National University of Computer and Emerging Sciences

OOP (CS1004)

Date: May 7th 2024

Course Instructor(s)
Ms Zainab ,Ms Mahnoor, Sir Shafique

Lab Final Exam (B)

Total Time: 2 Hours
Total Marks: 100
Total Questions: 03

Semester: SP-2024
Campus: Karachi
Dept: Computer Science

Student Name Roll No Section Student Signature

Vetted by Vetter Signature

Question Q1 ***Weightage:15;* Marks: 30**

You're tasked with developing a tour management system for a travel agency offering various types of tours: adventure, cultural, and wildlife. Each event type has unique ticket pricing scenario.

a) Design an abstract base class named "Tour" with common tour attributes such as name, duration, and destination. Include abstract methods calculateTicketPrice() and scheduleTour() to accommodate different pricing and scheduling logic for each tour type.
b) Create derived classes for each tour type: "AdventureTour", "CulturalTour", and "WildlifeTour". Override the abstract methods in each derived class to implement specific pricing and scheduling logic. For adventure tours, pricing may be based on activities and tour duration. Cultural tours might have pricing based on the number of sites visited and additional experiences offered. Wildlife tours could have pricing based on the rarity of sightings and guide expertise.
c) Utilize dynamic dispatch to determine the appropriate version of the methods at runtime based on the actual object type.
d) Populate the array with instances of derived classes representing different tour types, such as AdventureTour CulturalTour, and WildlifeTour.
e) Demonstrate the functionality by creating instances of different tour types and calling the calculateTicketPrice() and scheduleTour() methods for each tour.

Question Q2 **Weightage: 15; Marks: 30**

Your task is to develop a Ticket Reservation System for a theater where users can view available seats, reserve seats, and cancel reservations.

Create Seat Class: The Seat class represents individual seats in the theater.. Member variables include row, seatNumber, and reserved. Methods include: reserve(): Marks the seat as reserved. cancelReservation(): Marks the seat as available again. getStatus(): Retrieves the status of the seat (available or reserved).

Now, in the `Theater` class, manage the theater's seating arrangement. Dynamically handle a collection of `Seat` objects representing all seats in the theater. Implement methods such as `initializeSeats(int numRows, int numSeatsPerRow)` to set up the seating arrangement, `viewAvailableSeats()` to display a seating chart showing available and reserved seats, `reserve(int row, int seat)` to allow a user to reserve a specific seat, and `cancelReservation(int row, int seat)` to enable a user to cancel a reservation for a specific seat. Additionally, the `Theater` class should include a private helper method `isValidSeat(int row, int seat)` to validate whether the specified seat exists.

Note: As you design the system, consider how the `Theater` class can interact with the `Seat` objects to manage seat reservations effectively. Think about encapsulation and data integrity: how can you ensure that only the `Theater` class has the necessary access to modify the reservation status of `Seat` objects? Consider the principles of object-oriented design and abstraction in your solution.

Ensure proper functionality and exception handling, such as attempting to reserve or cancel a reservation for a nonexistent seat, trying to reserve an already reserved seat, or attempting to cancel a reservation for an unreserved seat. Keep in mind that you need to maintain the integrity of the system while achieving the desired functionality. How can you achieve this through the design of your classes and their interactions?

Main Function:

In the main() function, an instance of the Theater class is created.

The seating arrangement is initialized with the desired number of rows and seats per row.

Users can view available seats, reserve seats, and cancel reservations through method calls.

The seating chart is displayed after each operation to reflect the changes.

Note: Apart from above mentioned functions, you can create more helper functions if required.

The system must provide a user-friendly interface for managing seat reservations in a theater, ensuring proper management of seats and handling of user requests with exception handling mechanisms.

Question Q3 ***Weightage: 20; Marks:40***

You're tasked with developing a property management system for a real estate company that deals with various types of properties, including residential apartments, commercial spaces, and vacation rentals.

a) Start by designing a base class called "Property" that encompasses common functionality shared by all properties, such as property address, size, type, and owner information. Additionally, create three derived classes, namely "ResidentialProperty," "CommercialProperty," and "VacationRental," which inherit from the "Property" class. Each derived class should include attributes specific to the type of property, such as the number of bedrooms and bathrooms for residential properties, rental rates for commercial spaces, and amenities for vacation rentals.

b) Ensure that each derived class includes unique attributes relevant to the type of property, such as lease terms for commercial spaces, property management services for vacation rentals, and neighborhood features for residential properties.

c) Develop a generic booking system capable of handling reservations for all types of properties using templates. This system should streamline the booking process and ensure consistency across all property bookings, regardless of the chosen type or location.

d) Enhance the booking system by implementing operator overloading for tax addition. Develop three operator overloading functions within the "Property" class to accommodate different tax rates for property bookings. The first operator overloading function should add a 40% tax rate to the total booking amount, suitable for properties in areas subject to higher taxes or fees. The second operator overloading function should apply a 10% tax rate, catering to properties in locations with moderate tax rates. Additionally, design a third operator overloading function to return both amounts after applying respective taxes, providing customers with transparent pricing and empowering them to make informed decisions.