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

Q1: You're tasked with developing an event handling system for the "GlobaArena" entertainment venue, which hosts various events such as rock concerts, business conferences, and art exhibitions. Each event type has unique ticket pricing scenarios.

a) Begin by creating an abstract base class named "Event" that encapsulates common event attributes such as name, date, and venue. Implement overloaded versions of the calculateTicketPrice() method within the Event class to accommodate the pricing logic specific to concerts, conferences, and exhibitions.
b) For concerts, tickets are priced based on seating tiers; for conferences, fees vary for attendees and speakers; and for exhibitions, admission is based on ticket categories.
c) Ensure that the event handling system can dynamically determine the appropriate version of the calculateTicketPrice() method at runtime based on the event type and provided parameters, utilizing late binding.
d) Populate the array with instances of derived classes representing different event types, such as rock concerts, business conferences, and art exhibitions.
e) Demonstrate the functionality by iterating through the array of events and calling the calculateTicketPrice() method for each event.

National University of Computer and Emerging Sciences

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

Consider that you are working an online Auction System where users can add items, place bids, and determine the winning bids for those items. The implementation ensures proper management of items and handles exceptions for invalid bids or item-related issues.

First Create the AuctionSystem Class and this class is friend of Item class (Explained below): AuctionSystem class designed to manage the auction system. It dynamically manages a collection of items available for auction. It contains private member variables to store an array of Item objects (items), the current number of items (itemCount), and the capacity of the array (capacity).

The AuctionSystem class provides methods to: addItem(int ID, double startingPrice): Adds a new item to the auction with the specified ID and starting price. placeBid(int itemID, int bidderID, double bidAmount): Places a bid on the item with the given ID by the specified bidder with the given bid amount. getWinningBidderID(int itemID): Retrieves the winning bidder ID for the item with the specified ID. getWinningBidAmount(int itemID): Retrieves the winning bid amount for the item with the specified ID. romve_Item(int Item_id): used to remove an item from the auction system once it's sold.

Create Item Class: The Item class represents an individual item available for auction. Each item has a unique ID, a starting price, and information about the highest bid placed on it.

Item class contains member variables to store the item's ID, starting price, ID of the highest bidder (highestBidderID), and the amount of the highest bid (highestBidAmount). The class provides methods to: placeBid(int bidderID, int bidAmount): this is pure virtual function Allows placing a bid on the item. If the bid is valid (higher than the starting price and the current highest bid), it updates the highest bidder ID and bid amount. getWinningBidderID(): Retrieves the ID of the winning bidder for the item. getWinningBidAmount(): Retrieves the amount of the winning bid for the item.

Note: Given the scenario where the AuctionSystem class has a unique access to private data within the Item class ensuring encapsulation, how might this exclusive relationship between the AuctionSystem and Item classes benefit bid management in the auction system?

Exception Handling: when an invalid bid amount is placed on an item (e.g., negative bid amount, bid lower than the starting price or current highest bid) throw the exception and handle it properly. Also, handle exception cases such as an item not found or no bids placed for an item, respectively.

In the main function, you'll create instances of the AuctionSystem class, add items, place bids, and display winning bidder IDs and bid amounts for each item as output.
This completes the implementation of the Auction System, ensuring proper management of items and handling of exceptions throughout the process.

Then display the Winning bidder IDs and bid amounts for each item as output.

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

National University of Computer and Emerging Sciences

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

Consider that you are working an online Auction System where users can add items, place bids, and determine the winning bids for those items. The implementation ensures proper management of items and handles exceptions for invalid bids or item-related issues.

First Create the AuctionSystem Class and this class is friend of Item class (Explained below): AuctionSystem class designed to manage the auction system. It dynamically manages a collection of items available for auction. It contains private member variables to store an array of Item objects (items), the current number of items (itemCount), and the capacity of the array (capacity).

The AuctionSystem class provides methods to: addItem(int ID, double startingPrice): Adds a new item to the auction with the specified ID and starting price. placeBid(int itemID, int bidderID, double bidAmount): Places a bid on the item with the given ID by the specified bidder with the given bid amount. getWinningBidderID(int itemID): Retrieves the winning bidder ID for the item with the specified ID. getWinningBidAmount(int itemID): Retrieves the winning bid amount for the item with the specified ID. romve_Item(int Item_id): used to remove an item from the auction system once it's sold.

Create Item Class: The Item class represents an individual item available for auction. Each item has a unique ID, a starting price, and information about the highest bid placed on it.

Item class contains member variables to store the item's ID, starting price, ID of the highest bidder (highestBidderID), and the amount of the highest bid (highestBidAmount). The class provides methods to: placeBid(int bidderID, int bidAmount): this is pure virtual function Allows placing a bid on the item. If the bid is valid (higher than the starting price and the current highest bid), it updates the highest bidder ID and bid amount. getWinningBidderID(): Retrieves the ID of the winning bidder for the item. getWinningBidAmount(): Retrieves the amount of the winning bid for the item.

Note: Given the scenario where the AuctionSystem class has a unique access to private data within the Item class ensuring encapsulation, how might this exclusive relationship between the AuctionSystem and Item classes benefit bid management in the auction system?

Exception Handling: when an invalid bid amount is placed on an item (e.g., negative bid amount, bid lower than the starting price or current highest bid) throw the exception and handle it properly. Also, handle exception cases such as an item not found or no bids placed for an item, respectively.

In the main function, you'll create instances of the AuctionSystem class, add items, place bids, and display winning bidder IDs and bid amounts for each item as output.
This completes the implementation of the Auction System, ensuring proper management of items and handling of exceptions throughout the process.

Then display the Winning bidder IDs and bid amounts for each item as output.

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

Question Q3

Weightage: 20; **Marks: 40**

You're tasked with developing a flight reservation system for a global airline company that offers various types of flights, including standard, premium, and luxury classes.

a) Begin by designing a base class called "Flight" that encompasses common functionality shared by all flights, such as flight number, departure and arrival airports, departure and arrival times, and airline information. Additionally, create three derived classes, namely "StandardFlight," "PremiumFlight," and "LuxuryFlight," which inherit from the "Flight" class. Each derived class should include attributes specific to the type of flight, such as seating configurations, in-flight amenities, and fare types.

b) Ensure that each derived class includes unique attributes relevant to the type of flight, such as the availability of gourmet meals, spacious seating arrangements, and exclusive lounge access.

c) Develop a generic booking system capable of handling reservations for all types of flights using templates. This system should streamline the booking process and ensure consistency across all flight bookings, regardless of the chosen class or destination.

d) Enhance the booking system by implementing operator overloading for tax addition. Develop three operator overloading functions within the "Flight" class to accommodate different tax rates for flight bookings. The first operator overloading function should add a 50% tax rate to the total booking amount, suitable for routes or destinations subject to higher taxes or fees. The second operator overloading function should apply a 20% tax rate, catering to routes or destinations with moderate tax rates. Additionally, design a third operator overloading function to return both amounts after applying respective taxes, providing passengers with transparent pricing and empowering them to make informe d decisions.