National University of Computer and Emerging Sciences

OOP (CS1004)

Date: May 7th 2024

Course Instructor(s)
Ms Shaharbano, Ms Fatima

Final Exam

Total Time: 2 Hours
Total Marks: 50
Total Questions: 03
PAPER : B

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

Student Name	Roll No	Section	Student Signature
Fozan Javaid ~~23K-0605~~	23K-0605	H	

Question no 01 ***[CLO# 1, points: 15, estimated time: 35 mins]***

Design a comprehensive ship hierarchy system in C++ that categorizes ships into 3 types: Naval Vessels, Cargo Ships and Cruise Ships, with specific subcategories reflecting their unique functionalities and attributes.

Define a base class **Ship** with common attributes (**name**, **yearBuilt**, **length**, **capacity**) and a virtual **display()** method to print basic ship information. Implement derived classes for each ship category: **NavalVessel**, **CargoShip** and **CruiseShip**, all publicly inheriting from **Ship** to extend it with specialized attributes and methods.

For **NavalVessel**, introduce attributes like **shipClass**, **missiles** and **armament**, along with methods for naval operations (**deployTorpedoes()**, **fireMissiles()**). Each time a torpedo is deployed, the number of missiles is set by a given number. And each time a missile is fired, the count of missiles is decremented. **CargoShip** should include attributes such as **cargoType**, **container** and cargo-handling methods (**loadCargo(string container)**, **unloadCargo()**). Each time a cargo is loaded, the given container is copied. And each time a cargo is unloaded, the container value comes "vacant". **CruiseShip** will have attributes like **passengerCapacity**, **destination**, and passenger services methods (**bookCabin()**, **updatecapacity()**). Each time a cabin is booked, it is checked whether the given number of passengers are exceeding the capacity, if they are not then the cabin is booked. Each time a capacity is updated, it is checked that it must be greater than the previous capacity.

Implement proper constructors for each class and use appropriate access specifiers (**public**, **protected**, **private**) for encapsulation. Override the **display()** method in each derived class to print detailed ship information specific to its category.

Demonstrate polymorphism by creating an array of **Ship*** pointers containing instances of different ship types. Iterate through this array and invoke the **display()** method for each ship, observing how virtual function dispatching works based on the actual object type.

Question no 02 ***[CLO# 2, points: 15, estimated time: 35 mins]***

You are tasked with designing a loan management system that caters to various types of loans including car loans, home loans, and business loans. Implement the system using object-oriented principles, utilizing abstract classes, friend classes, and friend functions to ensure encapsulation and modularity. Additionally, incorporate exception handling to ensure robustness and error tolerance.

1. Begin by creating an abstract class Loan with common attributes such as loanAmount, interestRate, and duration (in months). Include pure virtual functions for calculating monthly payments **virtual double calculateMonthlyPayment() const = 0** and displaying loan details **virtual void display() const = 0.**
2. Implement derived classes for different types of loans: CarLoan, HomeLoan, and BusinessLoan. Each derived class should inherit from the Loan class and provide concrete implementations for the pure virtual functions.
3. Additionally, implement a LoanManager class as a friend class to the Loan class. This class should have access to private and protected members of the Loan class to perform operations like approving loans and calculating total interest payments.
4. Implement a `LoanUtils` namespace with friend functions to perform common loan calculations, such as determining the total interest paid over the loan duration **static double calculateTotalInterestPaid(const Loan& loan).**
5. Demonstrate the functionality of the loan management system by creating instances of car loans, home loans, and business loans, setting their attributes, calculating monthly payments, and displaying loan details. Handle exceptions gracefully, ensuring that loan attributes are within valid ranges and that the loan duration is a positive integer.

Here are five common errors that you can handle with exception handling in a loan management system:

i. Invalid Loan Amount: Throw an exception if the loan amount is negative or zero.
ii. Invalid Interest Rate: Throw an exception if the interest rate is negative or exceeds a certain maximum value.
iii. Invalid Loan Duration: Throw an exception if the loan duration is negative or zero.
iv. Insufficient Funds: Throw an exception if the borrower does not have sufficient funds to make a payment.
v. Invalid Input Format: Throw an exception if the user enters invalid input format for loan attributes, such as entering non-numeric characters for loan amount or duration.

Instructions:

- Design the loan management system using abstract classes, friend classes, and friend functions as described above.
- Implement appropriate constructors, setter and getter for each class.
- Ensure that derived classes override pure virtual functions from the base class.
- Incorporate exception handling mechanisms to handle errors related to invalid loan attributes or negative loan durations.
- Demonstrate the functionality of the loan management system by creating instances of different loan types and performing relevant operations.

Question no 03 ***[CLO# 3, points: 20, estimated time: 35 mins]***

You are designing a system for a plant nursery that caters to landscaping projects. The nursery has been contracted to provide plants for two projects: **a company lawn** and **a family park**. Implement the system using object-oriented principles, generics, and operator overloading to ensure modularity and flexibility. Define a generic **class Plant** with attributes such as **name**, **type**, and **price**. Include appropriate constructors, accessors, and mutators(getters/setters). Implement derived classes **CompanyLawnPlant** and **FamilyParkPlant**, which inherit from the **Plant class**. These classes should include additional attributes specific to each project, such as **quantityNeeded** and **maintenanceLevel**.

i. Provide concrete implementations for constructors and appropriate methods to calculate the total cost of plants based on quantity and maintenance level.
ii. Create a generic class Project to represent a landscaping project. Include attributes such as projectName and totalCost. Implement methods to add plants to the project, calculate the total cost, and display project details.
iii. Overload the addition (+) operator to allow adding plants directly to a project.
iv. Demonstrate the functionality of the system by creating instances of plants for both the company lawn and the family park, adding them to their respective projects, calculating the total cost, and displaying project details.
v. Implement operator overloading for the output stream (<<) operator to allow printing plant and project details directly using cout.
Include error handling mechanisms to handle cases such as adding plants with negative quantities or invalid plant names.

Example Output:

Plants for Company Lawn: 1. Name: Marigold, Type: Flower, Quantity: 100, Price: $2.50 2. Name: Oak Tree, Type: Tree, Quantity: 10, Price: $30.00 Total Cost for Company Lawn: $325.00	Plants for Family Park: 1. Name: Rose Bush, Type: Shrub, Quantity: 50, Price: $5.00 2. Name: Maple Tree, Type: Tree, Quantity: 20, Price: $40.00 Total Cost for Family Park: $250.00

Instructions:

- Design the system using object-oriented principles, generics, and operator overloading as described above.
- Ensure that the system is modular and flexible, allowing for easy addition of new plant types and projects in the future.
- Provide appropriate constructors, accessors, and mutators(getter/setter) for each class.