Software Engineering
Practical work 1

Objective: Reminder on Object-Oriented Programming

Encapsulation

Encapsulation is a fundamental concept in object-oriented programming (OOP) that refers to the bundling of data (attributes) and the methods (functions) that operate on the data into a single unit or class. It is a mechanism of hiding the internal details of an object, including its state and functionality, from other objects and allowing access only through a public set of functions. This concept helps in preventing outside interference and misuse of the data, ensuring that the object's internal state is protected, and exposing only what is necessary to the user of the class

Example: Defining a class with private fields and public methods to access those fields.

```
class Circle {
        private double radius;
        public Circle(double r) { radius = r; }
        public double GetArea() { return Math.PI * radius * radius; }
}
```

Exercise 1: Create a `**Rectangle**` class with private fields for length and width and use public methods to calculate the **area** and **perimeter**.

Inheritance

a fundamental concept of object-oriented programming (OOP) that allows the creation of a new class derived from an existing one, enabling the new class to reuse, extend or modify the behaviors defined in the existing class. The existing class is termed as the "base class," and the new class is the "derived class." Inheritance promotes code reusability and establishes a relationship between the base and the derived classes, where the derived class can inherit fields and methods of the base class

Example: Creating a `Vehicle` base class and deriving a `Car` class from it.

```
class Vehicle {
        public string brand;
        public void Honk() { Console.WriteLine("Honk!"); }
 }
 class Car : Vehicle {
        public string model;
}
```

Exercise 2: Define a `Bird` class and extend it to create different bird species classes, such as `Sparrow` and `Eagle`.

Polymorphism

Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different types to be treated as objects of a common super type. It enables one entity, such as a method or operator, to have multiple forms. In practical terms, polymorphism allows for methods in different classes that do related things to have the same name, enhancing the readability and maintainability of the code by allowing for more uniform syntax. Polymorphism is often expressed as "one interface, many implementations."

Example: Overriding the base class method in the derived class.

```
class Animal {
        public virtual void MakeSound() { Console.WriteLine("Some sound"); }
}
class Dog : Animal {
        public override void MakeSound() { Console.WriteLine("Bark"); }
}
```

Exercise 3: Create an `Animal` class with a virtual `Move` method and override this method in derived classes like `Fish` and `Horse`.

Abstraction

Abstraction in C# and other object-oriented programming languages is a principle used to hide the complex implementation details and display only the essential features of an object. This process aids in reducing programming complexity and allows the programmer to focus on interactions at a higher level. Abstraction is crucial for managing complexity in software development by exposing only the relevant attributes and interactions of entities, while the non-essential details are concealed from the user

Example: Defining an abstract class or an interface which cannot be instantiated and contains abstract methods.

```
abstract class Shape {
        public abstract double GetArea();
}
class Square : Shape {
        private double side;
        public Square(double s) { side = s; }
        public override double GetArea() { return side * side; }
}
```

Exercise 4: Define an abstract `Drawing` class with an abstract `Draw` method and implement this class with concrete classes like `CircleDrawing` and `RectangleDrawing`.

Exercise 5:

1. Create an abstract class Animal
    a. It should have properties like name, age, species, and diet.
    b. It should have methods like eat() and sleep().
2. Implement Encapsulation:
    a. Encapsulate properties of the Animal class and allow them to be accessed and modified using proper getter and setter methods.
3. Implement Inheritance:
    a. Create different animal classes such as Mammal, Bird, Reptile inheriting from the Animal abstract class.
    b. These classes should have additional properties unique to them, for example, Mammal can have furColor, Bird can have wingSpan, and Reptile can have scaleType.
4. Implement Polymorphism:
    a. Override the eat() and sleep() methods in each derived class to provide class-specific implementations.
    b. Create an interface IWalk with a method walk(). Implement this interface in appropriate classes, allowing different walking behaviors for different animals.
5. Implement Abstraction:
    a. Keep the Animal class and the IWalk interface abstract, ensuring that instances can only be created from the derived classes.
6. Testing:
    a. Create objects for different animal classes and invoke their methods to test if they are working as expected.
    b. Use a list of type Animal to store different animal objects and invoke the eat() and sleep() methods using a loop, demonstrating polymorphism.