****

**Programming Fundamentals - OOP Assignment (v3)**

**Task 1 (Code Already Provided): Design a Zoo Management System Using OOP**

**Objective:** The goal of this assignment is to design a Zoo Management System using Object-Oriented Programming. You will be applying the concepts of classes, objects, interfaces, and different relationships between them like Inheritance, Association, Aggregation, Composition, Dependency, and Realization/Implementation.

**Task Details:**

1. Animal Class: Create an abstract class 'Animal' with the following attributes: name, age, and species. This class should have methods 'Eat()' that simply prints a message saying that the animal is eating and another 'Eat(string food)' that prints what the animal is eating. Make sure to encapsulate these attributes properly.
2. ISoundBehaviour Interface: Create an interface 'ISoundBehaviour' with a method 'MakeSound()'. This method should not have any implementation.
3. Specific Animal Classes: Create classes for at least three types of animals (for example, Lion, Elephant, and Monkey). Each of these classes should inherit from the 'Animal' class and implement 'ISoundBehaviour' interface.

· Each specific animal class should override the 'Eat()' method to provide a unique message.

· Each specific animal class should provide the implementation for 'MakeSound()' method, which prints a unique sound that the animal makes.

· The specific animal class's constructor should accept a 'Habitat' object. Inside the constructor, add the created animal to the given 'Habitat' object.

1. Habitat Class: Create a class 'Habitat' that has a list of Animals and a name. This class should have a method to 'AddAnimal()'.
2. Zoo Class: Create a class 'Zoo' that has a list of Habitats. This class should have methods to 'AddHabitat()' and 'FeedAllAnimals()'. The 'FeedAllAnimals()' method should call the 'Eat()' method on each animal in the zoo.
3. Main Program: In your main program, create an instance of the 'Zoo' class. Create instances of your 'Habitat' class and add them to the zoo. Then, create instances of your specific animal classes and pass the 'Habitat' instance to them during creation. After that, call the 'FeedAllAnimals()' method to show that each type of animal eats in its own way and then call 'MakeSound()' method on each animal.

Your goal is to understand the problem statement and implement the solution while properly applying the concepts of encapsulation, abstraction, polymorphism, and inheritance, along with understanding the use of interfaces and different relationships like Association, Aggregation, Composition, Dependency, and Realization/Implementation.

The code for the above assignment is already given. Your goal is to understand the problem statement, analyze and debug the given code.

**Task 2: Create a Bank Account Management System**

**Objective:** The goal of this assignment is to design a Bank Account Management System using the core concepts of Object-Oriented Programming (OOP) - Abstraction, Encapsulation, Polymorphism, and Inheritance. Furthermore, the design should effectively utilize interfaces and display an understanding of relationships such as Association, Aggregation, Composition, Dependency, and Realization/Implementation.

**Task Details:**

1. Abstraction: Abstraction is the concept of hiding the internal details and describing things in simple terms. For example, a bank account can be considered as an abstraction. In this assignment, the BankAccount class is an abstraction. It abstracts the concept of a bank account with some common attributes and methods.
2. Encapsulation: Encapsulation is the technique used to hide the data and methods within an object and protect it from outside interference. In the BankAccount class, the data members accountNumber, accountHolderName, and balance should be declared private and accessed through public methods, thus ensuring encapsulation.
3. Polymorphism: Polymorphism allows us to perform a single action in different ways. In this assignment, polymorphism is achieved in two ways - Method Overriding (run-time polymorphism) and Method Overloading (compile-time polymorphism).
   * For example, the CalculateInterest() method in the BankAccount class is overridden in each of the subclasses (SavingsAccount, CheckingAccount, LoanAccount) to provide a unique calculation of interest. This is Method Overriding.
   * The Deposit() and Withdraw() methods in the BankAccount class can be overloaded with different parameters to allow different ways of depositing or withdrawing money. This is Method Overloading.
4. Inheritance: Inheritance is the mechanism of basing a class upon another class, retaining similar implementation. In this assignment, SavingsAccount, CheckingAccount, and LoanAccount are subclasses that inherit from the superclass BankAccount.
5. Interfaces: Interfaces contain complete abstract methods and constants. They are typically used to provide common functionality to unrelated classes. In this assignment, you need to create the ITransaction interface, and it should be implemented by the SavingsAccount, CheckingAccount, LoanAccount classes. This interface contains the abstract methods ExecuteTransaction(decimal amount) and PrintTransaction() that are implemented by the subclasses.
6. Relationships:
   * Association: It is a relationship between two classes that is best described as a "uses-a" and "has-a" relationship. In the context of this assignment, the Bank class is associated with the BankAccount class, as it manages the different bank accounts.
   * Aggregation: It represents the "Has-a" relationship. Aggregation is a specialized form of Association where all objects have their own lifecycle, but there is ownership. In this assignment, the Bank class is an aggregate of BankAccount objects.
   * Composition: It is a "part-of" relationship. In composition, if the parent object is deleted, then the child object also loses its status. In this assignment, there is no composition relationship.
   * Dependency: It is a relationship between two or more objects where a change in one object causes a change in another. The BankAccount objects are dependent on the Bank object as the bank account objects need the bank object for their creation.
   * Realization/Implementation: It is a relationship between the interface and the class that implements the interface. In this assignment, the SavingsAccount, CheckingAccount, LoanAccount classes implement the ITransaction interface.

Your goal is to understand these concepts and the problem statement, and implement the solution while properly applying these concepts. Please remember to follow all the necessary error checking and validation steps while dealing with financial data. The goal of this assignment is to provide you a concept about how things work in real-world applications. Implementing a robust, error-free financial system requires careful consideration and extensive testing beyond the scope of this assignment.