**Assignment-5 TicketBookingSystem**

# Table of Contents

Task 1: Conditional Statements

Task 2: Nested Conditional Statements

Task 3: Looping

Task 4: Class & Object

Task 5: Inheritance and polymorphism

Task 6: Abstraction

Task7: Has A Relation / Association

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

Task 9: Exception Handling

Task 10: Collection

Task 11: Database Connectivity.

Task 1: Conditional Statements

import java.util.Scanner;

public class Task1\_TicketAvailability {

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("Tickets unavailable!");

}

scanner.close();

}

}

Task 2: Nested Conditional Statements

import java.util.Scanner;

public class Task2\_TicketCostCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Available ticket types: Silver / Gold / Diamond");

System.out.print("Enter ticket type: ");

String ticketType = scanner.nextLine();

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

int numberOfTickets = scanner.nextInt();

double ticketPrice;

if (ticketType.equalsIgnoreCase("Silver")) {

ticketPrice = 100.0;

} else if (ticketType.equalsIgnoreCase("Gold")) {

ticketPrice = 200.0;

} else if (ticketType.equalsIgnoreCase("Diamond")) {

ticketPrice = 300.0;

} else {

System.out.println("Invalid ticket type.");

scanner.close();

return;

}

double totalCost = ticketPrice \* numberOfTickets;

System.out.println("Total cost: ₹" + totalCost);

scanner.close();}}

Task 3: Looping

import java.util.Scanner;

public class Task3\_LoopBooking {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

boolean continueBooking = true;

while (continueBooking) {

System.out.println("\n--- Ticket Booking ---");

System.out.print("Enter ticket type (Silver / Gold / Diamond): ");

String ticketType = scanner.nextLine();

double ticketPrice;

if (ticketType.equalsIgnoreCase("Silver")) {

ticketPrice = 100.0;

} else if (ticketType.equalsIgnoreCase("Gold")) {

ticketPrice = 200.0;

} else if (ticketType.equalsIgnoreCase("Diamond")) {

ticketPrice = 300.0;

} else {

System.out.println("Invalid ticket type.");

continue;

}

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

int numberOfTickets = scanner.nextInt();

double totalCost = ticketPrice \* numberOfTickets;

System.out.println("Total cost: ₹" + totalCost);

scanner.nextLine(); // Consume leftover newline

System.out.print("Type 'Exit' to stop or press Enter to continue: ");

String choice = scanner.nextLine();

if (choice.equalsIgnoreCase("Exit")) {

continueBooking = false;

System.out.println("Thank you for booking!");

}

}

scanner.close();

}

}

# Task 4: Entity Classes

The following are the core classes used in the system, each representing a real-world component:

## 1. Venue.java

public class Venue {

private int venueId;

private String venueName;

private String address;

public Venue() {}

public Venue(int venueId, String venueName, String address) {

this.venueId = venueId;

this.venueName = venueName;

this.address = address;

}

// Getters and setters...

}

## 2. Customer.java

public class Customer {

private int customerId;

private String customerName;

private String email;

private String phoneNumber;

public Customer() {}

public Customer(String name, String email, String phone) {

this.customerName = name;

this.email = email;

this.phoneNumber = phone;

}

// Getters and setters...

}

## 3. Event.java (Abstract)

public abstract class Event {

private int eventId;

private String eventName;

private LocalDate eventDate;

private LocalTime eventTime;

private Venue venue;

private int totalSeats;

private int availableSeats;

private double ticketPrice;

private String eventType;

public Event() {}

public Event(String eventName, LocalDate eventDate, LocalTime eventTime, Venue venue, int totalSeats, double ticketPrice, String eventType) {

this.eventName = eventName;

this.eventDate = eventDate;

this.eventTime = eventTime;

this.venue = venue;

this.totalSeats = totalSeats;

this.availableSeats = totalSeats;

this.ticketPrice = ticketPrice;

this.eventType = eventType;

}

public abstract void displayEventDetails();

// Common methods: bookTickets(), cancelBooking(), calculateTotalRevenue()...

}

## 4. Booking.java

public class Booking {

private int bookingId;

private Customer[] customers;

private Event event;

private int numTickets;

private double totalCost;

private LocalDateTime bookingDate;

public Booking() {}

public Booking(Customer[] customers, Event event, int numTickets) {

this.customers = customers;

this.event = event;

this.numTickets = numTickets;

this.totalCost = event.getTicketPrice() \* numTickets;

this.bookingDate = LocalDateTime.now();

event.bookTickets(numTickets);

}

// Getters, setters, and displayBookingDetails()...

}

# Task 5: Event Subclasses (Inheritance)

The Event class is declared as abstract and extended by three specialized subclasses:

1. Movie

2. Concert

3. Sports

Each subclass contains additional attributes relevant to the specific event type, and all override the displayEventDetails() method from the abstract parent class.

public class Movie extends Event {

private String genre;

private String actorName;

private String actressName;

public Movie() {}

public Movie(String eventName, LocalDate eventDate, LocalTime eventTime, Venue venue,

int totalSeats, double ticketPrice, String genre, String actorName, String actressName) {

super(eventName, eventDate, eventTime, venue, totalSeats, ticketPrice, "Movie");

this.genre = genre;

this.actorName = actorName;

this.actressName = actressName;

}

@Override

public void displayEventDetails() {

super.displayEventDetails();

System.out.println("Genre: " + genre);

System.out.println("Actor: " + actorName);

System.out.println("Actress: " + actressName);

}

// Getters and setters...

}

public class Concert extends Event {

private String artist;

private String type; // Theatrical, Classical, Rock, Recital

public Concert() {}

public Concert(String eventName, LocalDate eventDate, LocalTime eventTime, Venue venue,

int totalSeats, double ticketPrice, String artist, String type) {

super(eventName, eventDate, eventTime, venue, totalSeats, ticketPrice, "Concert");

this.artist = artist;

this.type = type;

}

@Override

public void displayEventDetails() {

super.displayEventDetails();

System.out.println("Artist: " + artist);

System.out.println("Concert Type: " + type);

}

// Getters and setters...

}

public class Sports extends Event {

private String sportName;

private String teamsName;

public Sports() {}

public Sports(String eventName, LocalDate eventDate, LocalTime eventTime, Venue venue,

int totalSeats, double ticketPrice, String sportName, String teamsName) {

super(eventName, eventDate, eventTime, venue, totalSeats, ticketPrice, "Sports");

this.sportName = sportName;

this.teamsName = teamsName;

}

@Override

public void displayEventDetails() {

super.displayEventDetails();

System.out.println("Sport: " + sportName);

System.out.println("Teams: " + teamsName);

}

// Getters and setters...

}

# Task 6: Abstraction & Interfaces

These interfaces provide abstract method definitions which are implemented by their respective service or repository implementation classes.

public interface IEventServiceProvider {

Event createEvent(String eventName, String date, String time, int totalSeats, double ticketPrice, String eventType, Venue venue);

List<Event> getAllEvents();

Event getEventByName(String eventName);

int getAvailableNoOfTickets(String eventName);

}

public interface IBookingSystemServiceProvider {

Booking bookTickets(String eventName, int numTickets, List<Customer> customers);

boolean cancelBooking(int bookingId);

Booking getBookingDetails(int bookingId);

}

public interface IBookingSystemRepository {

Event createEvent(String eventName, String date, String time, int totalSeats, double ticketPrice, String eventType, Venue venue);

List<Event> getEventDetails();

int getAvailableNoOfTickets(String eventName);

double calculateBookingCost(Event event, int numTickets);

Booking bookTickets(String eventName, int numTickets, List<Customer> customers);

boolean cancelBooking(int bookingId);

Booking getBookingDetails(int bookingId);

}

# Task 7: HAS A RELATION

All interactions with the MySQL database are abstracted into the BookingSystemRepositoryImpl class. This class implements the IBookingSystemRepository interface and handles all insert, select, update, and delete operations.

JDBC is used to establish database connections via a helper class DBConnUtil.

@Override

public Event createEvent(String eventName, String date, String time,

int totalSeats, double ticketPrice, String eventType, Venue venue) {

String sql = "INSERT INTO event (event\_name, event\_date, event\_time, venue\_id, total\_seats, available\_seats, ticket\_price, event\_type) "

+ "VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

try (Connection conn = DBConnUtil.getDbConnection();

PreparedStatement ps = conn.prepareStatement(sql, Statement.RETURN\_GENERATED\_KEYS)) {

ps.setString(1, eventName);

ps.setDate(2, Date.valueOf(LocalDate.parse(date)));

ps.setTime(3, Time.valueOf(LocalTime.parse(time)));

ps.setInt(4, venue.getVenueId());

ps.setInt(5, totalSeats);

ps.setInt(6, totalSeats);

ps.setDouble(7, ticketPrice);

ps.setString(8, eventType);

ps.executeUpdate();

ResultSet rs = ps.getGeneratedKeys();

if (rs.next()) {

int eventId = rs.getInt(1);

Event event = new Event() {}; // Or use subclass

event.setEventId(eventId);

event.setVenue(venue);

return event;

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

@Override

public Booking bookTickets(String eventName, int numTickets, List<Customer> customers) {

Event event = getEventByName(eventName);

if (event == null || event.getAvailableSeats() < numTickets) return null;

Customer customer = customers.get(0); // Simplified for 1-customer bookings

try (Connection conn = DBConnUtil.getDbConnection()) {

conn.setAutoCommit(false);

// Insert customer

String custSql = "INSERT INTO customer (customer\_name, email, phone\_number) VALUES (?, ?, ?)";

int customerId;

try (PreparedStatement ps = conn.prepareStatement(custSql, Statement.RETURN\_GENERATED\_KEYS)) {

ps.setString(1, customer.getCustomerName());

ps.setString(2, customer.getEmail());

ps.setString(3, customer.getPhoneNumber());

ps.executeUpdate();

ResultSet rs = ps.getGeneratedKeys();

rs.next();

customerId = rs.getInt(1);

}

// Insert booking

double totalCost = event.getTicketPrice() \* numTickets;

String bookSql = "INSERT INTO booking (customer\_id, event\_id, num\_tickets, total\_cost, booking\_date) VALUES (?, ?, ?, ?, NOW())";

try (PreparedStatement ps = conn.prepareStatement(bookSql)) {

ps.setInt(1, customerId);

ps.setInt(2, event.getEventId());

ps.setInt(3, numTickets);

ps.setDouble(4, totalCost);

ps.executeUpdate();

}

// Update event seats

String updateSql = "UPDATE event SET available\_seats = available\_seats - ? WHERE event\_id = ?";

try (PreparedStatement ps = conn.prepareStatement(updateSql)) {

ps.setInt(1, numTickets);

ps.setInt(2, event.getEventId());

ps.executeUpdate();

}

conn.commit();

return new Booking(); // Optionally populate

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

@Override

public boolean cancelBooking(int bookingId) {

try (Connection conn = DBConnUtil.getDbConnection()) {

int numTickets = 0, eventId = 0;

PreparedStatement ps1 = conn.prepareStatement("SELECT event\_id, num\_tickets FROM booking WHERE booking\_id = ?");

ps1.setInt(1, bookingId);

ResultSet rs = ps1.executeQuery();

if (rs.next()) {

eventId = rs.getInt("event\_id");

numTickets = rs.getInt("num\_tickets");

}

PreparedStatement ps2 = conn.prepareStatement("DELETE FROM booking WHERE booking\_id = ?");

ps2.setInt(1, bookingId);

ps2.executeUpdate();

PreparedStatement ps3 = conn.prepareStatement("UPDATE event SET available\_seats = available\_seats + ? WHERE event\_id = ?");

ps3.setInt(1, numTickets);

ps3.setInt(2, eventId);

ps3.executeUpdate();

return true;

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

APP.Java

package main;

import bean.\*;

import service.\*;

import service.impl.\*;

import java.time.LocalDate;

import java.time.LocalTime;

import java.util.List;

import java.util.Scanner;

public class App {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

IEventServiceProvider eventService = new EventServiceImpl();

IBookingSystemRepository bookingService = new BookingSystemRepositoryImpl();

while (true) {

System.out.println("\n==== Ticket Booking System Menu ====");

System.out.println("1. Create Event");

System.out.println("2. Book Tickets");

System.out.println("3. Cancel Booking");

System.out.println("4. Get Event Details");

System.out.println("5. Get Booking Details");

System.out.println("6. Exit");

System.out.print("Choose an option: ");

int choice = Integer.parseInt(sc.nextLine());

switch (choice) {

case 1 -> {

System.out.print("Enter event name: ");

String eventName = sc.nextLine();

System.out.print("Enter date (yyyy-mm-dd): ");

String date = sc.nextLine();

System.out.print("Enter time (HH:mm): ");

String time = sc.nextLine();

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 ID: ");

int venueId = sc.nextInt();

// System.out.println("Enter venue name: ");

// String venueName = sc.nextLine();

// System.out.println("Enter venue address: ");

// String venueAddress = sc.nextLine();

Venue venue = new Venue(venueId, "", "");

Event event = eventService.createEvent(eventName, date, time, seats, price, type, venue);

System.out.println("Event created with ID: " + event.getEventId());

}

case 2 -> {

System.out.print("Enter event name: ");

String eventName = sc.nextLine();

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

int numTickets = Integer.parseInt(sc.nextLine());

System.out.print("Enter customer name: ");

String name = sc.nextLine();

System.out.print("Enter customer email: ");

String email = sc.nextLine();

System.out.print("Enter phone number: ");

String phone = sc.nextLine();

Customer customer = new Customer(name, email, phone);

List<Customer> customerList = List.of(customer);

Booking booking = bookingService.bookTickets(eventName, numTickets, customerList);

if (booking != null) {

System.out.println("Booking successful with booking\_id="+booking.getBookingId());

} else {

System.out.println("Booking failed.");

}

}

case 3 -> {

System.out.print("Enter booking ID to cancel: ");

int id = Integer.parseInt(sc.nextLine());

boolean success = bookingService.cancelBooking(id);

if (success) {

System.out.println("Booking cancelled successfully.");

} else {

System.out.println("Cancellation failed.");

}

}

case 4 -> {

List<Event> events = eventService.getAllEvents();

for (Event e : events) {

System.out.println("----------------------------");

e.displayEventDetails();

}

}

case 5 -> {

System.out.print("Enter booking ID: ");

int id = Integer.parseInt(sc.nextLine());

Booking b = bookingService.getBookingDetails(id);

if (b != null) {

b.displayBookingDetails();

} else {

System.out.println("No booking found.");

}

}

case 6 -> {

System.out.println("Thank you! Exiting...");

sc.close();

return;

}

default -> System.out.println("Invalid choice.");

}

}

}

}

Task 8: Interface

package bean;

import java.time.LocalDate;

import java.time.LocalTime;

public abstract class Event {

private int eventId;

private String eventName;

private LocalDate eventDate;

private LocalTime eventTime;

private Venue venue;

private int totalSeats;

private int availableSeats;

private double ticketPrice;

private String eventType;

public Event() {}

public Event(String eventName, LocalDate eventDate, LocalTime eventTime,

Venue venue, int totalSeats, double ticketPrice, String eventType) {

this.eventName = eventName;

this.eventDate = eventDate;

this.eventTime = eventTime;

this.venue = venue;

this.totalSeats = totalSeats;

this.availableSeats = totalSeats;

this.ticketPrice = ticketPrice;

this.eventType = eventType;

}

// Getters and setters...

public int getEventId() {

return eventId;

}

public void setEventId(int eventId) {

this.eventId = eventId;

}

public String getEventName() {

return eventName;

}

public void setEventName(String eventName) {

this.eventName = eventName;

}

public LocalDate getEventDate() {

return eventDate;

}

public void setEventDate(LocalDate eventDate) {

this.eventDate = eventDate;

}

public LocalTime getEventTime() {

return eventTime;

}

public void setEventTime(LocalTime eventTime) {

this.eventTime = eventTime;

}

public Venue getVenue() {

return venue;

}

public void setVenue(Venue venue) {

this.venue = venue;

}

public int getTotalSeats() {

return totalSeats;

}

public void setTotalSeats(int totalSeats) {

this.totalSeats = totalSeats;

}

public int getAvailableSeats() {

return availableSeats;

}

public void setAvailableSeats(int availableSeats) {

this.availableSeats = availableSeats;

}

public double getTicketPrice() {

return ticketPrice;

}

public void setTicketPrice(double ticketPrice) {

this.ticketPrice = ticketPrice;

}

public String getEventType() {

return eventType;

}

public void setEventType(String eventType) {

this.eventType = eventType;

}

public void bookTickets(int numTickets) {

if (availableSeats >= numTickets) {

availableSeats -= numTickets;

}

}

public void cancelBooking(int numTickets) {

availableSeats += numTickets;

}

public int getBookedNoOfTickets() {

return totalSeats - availableSeats;

}

public double calculateTotalRevenue() {

return getBookedNoOfTickets() \* ticketPrice;

}

public void displayEventDetails() {

System.out.println("Event: " + eventName);

System.out.println("Available Seats: " + availableSeats);

System.out.println("Price: ₹" + ticketPrice);

}

}

package bean;

import java.time.LocalDate;

import java.time.LocalTime;

public class Movie extends Event {

private String genre;

private String actorName;

private String actressName;

public Movie() {

super();

}

public Movie(String eventName, LocalDate eventDate, LocalTime eventTime,

Venue venue, int totalSeats, double ticketPrice,

String genre, String actorName, String actressName) {

super(eventName, eventDate, eventTime, venue, totalSeats, ticketPrice, "Movie");

this.genre = genre;

this.actorName = actorName;

this.actressName = actressName;

}

@Override

public void displayEventDetails() {

super.displayEventDetails();

System.out.println("Genre: " + genre);

System.out.println("Lead Actor: " + actorName);

System.out.println("Lead Actress: " + actressName);

}

}

package bean;

import java.time.LocalDate;

import java.time.LocalTime;

public class Sports extends Event {

private String sportName;

private String teamsName;

public Sports() {

super();

}

public Sports(String eventName, LocalDate eventDate, LocalTime eventTime,

Venue venue, int totalSeats, double ticketPrice,

String sportName, String teamsName) {

super(eventName, eventDate, eventTime, venue, totalSeats, ticketPrice, "Sports");

this.sportName = sportName;

this.teamsName = teamsName;

}

@Override

public void displayEventDetails() {

super.displayEventDetails();

System.out.println("Sport: " + sportName);

System.out.println("Teams: " + teamsName);

}

}

package bean;

import java.time.LocalDate;

import java.time.LocalTime;

public class Concert extends Event {

private String artist;

private String type;

public Concert() {

super();

}

public Concert(String eventName, LocalDate eventDate, LocalTime eventTime,

Venue venue, int totalSeats, double ticketPrice,

String artist, String type) {

super(eventName, eventDate, eventTime, venue, totalSeats, ticketPrice, "Concert");

this.artist = artist;

this.type = type;

}

@Override

public void displayEventDetails() {

super.displayEventDetails();

System.out.println("Artist: " + artist);

System.out.println("Concert Type: " + type);

}

}

Here abstract class event is created and movie sport concert is extending them

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

Here most of the sections are done previously and I will show the remaining changes

package service;

import bean.Event;

import bean.Venue;

import java.util.List;

public interface IEventServiceProvider {

// Create an event with all required attributes and store it

Event createEvent(String eventName, String date, String time,

int totalSeats, double ticketPrice, String eventType, Venue venue);

// Retrieve all event details from the system

List<Event> getAllEvents();

// Retrieve a single event by name

Event getEventByName(String eventName);

// Get available number of tickets for a specific event

int getAvailableNoOfTickets(String eventName);

}

package service;

import bean.Booking;

import bean.Customer;

import java.util.List;

public interface IBookingSystemServiceProvider {

// Book tickets for an event using event name and list of customers

Booking bookTickets(String eventName, int numTickets, List<Customer> customers);

// Cancel a booking using its booking ID

boolean cancelBooking(int bookingId);

// Get full booking details by ID

Booking getBookingDetails(int bookingId);

}

package service;

import bean.Booking;

import bean.Customer;

import bean.Event;

import bean.Venue;

import java.util.List;

public interface IBookingSystemRepository {

// 1. Create a new event and store in DB

Event createEvent(String eventName, String date, String time,

int totalSeats, double ticketPrice, String eventType, Venue venue);

// 2. Retrieve all event details from the database

List<Event> getEventDetails();

// 3. Get available tickets for an event

int getAvailableNoOfTickets(String eventName);

// 4. Calculate and return booking cost

double calculateBookingCost(Event event, int numTickets);

// 5. Book tickets for an event and store booking/customer info in DB

Booking bookTickets(String eventName, int numTickets, List<Customer> listOfCustomers);

// 6. Cancel booking by booking ID and update available seats in DB

boolean cancelBooking(int bookingId);

// 7. Get booking details by ID

Booking getBookingDetails(int bookingId);

}

Task 9: Exception Handling

**package** exception;

**public** **class** EventNotFoundException **extends** RuntimeException {

**public** EventNotFoundException(String message) {

**super**(message);

// **TODO** Auto-generated constructor stub

}

}

**package** exception;

**public** **class** BookingNotFoundException **extends** RuntimeException{

**public** BookingNotFoundException(String message) {

**super**(message);

// **TODO** Auto-generated constructor stub

}

}

**package** exception;

**public** **class** NullPointException **extends** RuntimeException{

**public** NullPointException (String message) {

**super**(message);

// **TODO** Auto-generated constructor stub

}

}

Task 10: Collection

@Override

**public** List<Event> getEventDetails() {

List<Event> eventList = **new** ArrayList<>();

String sql = "SELECT \* FROM event";

**try** (Connection conn = DBConnUtil.*getDbConnection*();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

**while** (rs.next()) {

Event e = **new** Event() {}; // Replace with appropriate subclass if needed

e.setEventId(rs.getInt("event\_id"));

e.setEventName(rs.getString("event\_name"));

e.setEventDate(rs.getDate("event\_date").toLocalDate());

e.setEventTime(rs.getTime("event\_time").toLocalTime());

e.setTotalSeats(rs.getInt("total\_seats"));

e.setAvailableSeats(rs.getInt("available\_seats"));

e.setTicketPrice(rs.getDouble("ticket\_price"));

e.setEventType(rs.getString("event\_type"));

Venue v = **new** Venue(); // optional: load venue details

v.setVenueId(rs.getInt("venue\_id"));

e.setVenue(v);

eventList.add(e);

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** eventList;

}

Task 11: Database Connectivity.

Many classes and interfaces are same as task 8

**package** service.impl;

**import** bean.\*;

**import** exception.\*;

**import** service.\*;

**import** util.DBConnUtil;

**import** java.sql.\*;

**import** java.time.LocalDate;

**import** java.time.LocalTime;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** BookingSystemRepositoryImpl **implements** IBookingSystemRepository {

@Override

**public** Event createEvent(String eventName, String date, String time,

**int** totalSeats, **double** ticketPrice, String eventType, Venue venue) {

String sql = "INSERT INTO event (event\_name, event\_date, event\_time, venue\_id, total\_seats, available\_seats, ticket\_price, event\_type) " +

"VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

**try** (Connection conn = DBConnUtil.*getDbConnection*();

PreparedStatement ps = conn.prepareStatement(sql, Statement.***RETURN\_GENERATED\_KEYS***)

) {

ps.setString(1, eventName);

ps.setDate(2, Date.*valueOf*(LocalDate.*parse*(date)));

ps.setTime(3, Time.*valueOf*(LocalTime.*parse*(time)));

ps.setInt(4, venue.getVenueId());

ps.setInt(5, totalSeats);

ps.setInt(6, totalSeats);

ps.setDouble(7, ticketPrice);

ps.setString(8, eventType);

ps.executeUpdate();

ResultSet rs = ps.getGeneratedKeys();

**if** (rs.next()) {

**int** eventId = rs.getInt(1);

Event event = **new** Event() {}; // Replace with actual subclass as needed

event.setEventId(eventId);

event.setVenue(venue);

**return** event;

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** **null**;

}

@Override

**public** List<Event> getEventDetails() {

List<Event> eventList = **new** ArrayList<>();

String sql = "SELECT \* FROM event";

**try** (Connection conn = DBConnUtil.*getDbConnection*();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

**while** (rs.next()) {

Event e = **new** Event() {}; // Replace with appropriate subclass if needed

e.setEventId(rs.getInt("event\_id"));

e.setEventName(rs.getString("event\_name"));

e.setEventDate(rs.getDate("event\_date").toLocalDate());

e.setEventTime(rs.getTime("event\_time").toLocalTime());

e.setTotalSeats(rs.getInt("total\_seats"));

e.setAvailableSeats(rs.getInt("available\_seats"));

e.setTicketPrice(rs.getDouble("ticket\_price"));

e.setEventType(rs.getString("event\_type"));

Venue v = **new** Venue(); // optional: load venue details

v.setVenueId(rs.getInt("venue\_id"));

e.setVenue(v);

eventList.add(e);

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** eventList;

}

@Override

**public** **int** getAvailableNoOfTickets(String eventName) {

String sql = "SELECT available\_seats FROM event WHERE event\_name = ?";

**try** (Connection conn = DBConnUtil.*getDbConnection*();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setString(1, eventName);

ResultSet rs = ps.executeQuery();

**if** (rs.next()) {

**return** rs.getInt("available\_seats");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** 0;

}

@Override

**public** **double** calculateBookingCost(Event event, **int** numTickets) {

**return** event.getTicketPrice() \* numTickets;

}

@Override

**public** Booking bookTickets(String eventName, **int** numTickets, List<Customer> customers) {

**if** (customers == **null** || customers.size() != 1) {

System.***out***.println("Only one customer allowed per booking.");

**return** **null**;

}

Customer customer = customers.get(0);

Event event = getEventByName(eventName);

**if** (event.getAvailableSeats() < numTickets) **return** **null**;

**if**(event==**null**)

{

**throw** **new** EventNotFoundException("Ivalid Event id ");

}

**try** (Connection conn = DBConnUtil.*getDbConnection*()) {

conn.setAutoCommit(**false**);

**boolean** cust\_exist=**false**;

**int** customerId=0;

cust\_exist=customerExist(customer.getCustomerName());

**if**(!cust\_exist)

{

// 1. Insert customer

String customerSql = "INSERT INTO customer (customer\_name, email, phone\_number) VALUES (?, ?, ?)";

**try** (PreparedStatement ps = conn.prepareStatement(customerSql, Statement.***RETURN\_GENERATED\_KEYS***))

{

ps.setString(1, customer.getCustomerName());

ps.setString(2, customer.getEmail());

ps.setString(3, customer.getPhoneNumber());

ps.executeUpdate();

ResultSet rs = ps.getGeneratedKeys();

rs.next();

customerId = rs.getInt(1);

}

}

**else**

{

String sql = "SELECT customer\_id FROM customer WHERE customer\_name = ? limit 1";

**try**

{

PreparedStatement ps = conn.prepareStatement(sql);

ps.setString(1,customer.getCustomerName());

ResultSet rs = ps.executeQuery();

rs.next();

customerId = rs.getInt("customer\_id");

}

**catch**(SQLException e)

{

e.printStackTrace();

}

}

// 2. Insert booking

**double** totalCost = calculateBookingCost(event, numTickets);

**int** bookingId;

String bookingSql = "INSERT INTO booking (customer\_id, event\_id, num\_tickets, total\_cost, booking\_date) VALUES (?, ?, ?, ?, ?)";

**try** (PreparedStatement ps = conn.prepareStatement(bookingSql, Statement.***RETURN\_GENERATED\_KEYS***)) {

ps.setInt(1, customerId);

ps.setInt(2, event.getEventId());

ps.setInt(3, numTickets);

ps.setDouble(4, totalCost);

ps.setTimestamp(5, Timestamp.*valueOf*(java.time.LocalDateTime.*now*()));

ps.executeUpdate();

ResultSet rs = ps.getGeneratedKeys();

rs.next();

bookingId = rs.getInt(1);

}

// 3. Update available seats

String updateSeats = "UPDATE event SET available\_seats = available\_seats - ? WHERE event\_id = ?";

**try** (PreparedStatement ps = conn.prepareStatement(updateSeats)) {

//String event\_id="select event\_id from event where ";

ps.setInt(1, numTickets);

ps.setInt(2, event.getEventId());

ps.executeUpdate();

}

conn.commit();

Booking b=**new** Booking();

b.setBookingId(bookingId);

**return** b;

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** **null**;

}

@Override

**public** **boolean** cancelBooking(**int** bookingId) {

**try** (Connection conn = DBConnUtil.*getDbConnection*()) {

// Step 1: Find num\_tickets and event\_id before deleting

**int** numTickets = 0;

**int** eventId = 0;

String selectSql = "SELECT event\_id, num\_tickets FROM booking WHERE booking\_id = ?";

**try** (PreparedStatement ps = conn.prepareStatement(selectSql)) {

ps.setInt(1, bookingId);

ResultSet rs = ps.executeQuery();

**if** (rs.next()) {

eventId = rs.getInt("event\_id");

numTickets = rs.getInt("num\_tickets");

}

**else**

{

**throw** **new** BookingNotFoundException("Invalid Booking Id");

}

}

// Step 2: Delete booking

String deleteSql = "DELETE FROM booking WHERE booking\_id = ?";

**try** (PreparedStatement ps = conn.prepareStatement(deleteSql)) {

ps.setInt(1, bookingId);

ps.executeUpdate();

}

// Step 3: Update seats

String updateSeats = "UPDATE event SET available\_seats = available\_seats + ? WHERE event\_id = ?";

**try** (PreparedStatement ps = conn.prepareStatement(updateSeats)) {

ps.setInt(1, numTickets);

ps.setInt(2, eventId);

ps.executeUpdate();

}

**return** **true**;

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** **false**;

}

@Override

**public** Booking getBookingDetails(**int** bookingId) {

String sql = "SELECT b.\*, c.customer\_name, c.email, c.phone\_number, e.event\_name " +

"FROM booking b " +

"JOIN customer c ON b.customer\_id = c.customer\_id " +

"JOIN event e ON b.event\_id = e.event\_id " +

"WHERE b.booking\_id = ?";

**try** (Connection conn = DBConnUtil.*getDbConnection*();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, bookingId);

ResultSet rs = ps.executeQuery();

**if** (rs.next()) {

Booking booking = **new** Booking();

booking.setNumTickets(rs.getInt("num\_tickets"));

booking.setTotalCost(rs.getDouble("total\_cost"));

booking.setBookingDate(rs.getTimestamp("booking\_date").toLocalDateTime());

Customer customer = **new** Customer();

customer.setCustomerName(rs.getString("customer\_name"));

customer.setEmail(rs.getString("email"));

customer.setPhoneNumber(rs.getString("phone\_number"));

Event event = **new** Event() {};

event.setEventName(rs.getString("event\_name"));

booking.setCustomers(**new** Customer[] { customer });

booking.setEvent(event);

**return** booking;

}

**else**

{

**throw** **new** BookingNotFoundException("booking not found");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** **null**;

}

// Helper

**private** Event getEventByName(String name) {

String sql = "SELECT \* FROM event WHERE event\_name = ?";

**try** (Connection conn = DBConnUtil.*getDbConnection*();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setString(1, name);

ResultSet rs = ps.executeQuery();

**if** (rs.next()) {

Event e = **new** Event() {};

e.setEventId(rs.getInt("event\_id"));

e.setAvailableSeats(rs.getInt("available\_seats"));

e.setTicketPrice(rs.getDouble("ticket\_price"));

**return** e;

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** **null**;

}

//Helper

**private** **boolean** customerExist(String customerName) {

String sql = "SELECT \* FROM customer WHERE customer\_name = ?";

**try**

{

Connection conn = DBConnUtil.*getDbConnection*();

PreparedStatement ps = conn.prepareStatement(sql);

ps.setString(1,customerName);

ResultSet rs = ps.executeQuery();

**if**(rs.next()) **return** **true**;

}

**catch**(SQLException e)

{

e.printStackTrace();

}

**return** **false**;

}

}

Util classes which returns connection object from connection string read from a db.properties file

package util;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.io.Reader;

import java.util.Properties;

public class DBPropertyUtil {

// return "jdbc:mysql://localhost:3306/transportmanagementsystem?user=root&password=root";

public static String getConnectionString(String fileName)throws IOException {

//fileName="db.properties"

String connStr=null;

Properties props=new Properties();

FileInputStream fis=new FileInputStream(fileName);

//InputStream fis = DBPropertyUtil.class.getClassLoader().getResourceAsStream("db.properties");

props.load(fis);

String user=props.getProperty("user");

String password=props.getProperty("password");

String protocol=props.getProperty("protocol");

String system=props.getProperty("system");

String database=props.getProperty("database");

String port=props.getProperty("port");

connStr=protocol+"//"+system+":"+port+"/"+database+"?user="+user+"&password="+password;

return connStr;

}

}

package util;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.logging.Logger;

public class DBConnUtil {

private static final String fileName="db.properties";

public static Connection getDbConnection() {

Connection con=null;

String connString=null;

try {

connString=DBPropertyUtil.getConnectionString(fileName);

}catch (IOException e) {

System.out.println("Connection String Creation Failed");

e.printStackTrace();

}

if(connString!=null)

{

try

{

con=DriverManager.getConnection(connString);

}

catch (SQLException e)

{

System.out.println("Error While Establishing DBConnection........");

e.printStackTrace();

}

}

return con;

}

}