movie(event_name, event_date, event_time, venue, tot
        }

        if(event_type.equalsIgnoreCase("Concert")){
            System.out.println("Enter Artist: ");
            String artist = sc.nextLine();
            System.out.println("Enter ConcertType");
            String concertType = sc.nextLine();

            event = new Concert(event_name, event_date, event_time, venue, t
        }

        if(event_type.equalsIgnoreCase("Sports"))
        {
```

```
            System.out.println("Enter SportName: ");
            String sportName = sc.nextLine();
            System.out.println("Enter TeamsName: ");
            String teamsName = sc.nextLine();

            event = new Sport(event_name, event_date, event_time, venue, tota

        }

        return event;
    }

    @Override
    public void getEventDetails(Event event) throws EventNotFoundExceptio

        if(event == null){
            throw new EventNotFoundException("Event Not found");
        }

       event.display_event_details();


    }

    @Override
    public int getAvailableNoOfTickets(Event event) throws EventNotFoundE

        if(event == null){
            throw new EventNotFoundException("Event Not found");
        }



        return event.getAvailable_seats();
    }

}
```

```
package dao;

import Entity.Booking;
import Entity.Customer;
import Entity.Event;
import exception.EventNotFoundException;
import exception.InvalidBookingException;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class BookingSystemServiceProviderImpl implements IBookingServ
    List<Event> events = new ArrayList<>();
    List<Booking> bookings = new ArrayList<>();

   @Override
   public void calculate_booking_cost(Event event, int num_tickets, Bookin

      if(event == null){
         throw new EventNotFoundException("Event Not Found");
      }

      double totalCost = event.getTicket_price() * num_tickets;
      booking.setTotal_cost(totalCost);
      booking.setNum_tickets(num_tickets);
   }
   @Override
   public Booking book_tickets(Event event, int num_tickets, List<Custome

      if(event == null){
         throw new EventNotFoundException("Event Not Found");
      }



      double totalCost = 0.00;
      Booking booking = null;
```

```
        events.add(event);


        if(event == null){
            System.out.println("NO EVENT SELECTED");
            return booking;
        }
        if (customers == null || customers.size() != num_tickets) {
            System.out.println("Customer count does not match number of tick
            return null;
        }

        if(event.getAvailable_seats() < num_tickets)
        {
            System.out.println("Not enough tickets");
            return booking;
        }
        else{
            totalCost = num_tickets * event.getTicket_price();
            booking = new Booking(
                    event,
                    customers,
                    num_tickets,
                    totalCost,
                    LocalDate.now()
            );
            event.setAvailable_seats(event.getAvailable_seats() - num_tickets);
            bookings.add(booking);
            System.out.println("Booking Successful, your booking Id is: " + boo
        }
        return booking;
    }
    @Override
    public void cancel_tickets(int booking_id) throws InvalidBookingExceptio

        if(booking_id == 0 || booking_id < 0){
            throw new InvalidBookingException("Booking Id can't be 0 OR Null
        }
```

```
    Booking booking = null;

    for(Booking b: bookings){
        if(b.getBookingId()==booking_id){
            booking = b;
            break;
        }
    }

    if(booking != null){
        Event e = booking.getEvent();
        e.setAvailable_seats(e.getAvailable_seats() + booking.getNum_tick
        System.out.println("Canceled Successfully");
        bookings.remove(booking);
    }
    else{
        System.out.println("Booking NOT FOUND");
        throw new InvalidBookingException("Booking Id not FOUND");
    }



}
@Override
public void get_booking_details(int booking_id) throws InvalidBookingEx

    if(booking_id == 0 || booking_id < 0){
        throw new InvalidBookingException("Booking Id can't be 0 OR Null
    }


    Booking booking = null;

    for(Booking b: bookings){
        if(b.getBookingId() == booking_id){
```

```
            booking = b;
            break;
        }

    }

    if(booking != null){
        booking.display_booking_details();
    }
    else{
        System.out.println("BOOKING NOT FOUND");
        throw new InvalidBookingException("Booking Id not FOUND");
    }

  }
}
```

```
package Main;
import Entity.*;
import dao.BookingSystemServiceProviderImpl;
import dao.EventServiceProviderImpl;
import dao.IBookingServiceProvider;
import dao.IEventServiceProvider;
import exception.EventNotFoundException;
import exception.InvalidBookingException;


import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TicketBookingSystem {

   List<Event> events = new ArrayList<>();
   List<Booking> bookings = new ArrayList<>();
   IEventServiceProvider e = new EventServiceProviderImpl();
```

```
IBookingServiceProvider b = new BookingSystemServiceProviderImpl();
Scanner sc = new Scanner(System.in);


private static LocalDate ConvertDate(int year, int month, int day){
    return LocalDate.of(year, month, day);
}

public static void main(String[] args) {
    TicketBookingSystem system = new TicketBookingSystem();
    Scanner sc = system.sc;

    while (true) {
        System.out.println("\n1. Create Event");
        System.out.println("2. View Event Details");
        System.out.println("3. Book Tickets");
        System.out.println("4. Cancel Tickets");
        System.out.println("5. Get Booking Details");
        System.out.println("0. Exit");
        System.out.print("Enter choice: ");
        int choice = Integer.parseInt(sc.nextLine());

        switch (choice) {
            case 1 -> {
                System.out.print("Enter event name: ");
                String name = sc.nextLine();
                System.out.print("Enter date (yyyy-mm-dd): ");
                System.out.println("Enter Year: ");
                int year = sc.nextInt();
                System.out.println("Enter Month: ");
                int month = sc.nextInt();
                System.out.println("Enter Day: ");
                int day = sc.nextInt();
                LocalDate date = ConvertDate(year, month, day);
                System.out.print("Enter time (HH:mm): ");
                System.out.println("Enter Hour: ");
                int Hours = sc.nextInt();
                System.out.println("Enter Minutes:");
```

```
            int Minutes = sc.nextInt();
            LocalTime time = LocalTime.of(Hours, Minutes);
            sc.nextLine();
            System.out.print("Enter total seats: ");
            int seats = sc.nextInt();
            System.out.print("Enter ticket price: ");
            double price = sc.nextDouble();
            sc.nextLine();
            System.out.print("Enter event type (Movie, Concert, Sports): "
            String type = sc.nextLine();
            System.out.println("Enter Venue Name: ");
            String venueName = sc.nextLine();
            sc.nextLine();
            System.out.println("Enter Venue Address: ");
            String addressV = sc.nextLine();
            Venue venue = new Venue(venueName, addressV);

            Event newEvent = system.e.create_event(name, date, time, se

            if(newEvent == null){
                throw new NullPointerException("Null Event");
            }

            system.events.add(newEvent);

        }
        case 2 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEv
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {


                try{
                    system.e.getEventDetails(system.events.get(idx - 1));
                }
```

```
                catch (EventNotFoundException e){
                    System.out.println(e.getMessage());

                }




            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 3 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEv
            }
            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());

            if (idx >= 1 && idx <= system.events.size()) {
                System.out.print("Enter number of tickets to book: ");
                int tickets = Integer.parseInt(sc.nextLine());
                List<Customer> customers = new ArrayList<>();
                for(int i=0; i<tickets; i++)
                {
                    System.out.println("Enter Your name: ");
                    String name = sc.nextLine();
                    System.out.println("Enter email: ");
                    String email = sc.nextLine();
                    System.out.println("Enter Phone Number: ");
                    String phone = sc.next();
                    sc.nextLine();
                    Customer customer = new Customer(name, email, phone
                    customers.add(customer);
                }
                Booking booking = null;
                try{
                    booking = system.b.book_tickets(system.events.get(idx-1
```

```
                }
                catch(EventNotFoundException e){
                    System.out.println(e.getMessage());
                }

                if(booking == null){
                    throw new NullPointerException("Null Pointer");
                }

                system.bookings.add(booking);

            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 4 -> {
            for (int i = 0; i < system.events.size(); i++) {
                System.out.println((i + 1) + ". " + system.events.get(i).getEv
            }
            System.out.print("Enter Booking Id: ");
            int bookingId = Integer.parseInt(sc.nextLine());

            if (bookingId >= 1 && bookingId <= system.bookings.size()) {
              try{
                 system.b.cancel_tickets(bookingId);
              }
              catch(InvalidBookingException e){
                 System.out.println(e.getMessage());
               }


            }
            else {
                System.out.println("Booking NOT FOUND");
            }
        }

        case 5->{
```

```
                for (int i = 0; i < system.events.size(); i++) {
                    System.out.println((i + 1) + ". " + system.events.get(i).getEv
                }
                System.out.print("Enter BookingID: ");
                int bookingId = Integer.parseInt(sc.nextLine());

                if (bookingId >= 1 && bookingId <= system.bookings.size()) {
                    try{
                        system.b.get_booking_details(bookingId);
                    }
                    catch(InvalidBookingException e){
                        System.out.println(e.getMessage());
                    }


                } else {
                    System.out.println("Invalid Booking Id");
                }
            }


            case 0 -> {
                System.out.println("Exiting...");
                return;
            }
            default -> System.out.println("Invalid choice.");
        }
    }
  }

}
```

- **Task 10: Collections:**
  - Modified `BookingSystemServiceProviderImpl` (`Task10/src/dao/`) to use `Map<Integer, Event>` for events and `Map<Integer, Booking>` for bookings.
  - Updated the `Booking` class (`Task10/src/Entity/`) to use `Map<String, Customer>` (using email as key) to store customer details for a booking.

- Implemented `equals` and `hashCode` in `Customer` (based on email) and `Event` (based on eventId) to support usage in collections and avoid duplicates if Sets were used.
- Created an `EventComparator` (`Task10/src/util/EventComparator.java`) to sort events based on name and venue name. This was used in the main application (`Task10/src/Main/TicketBookingSystem.java`) to display sorted events.
- The main application logic was updated to work with Maps for retrieving and managing events, bookings, and customers.

```
package dao;

import Entity.Booking;
import Entity.Customer;
import Entity.Event;
import exception.EventNotFoundException;
import exception.InvalidBookingException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;


public class BookingSystemServiceProviderImpl implements IBookingServ

    private Map<Integer, Event> events = new HashMap<>();
    private Map<Integer, Booking> bookings = new HashMap<>();


    @Override
    public void calculate_booking_cost(Event event, int num_tickets, Bookin

        if(event == null){
            throw new EventNotFoundException("Event Not Found for cost cal
        }

        if (booking == null) {
            System.out.println("Booking object is null in cost calculation.");
```

```
        return;
    }


    double totalCost = event.getTicket_price() * num_tickets;
    booking.setTotal_cost(totalCost);
    booking.setNum_tickets(num_tickets);
}


@Override
public Booking book_tickets(Event event, int num_tickets, Map<String, C

    if(event == null){
        throw new EventNotFoundException("Event Not Found for booking
    }

    events.putIfAbsent(event.getEventId(), event);


    Booking booking = null;


    if (customers == null || customers.size() != num_tickets) {
        System.out.println("Customer count (" + (customers == null ? 0 : cu
        return null;
    }

    if(event.getAvailable_seats() < num_tickets)
    {
        System.out.println("Not enough tickets available for event: " + ever
        return booking;
    }
    else{
        double totalCost = num_tickets * event.getTicket_price();
        booking = new Booking(
                event,
                customers,
```

```
                num_tickets,
                totalCost,
                LocalDate.now()
        );
        event.setAvailable_seats(event.getAvailable_seats() - num_tickets);
        bookings.put(booking.getBookingId(), booking); // Use put for Map
        System.out.println("Booking Successful, your booking Id is: " + boo
    }
    return booking;
}


@Override
public void cancel_tickets(int booking_id) throws InvalidBookingExceptio

    if(!bookings.containsKey(booking_id)){
        System.out.println("Booking ID " + booking_id + " NOT FOUND for
        throw new InvalidBookingException("Booking Id " + booking_id + "
    }

    Booking booking = bookings.get(booking_id);


    if(booking != null){
        Event e = booking.getEvent();
        if (e != null) {
            e.setAvailable_seats(e.getAvailable_seats() + booking.getNum_ti
            System.out.println("Canceled Successfully for Booking ID: " + bo
            bookings.remove(booking_id);
        } else {
            System.out.println("Error: Event associated with booking " + boo
            bookings.remove(booking_id);
        }

    }
    else{

        System.out.println("Booking NOT FOUND (Error in logic if reached)
```

```
            throw new InvalidBookingException("Booking Id not FOUND (intern
        }
    }


    @Override
    public void get_booking_details(int booking_id) throws InvalidBookingEx

        if(!bookings.containsKey(booking_id)){
            System.out.println("Booking ID " + booking_id + " NOT FOUND for
            throw new InvalidBookingException("Booking Id " + booking_id + "
        }

        Booking booking = bookings.get(booking_id);

        if(booking != null){
            booking.display_booking_details();
        }
        else{
            System.out.println("BOOKING NOT FOUND (Error in logic if reache
            throw new InvalidBookingException("Booking Id not FOUND (intern
        }

    }



    public Event findEventByName(String name) {
        for (Event event : events.values()) {
            if (event.getEvent_name().equalsIgnoreCase(name)) {
                return event;
            }
        }
        return null;
    }


    public Map<Integer, Event> getAllEvents() {
```

```
        return events;
    }

    public Map<Integer, Booking> getAllBookings() {
        return bookings;
    }
}

package Entity;

import java.time.LocalDate;
import java.util.Map;
import java.util.HashMap;


public class Booking {
    private static int nextBookingId = 1;
    private int bookingId;
    private Event event;
    private Map<String, Customer> customers;
    private int num_tickets;
    private double total_cost;
    private LocalDate booking_date;

    public Booking(){
        this.bookingId = nextBookingId++;
        this.customers = new HashMap<>();
    }
    public Booking(Event event, Map<String, Customer> customers, int num
        this.bookingId = nextBookingId++;
        this.event = event;
        this.customers = customers;
        this.num_tickets = num_tickets;
        this.total_cost = total_cost;
        this.booking_date = booking_date;
    }
```

```
public void setBookingId(int bookingId) {
    this.bookingId = bookingId;
}


public void setCustomers(Map<String, Customer> customers) {
    this.customers = customers;
}


public static void setNextBookingId(int nextBookingId) {
    Booking.nextBookingId = nextBookingId;
}

public int getBookingId() {
    return bookingId;
}


public void setBooking_date(
        LocalDate booking_date) {
    this.booking_date = booking_date;
}


public void setEvent(Event event) {
    this.event = event;
}


public void setNum_tickets(int num_tickets) {
    this.num_tickets = num_tickets;
}


public void setTotal_cost(double total_cost) {
    this.total_cost = total_cost;
}
```

```
public Event getEvent() {
    return event;
}

public int getNum_tickets() {
    return num_tickets;
}

public double getTotal_cost() {
    return total_cost;
}

public LocalDate getBooking_date() {
    return booking_date;
}

public Map<String, Customer> getCustomers() {
    return customers;
}


public void display_booking_details(){

    if(customers == null || customers.isEmpty()){
        System.out.println("No customers for this booking.");
    }
    else{
        System.out.println("Customers: ");
        for(Customer c: customers.values()) {
            System.out.println(c);
        }
    }

    if(event == null){
        System.out.println("No event associated with this booking.");
    }
    else{
```

```
            System.out.println(event);
        }

        System.out.println("Num Tickets: " + num_tickets);
        System.out.println("Total Cost: " + total_cost);
        System.out.println("Booking Date: "+ booking_date);

    }

    @Override
    public String toString(){
        StringBuilder sb = new StringBuilder();
        sb.append("Booking ID: ").append(bookingId).append("\n");
        if(customers == null || customers.isEmpty()){
            sb.append("No Customers\n");
        } else {
            sb.append("Customers:\n");
            for(Customer c : customers.values()){
                sb.append(c.toString()).append("\n");
            }
        }
        if(event == null){
            sb.append("No Event\n");
        } else {
            sb.append(event.toString()).append("\n");
        }
        sb.append("Num Tickets: ").append(num_tickets).append("\n");
        sb.append("Total Cost: ").append(total_cost).append("\n");
        sb.append("Booking Date: ").append(booking_date);
        return sb.toString();
    }
}
```

```
package Entity;
import java.util.Objects;

public class Customer {
    private String customer_name;
```

```
    private String email;
    private String phone_number;

    public Customer() { }

    public Customer(String customer_name, String email, String phone_number
        this.customer_name = customer_name;
        this.email = email;
        this.phone_number = phone_number;
    }

    public String getCustomer_name() {
        return customer_name;
    }

    public void setCustomer_name(String customer_name) {
        this.customer_name = customer_name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone_number() {
        return phone_number;
    }

    public void setPhone_number(String phone_number) {
        this.phone_number = phone_number;
    }

    public void display_customer_details() {
        System.out.println("Customer Name: " + customer_name);
        System.out.println("Email: " + email);
```

```
        System.out.println("Phone Number: " + phone_number);
    }

    @Override
    public String toString() {
        return "Customer{" +
                "customer_name='" + customer_name + '\'' +
                ", email='" + email + '\'' +
                ", phone_number='" + phone_number + '\'' +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Customer customer = (Customer) o;
        return Objects.equals(email, customer.email);
    }

    @Override
    public int hashCode() {
        return Objects.hash(email);
    }
}

package util; // Assuming a util package

import Entity.Event;
import java.util.Comparator;

public class EventComparator implements Comparator<Event> {
    @Override
    public int compare(Event e1, Event e2) {
        int nameCompare = e1.getEvent_name().compareToIgnoreCase(e2.getEv
        if (nameCompare == 0) {
            return e1.getVenue().getVenue_name().compareToIgnoreCase(e2.getV
        }
```

```
        return nameCompare;
    }
}

package Main;
import Entity.*;
import dao.BookingSystemServiceProviderImpl;
import dao.EventServiceProviderImpl;
import dao.IBookingServiceProvider;
import dao.IEventServiceProvider;
import exception.EventNotFoundException;
import exception.InvalidBookingException;
import util.EventComparator;


import java.time.LocalDate;
import java.time.LocalTime;
import java.time.format.DateTimeParseException;
import java.util.*;
import java.util.stream.Collectors;


public class TicketBookingSystem {


    IEventServiceProvider eventService = new EventServiceProviderImpl();
    BookingSystemServiceProviderImpl bookingService = new BookingSystemS
    List<Event> events = new ArrayList<>();

    Scanner sc = new Scanner(System.in);


    private static LocalDate ConvertDate(String dateStr) {
        try {
            return LocalDate.parse(dateStr); // Assumes yyyy-mm-dd format
        } catch (DateTimeParseException e) {
            System.out.println("Invalid date format. Please use yyyy-mm-dd.");
            return null;
```

```
    }
}

private static LocalTime ConvertTime(String timeStr) {
    try {
        return LocalTime.parse(timeStr); // Assumes HH:mm format
    } catch (DateTimeParseException e) {
        System.out.println("Invalid time format. Please use HH:mm.");
        return null;
    }
}


public static void main(String[] args) {
    TicketBookingSystem system = new TicketBookingSystem();
    Scanner sc = system.sc;

    while (true) {
        System.out.println("\n--- Ticket Booking System Menu ---");
        System.out.println("1. Create Event");
        System.out.println("2. View All Events");
        System.out.println("3. Book Tickets");
        System.out.println("4. Cancel Tickets");
        System.out.println("5. Get Booking Details");
        System.out.println("0. Exit");
        System.out.print("Enter choice: ");
        int choice = -1;
        try {
            choice = Integer.parseInt(sc.nextLine());
        } catch (NumberFormatException e) {
            System.out.println("Invalid input. Please enter a number.");
            continue;
        }


        switch (choice) {
            case 1 -> {
                try {
```

```
            System.out.print("Enter event name: ");
            String name = sc.nextLine();
            System.out.print("Enter date (yyyy-mm-dd): ");
            LocalDate date = ConvertDate(sc.nextLine());
            if (date == null) continue;

            System.out.print("Enter time (HH:mm): ");
            LocalTime time = ConvertTime(sc.nextLine());
            if (time == null) continue;


            System.out.print("Enter total seats: ");
            int seats = Integer.parseInt(sc.nextLine());
            System.out.print("Enter ticket price: ");
            double price = Double.parseDouble(sc.nextLine());

            System.out.print("Enter event type (Movie, Concert, Sports): ");
            String type = sc.nextLine();
            System.out.print("Enter Venue Name: ");
            String venueName = sc.nextLine();
            System.out.print("Enter Venue Address: ");
            String addressV = sc.nextLine();
            Venue venue = new Venue(venueName, addressV);

            Event newEvent = system.eventService.create_event(name, dat
            system.events.add(newEvent);

            if(newEvent != null){

                System.out.println("Event '" + newEvent.getEvent_name() + "
            } else {
                System.out.println("Failed to create event. Check event type.'
            }
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format entered. Please try a
        } catch (Exception e) {
            System.out.println("An error occurred during event creation: " +
        }
```

```
        }
        case 2 -> {
            System.out.println("\n--- Available Events ---");

            if(system.events.isEmpty())
            {
                System.out.println("No Events Scheduled");
            }
            else{
                Collections.sort(system.events, new EventComparator());
                for (int i = 0; i < system.events.size(); i++) {
                    System.out.println((i + 1) + ". " + system.events.get(i).getEver
                }
            }


            System.out.print("Select event number: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= system.events.size()) {


                try{
                    system.eventService.getEventDetails(system.events.get(idx -
                }
                catch (EventNotFoundException e){
                    System.out.println(e.getMessage());

                }




            } else {
                System.out.println("Invalid selection.");
            }
        }
        case 3 -> {
```

```
            try {
                System.out.println("\n--- Select Event to Book ---");
                Map<Integer, Event> eventsMap = system.bookingService.getA
                if (eventsMap.isEmpty()) {
                    System.out.println("No events available to book.");
                    continue;
                }

                eventsMap.forEach((id, ev) -> System.out.println("ID: " + id + " -
                System.out.print("Enter Event ID to book: ");
                int eventIdToBook = Integer.parseInt(sc.nextLine());


                Event selectedEvent = eventsMap.get(eventIdToBook);

                if (selectedEvent != null) {
                    System.out.print("Enter number of tickets to book: ");
                    int tickets = Integer.parseInt(sc.nextLine());
                    Map<String, Customer> customers = new HashMap<>(); // C
                    for(int i=0; i<tickets; i++)
                    {
                        System.out.println("\nEnter details for ticket " + (i+1) + ":");
                        System.out.print("Enter Customer name: ");
                        String custName = sc.nextLine();
                        String email;
                        while (true) {
                            System.out.print("Enter email (unique identifier): ");
                            email = sc.nextLine();
                            if (customers.containsKey(email)) {
                                System.out.println("Email already used for this bookin
                            } else {
                                break;
                            }
                        }
                        System.out.print("Enter Phone Number: ");
                        String phone = sc.nextLine();

                        Customer customer = new Customer(custName, email, pho
```

```
                    customers.put(email, customer);
                }



                Booking booking = system.bookingService.book_tickets(sele



                if(booking == null){
                    System.out.println("Booking failed. Check availability or cu

                }



            } else {
                System.out.println("Invalid Event ID selection.");
            }
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format entered. Please ente
        } catch (EventNotFoundException e) {
            System.out.println("Booking Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("An unexpected error occurred during booki
            e.printStackTrace(); // For debugging
        }
    }
    case 4 -> {
        try {
            System.out.println("\n--- Cancel Booking ---");
            Map<Integer, Booking> currentBookings = system.bookingServ
            if(currentBookings.isEmpty()){
                System.out.println("No bookings available to cancel.");
                continue;
            }
            System.out.println("Current Booking IDs: " + currentBookings.k
            System.out.print("Enter Booking ID to cancel: ");
            int bookingId = Integer.parseInt(sc.nextLine());
```

```
                system.bookingService.cancel_tickets(bookingId);


            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a numeric Bookin
            } catch (InvalidBookingException e) {
                System.out.println("Cancellation Error: " + e.getMessage());
            } catch (Exception e) {
                System.out.println("An unexpected error occurred during cance
            }
        }


        case 5->{
            try {
                System.out.println("\n--- Get Booking Details ---");
                Map<Integer, Booking> currentBookings = system.bookingServ
                if(currentBookings.isEmpty()){
                    System.out.println("No bookings available.");
                    continue;
                }
                System.out.println("Current Booking IDs: " + currentBookings.k
                System.out.print("Enter Booking ID to view details: ");
                int bookingId = Integer.parseInt(sc.nextLine());


                system.bookingService.get_booking_details(bookingId);


            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a numeric Bookin
            } catch (InvalidBookingException e) {
                System.out.println("Error retrieving details: " + e.getMessage())
            } catch (Exception e) {
                System.out.println("An unexpected error occurred while fetchin
            }
        }
```

```
            case 0 -> {
                System.out.println("Exiting Ticket Booking System...");
                sc.close(); // Close scanner
                return;
            }
            default -> System.out.println("Invalid choice. Please try again.");
        }
    }
}


}
```

- **Task 11: Database Connectivity:**
  - Organized classes into packages (`bean`, `repository`, `service`, `util`, `exception`, `app`) as specified.
  - Created entity classes (in `bean` package) similar to Task 7/8.
  - Defined service interfaces (`service` package) and repository interface `IBookingSystemRepository` (`repository` package).
  - Implemented `BookingSystemRepositoryImpl` (`bean` package) which handled all database interactions (INSERT, SELECT, UPDATE, DELETE) for events, venues, customers, and bookings using JDBC. Implemented methods like `create_event`, `getEventDetails`, `getAvailableNoOfTickets`, `book_tickets`, `cancel_tickets`, `get_booking_details` interacting with the database. Used transactions (`setAutoCommit`, `commit`) for booking and cancellation logic.
  - Created `DBUtil` and `PropertyUtil` (`util` package) to manage database connection details (from `db.properties`) and provide connection objects.
  - Included `try-catch-finally` blocks in the repository implementation to handle `SQLException` and ensure resources (like connections) are closed. Custom exceptions (`EventNotFoundException`, `InvalidBookingException`) were also thrown from the repository layer.

- The main application (`app.TicketBookingSystem`) interacted with the `IBookingSystemRepository` implementation to perform operations, handling potential exceptions.

Entity/bean classes are same as Task 5, code for database connectivity:

```
package util;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class PropertyUtil {

    public static String getPropertyString(){

        Properties prop = new Properties();

        try{
            FileInputStream fs = new FileInputStream("db.properties");
            prop.load(fs);

            fs.close();

        }
        catch(IOException e)
        {
            e.printStackTrace();
        }

        String hostname = prop.getProperty("hostname");
        String dbname = prop.getProperty("dbname");
        String username = prop.getProperty("username");
        String password = prop.getProperty("password");
        String port = prop.getProperty("port");

        String connectionString = "jdbc:mysql://" + hostname + ":" + port + "
                "?user=" + username + "&password=" + password;
```

```
        return connectionString;

    }
}

package util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBUtil {

    public static Connection getDBConn() throws SQLException {
        return DriverManager.getConnection(PropertyUtil.getPropertyString()
    }
}
```

- Code for Method Implementations with database:

```
package bean;

import exception.EventNotFoundException;
import exception.InvalidBookingException;
import repository.IBookingSystemRepository;
import util.DBUtil;

import javax.swing.plaf.nimbus.State;
import javax.xml.transform.Result;
import java.sql.*;
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

public class BookingSystemRepositoryImpl implements IBookingSystemRe

    @Override
```

```
public Event create_event(String event_name, LocalDate event_date, Lo
    Event event = null;
    Connection con = null;

    try{
        con = DBUtil.getDBConn();
        String addVenue = "insert into Venue (venue_name, address) VALU
        PreparedStatement stmtt = con.prepareStatement(addVenue, State
        stmtt.setString(1, venue.getVenue_name());
        stmtt.setString(2, venue.getAddress());

        int venue_id = -1;

        int affectedRows = stmtt.executeUpdate();
        if(affectedRows > 0){
            ResultSet generatedKeys = stmtt.getGeneratedKeys();
            if(generatedKeys.next()){
                venue_id = generatedKeys.getInt(1);
            }
        }
        else{
            System.out.println("Something went wrong while adding the Ven
        }

        String sql = "insert into Event (event_name, event_date, event_time
        PreparedStatement stmt = con.prepareStatement(sql, Statement.RE
        stmt.setString(1, event_name);
        stmt.setDate(2, Date.valueOf(event_date));
        stmt.setTime(3, Time.valueOf(event_time));
        stmt.setInt(4, venue_id);
        stmt.setInt(5, total_seats);
        stmt.setInt(6, total_seats);
        stmt.setDouble(7, ticket_price);
        stmt.setString(8, event_type);
        int rowsAdded = stmt.executeUpdate();
        int eventId = -1;
        if(rowsAdded > 0){
            ResultSet generatedKeys = stmt.getGeneratedKeys();
```

```
            if(generatedKeys.next())
            {
              eventId =  generatedKeys.getInt(1);
            }
            event = new Event(
                  event_name,
                  event_date,
                  event_time,
                  venue,
                  total_seats,
                  ticket_price,
                  event_type
            );
            event.setEvent_id(eventId);
            System.out.println("Event Created Successfully");
         }
         else{
            System.out.println("Something WENT Wrong");
         }


      }
      catch(SQLException e)
      {
         e.printStackTrace();
      }

      finally {
         try{
            con.close();
         }
         catch(SQLException e){
            e.printStackTrace();
         }
      }
      return event;
   }
   @Override
```

```
public void getEventDetails(Event event) throws EventNotFoundExceptio

    if(event != null){
        event.display_event_details();
    }
    else{
        throw new EventNotFoundException("Event Not found");
    }

}
@Override
public int getAvailableNoOfTickets(Event event) throws EventNotFoundE
    int tickets = 0;
    if(event != null){
        tickets = event.getAvailable_seats();
    }
    else{
        throw new EventNotFoundException("Event Not found");
    }
    return tickets;

}
@Override
public void calculate_booking_cost(Event event, int num_tickets, Bookin

    if(event == null){
        throw new EventNotFoundException("Event Not found");
    }
    double totalCost = event.getTicket_price() * num_tickets;
    booking.setTotal_cost(totalCost);

}
@Override
public Booking book_tickets(Event event, int num_tickets, List<Custome
    Booking booking = null;
    Connection con = null;
    if(event == null){
        throw new EventNotFoundException("Event Not found");
```

```
        }

        if(num_tickets == 0){
            throw new EventNotFoundException("Enter valid no of Tickets");
        }

        if(customers.isEmpty()){
            throw new EventNotFoundException("Customer List can't be empty
        }

        if(event.getAvailable_seats() < num_tickets)
        {
            throw new EventNotFoundException("Only " + event.getAvailable_s
        }

        try{
            con = DBUtil.getDBConn();

            con.setAutoCommit(false);

            String addCustomers = "insert into customer(email, phone_number
            PreparedStatement stmt = con.prepareStatement(addCustomers, S
            int customerId = 0;
            boolean customerAdded = false;
            for(int i=0; i<customers.size(); i++){
                Customer c = customers.get(i);
                stmt.setString(1, c.getEmail());
                stmt.setString(2, c.getPhone_number());
                stmt.setString(3, c.getCustomer_name());
                int rowsAffected = stmt.executeUpdate();
                if(rowsAffected > 0){
                    ResultSet generatedKeys = stmt.getGeneratedKeys();
                    if(generatedKeys.next()){

                        if(i==0){
                            customerId = generatedKeys.getInt(1);
                        }
```

```
            }
            customerAdded = true;
        }
        else{
            System.out.println("SOMETHING WENT WRONG WHILE ADDI
            return booking;
        }
    }
    if(!customerAdded){
        System.out.println("SOMETHING WENT WRONG WHILE ADDING
        return booking;
    }

    String addBookings = "insert into booking (customer_id, event_id, r
            "VALUES(?, ?, ?, ?, ?)";
    PreparedStatement smt = con.prepareStatement(addBookings, Sta
    smt.setInt(1, customerId);
    smt.setInt(2, event.getEvent_id());
    smt.setInt(3, num_tickets);
    smt.setDouble(4, event.getTicket_price() * num_tickets);
    smt.setDate(5, Date.valueOf(LocalDate.now()));

    int rowsAffected = smt.executeUpdate();
    if(rowsAffected > 0){
        int bookingId = -1;

        ResultSet genereatedKeys = smt.getGeneratedKeys();
        if(genereatedKeys.next())
        {
            bookingId = genereatedKeys.getInt(1);
        }

        booking = new Booking(
                event,
                customers,
                num_tickets,
                event.getTicket_price() * num_tickets,
```

```
                LocalDate.now()
            );

            booking.setBookingId(bookingId);

        }
        else{
            System.out.println("Something Went Wrong while Booking");
            return booking;
        }

        //updating event

        String updateSeatsSql = "UPDATE Event SET available_seats = ava
        PreparedStatement emt = con.prepareStatement(updateSeatsSql);
        emt.setInt(1, num_tickets);
        emt.setInt(2, event.getEvent_id());
        emt.setInt(3, num_tickets);

        int rowsUpdated = emt.executeUpdate();
        if(rowsUpdated == 0){
            throw new EventNotFoundException("Something went wrong wh
        }



        con.commit();
        System.out.println("Booking Successfull!!!!");


    }
    catch (SQLException e){
        e.printStackTrace();
    }

    finally {
        if (con != null) {
            try {
                con.close(); // Close the connection
```

```
            } catch (SQLException finalEx) {
                System.err.println("Error closing connection: " + finalEx.getMe
            }
        }
    }

    return booking;
}
@Override
public void cancel_tickets(int booking_id) throws InvalidBookingExceptio
    Connection con = null;
    Booking booking = null;
    if(booking_id == 0 || booking_id < 0){
        throw new InvalidBookingException("Booking Id Can't be 0");
    }

    try{

        con = DBUtil.getDBConn();

        con.setAutoCommit(false);

        String sql = "Select from booking WHERE booking_id = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, booking_id);
       int eventId = 0;
       int num_tickets = 0;
        ResultSet rs = stmt.executeQuery();
        if(rs.next())
        {
            eventId = rs.getInt("event_id");
            num_tickets = rs.getInt("num_tickets");

        }
        else{
            System.out.println("Failed to fetch Events");
            return;
        }
```

```
        String sql2 = "Update Event SET available_seats = available_seats -
        PreparedStatement stmt2 = con.prepareStatement(sql2);
        stmt2.setInt(1, num_tickets);
        stmt2.setInt(2, eventId);

        int rowsUpdated = stmt2.executeUpdate();
        if(rowsUpdated > 0){
            System.out.println("Tickets Allocated");
        }
        else{
            System.out.println("Something went wrong while updating event
            return;
        }

        String sql3 = "Delete From Booking WHERE booking_id = ?";
        PreparedStatement stmt3 = con.prepareStatement(sql3);
        stmt3.setInt(1, booking_id);

        int rowsDeleted = stmt3.executeUpdate();
        if(rowsDeleted > 0){
            System.out.println("Booking canceled Successfully");
        }
        else{
            throw new InvalidBookingException("Booking Not FOund");
        }


        con.commit();


    }

    catch (SQLException e){
        e.printStackTrace();
    }

    finally {
```

```
        if (con != null) {
            try {
                con.close(); // Close the connection
            } catch (SQLException finalEx) {
                System.err.println("Error closing connection: " + finalEx.getMe
            }
        }

    }

    }
    @Override
    public void get_booking_details(int booking_id) throws InvalidBookingEx
    {
        Connection con = null;
        Booking booking = null;

        try{
            con = DBUtil.getDBConn();
            String sql = "SELECT * FROM Booking " +
                    "JOIN Event ON Booking.event_id = Event.event_id JOIN Venu
                    "JOIN Customer ON Booking.customer_id = Customer.custome
                    "WHERE Booking.booking_id = ?";
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setInt(1, booking_id);
            ResultSet rs = stmt.executeQuery();

            List<Customer> customers = new ArrayList<>();
            Event event = null;
            int num_tickets = 0;
            double total_cost = 0.0;
            LocalDate booking_date = null;

            while (rs.next()) {
                if (event == null) {
                    String eventName = rs.getString("event_name");
                    LocalDate eventDate = rs.getDate("event_date").toLocalDate()
                    LocalTime eventTime = rs.getTime("event_time").toLocalTime
```

```
            String venueName = rs.getString("venue_name");

            Venue venue = new Venue(venueName, "");
            int totalSeats = rs.getInt("total_seats");
            double ticketPrice = rs.getDouble("ticket_price");
            String eventType = rs.getString("event_type");
            event = new Event(eventName, eventDate, eventTime, venue,
            num_tickets = rs.getInt("num_tickets");
            total_cost = rs.getDouble("total_cost");
            booking_date = rs.getDate("booking_date").toLocalDate();
        }


        String customer_name = rs.getString("customer_name");
        String email = rs.getString("email");
        String phone_number = rs.getString("phone_number");
        Customer customer = new Customer(customer_name, email, pho
        customers.add(customer);
    }

    if (event == null) {
        throw new InvalidBookingException("No booking found for ID: " +
    }

    booking = new Booking(event, customers, num_tickets, total_cost,
    booking.display_booking_details();

} catch (SQLException e) {
    throw new RuntimeException(e);
}

finally {
    if (con != null) {
        try {
            con.close();
        } catch (SQLException finalEx) {
            System.err.println("Error closing connection: " + finalEx.getMe
        }
```

```
        }
      }
   }
}
```

```
package app;
import bean.BookingSystemRepositoryImpl;
import bean.Booking;
import bean.Customer;
import bean.Event;
import bean.Venue;
import exception.EventNotFoundException;
import exception.InvalidBookingException;
import repository.IBookingSystemRepository;
import service.IEventServiceProvider;
import util.DBUtil;
import java.sql.SQLException;
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TicketBookingSystem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        IBookingSystemRepository repo = new BookingSystemRepositoryImp
        List<Event> events = new ArrayList<>();
        while (true) {
            System.out.println("\n1. Create Event");
            System.out.println("2. View Event Details");
            System.out.println("3. Book Tickets");
            System.out.println("4. Cancel Tickets");
            System.out.println("5. Get Booking Details");
            System.out.println("0. Exit");
            int choice = Integer.parseInt(sc.nextLine());
            switch (choice) {
                case 1: {
```

```
            System.out.print("Enter event name: ");
            String name = sc.nextLine();
            System.out.print("Enter event date (yyyy-mm-dd): ");
            LocalDate eventDate = LocalDate.parse(sc.nextLine());
            System.out.print("Enter event time (HH:mm): ");
            LocalTime eventTime = LocalTime.parse(sc.nextLine());
            System.out.print("Enter total seats: ");
            int totalSeats = Integer.parseInt(sc.nextLine());
            System.out.print("Enter ticket price: ");
            double ticketPrice = Double.parseDouble(sc.nextLine());
            System.out.print("Enter event type: ");
            String eventType = sc.nextLine();
            System.out.print("Enter venue name: ");
            String venueName = sc.nextLine();
            System.out.print("Enter venue address: ");
            String venueAddress = sc.nextLine();
            Venue venue = new Venue(venueName, venueAddress);
            try {
                Event ev = repo.create_event(name, eventDate, eventTime,
                events.add(ev);
                System.out.println("Event created successfully with ID: " + e
            } catch (EventNotFoundException ex) {
                System.out.println(ex.getMessage());
            }
            break;
        }
        case 2: {

            System.out.println("Created Events: ");
            for(int i=0; i<events.size(); i++){
                Event e = events.get(i);
                System.out.println(i+1+"." + " " + e.getEvent_name());
            }

            System.out.print("Enter event index to view details: ");
            int idx = Integer.parseInt(sc.nextLine());
            if (idx >= 1 && idx <= events.size()) {
                try {
```

```
                repo.getEventDetails(events.get(idx - 1));
            } catch (EventNotFoundException ex) {
                System.out.println(ex.getMessage());
            }
        } else {
            System.out.println("Invalid index");
        }
        break;
    }
    case 3: {
        System.out.print("Enter event index to book tickets: ");
        int idx = Integer.parseInt(sc.nextLine());
        if (idx >= 1 && idx <= events.size()) {
            System.out.print("Enter number of tickets: ");
            int numTickets = Integer.parseInt(sc.nextLine());
            List<Customer> customers = new ArrayList<>();
            for (int i = 0; i < numTickets; i++) {
                System.out.print("Enter customer name: ");
                String custName = sc.nextLine();
                System.out.print("Enter customer email: ");
                String custEmail = sc.nextLine();
                System.out.print("Enter customer phone: ");
                String custPhone = sc.nextLine();
                customers.add(new Customer(custName, custEmail, cust
            }
            try {
                Booking booking = repo.book_tickets(events.get(idx - 1),
                if (booking != null)
                    System.out.println("Booking successful. Booking ID: " +
            } catch (EventNotFoundException ex) {
                System.out.println(ex.getMessage());
            }
        } else {
            System.out.println("Invalid index");
        }
        break;
    }
    case 4: {
```

```
                System.out.print("Enter booking ID to cancel tickets: ");
                int bookingId = Integer.parseInt(sc.nextLine());
                try {
                    repo.cancel_tickets(bookingId);
                } catch (InvalidBookingException ex) {
                    System.out.println(ex.getMessage());
                }
                break;
            }
            case 5: {
                System.out.print("Enter booking ID to view details: ");
                int bookingId = Integer.parseInt(sc.nextLine());
                try {
                    repo.get_booking_details(bookingId);
                } catch (InvalidBookingException ex) {
                    System.out.println(ex.getMessage());
                }
                break;
            }
            case 0: {
                System.out.println("Exiting...");
                return;
            }
            default: {
                System.out.println("Invalid choice");
            }
        }
    }
}
```

END