Encapsulation and Data Access in C#

In C#, access to class data is controlled using access modifiers and public methods or properties. Unlike some other programming languages, C# does not support friend functions. Instead, object data can be accessed safely through public member functions or properties that are defined inside the class. This approach follows the concept of **encapsulation**, where data members are protected and accessed only through well-defined interfaces. In the following examples, class objects are passed to methods or use their own public methods to perform operations such as calculating averages or summing values, without violating data encapsulation principles.

Let's define a student class that contains the following data members: ID, name, age, grade1, grade2, and grade3. Then implement a friend function that calculates the student's average; check out Program 1.

Program 1

```
using System;

class Student
{
    public string Name { get; }
    public int ID { get; }
    public int Age { get; }
    public int Grade1 { get; }
    public int Grade2 { get; }
    public int Grade3 { get; }

    public Student()
    {
        Console.Write("Enter student's info: ");
        ID = int.Parse(Console.ReadLine()!);
        Name = Console.ReadLine()!;
        Age = int.Parse(Console.ReadLine()!);
        Grade1 = int.Parse(Console.ReadLine()!);
        Grade2 = int.Parse(Console.ReadLine()!);
        Grade3 = int.Parse(Console.ReadLine()!);
    }

    public float Average() => (Grade1 + Grade2 + Grade3) / 3f;
}

class Program
{
    static void Main()
    {
        Student s1 = new Student();
```

```
        Console.WriteLine(s1.Average());
    }}
```

Let us define an `Employee` class that contains the following data members: name, age, salary, andaddress.
The class provides a constructor to initialize its data, and a public read-only property to allow controlled access to the salary value.

In the main function, three `Employee` objects are created. These objects are then passed to a separate method `Sum()`, which calculates and returns the total salaries by accessing the employees' data through their public property. This example demonstrates **encapsulation** and **object interaction** in C#, where data is accessed safely using public interfaces without violating data hiding principles. Check out Program 2.

Program 2

```
using System;

class Employee
{
    public int Salary { get; }

    public Employee(string n, int a, int sal, string adr)
    {
        Salary = sal;
    }
}

class Program
{
    static int Sum(Employee e1, Employee e2, Employee e3)
        => e1.Salary + e2.Salary + e3.Salary;

    static void Main()
    {
        Employee e1 = new Employee("Mustafa", 42, 1500, "Baghdad");
        Employee e2 = new Employee("Ahmed", 39, 2500, "Mosul");
        Employee e3 = new Employee("Muhannad", 40, 1200, "kut");

        Console.WriteLine(Sum(e1, e2, e3));
    }
}
```

In C#, encapsulation is achieved by hiding class data members and providing controlled access to them using **properties**. Properties use **the `get` and `set`** accessors to read or modify the values of private variables in a safe and organized way. The `get` accessor is used to **retrieve** the value of a data member, while the `set` accessor is used to **assign or update** its value. By using `get` and `set`, programmers can apply validation rules, restrict access, or make data read-only or write-only when needed.

For example, a property that uses only the `get` accessor allows the value to be read but prevents it from being modified from outside the class.
This is useful when data should be initialized only once, such as inside a constructor, and remain unchanged afterward.

Using properties with `get` and `set` improves **data security**, **maintainability**, and **code clarity**, and represents the standard approach for implementing encapsulation in C#.
Unlike some other languages, C# does not use friend functions; instead, it relies on access modifiers and properties to control access to class data.

Interaction Between Classes in C#

In C#, classes can interact with each other through **public** and **internal** methods without violating encapsulation principles.

Unlike some programming languages, C# does not support the concept of friend classes. Instead, controlled access between classes is achieved using access modifiers, methods, and object collaboration. When one class needs to perform operations on another class, objects can be passed as parameters to methods. The receiving class can then safely access or modify data using well-defined public or internal member functions. This approach ensures data protection, maintains encapsulation, and allows classes to collaborate in a clear and organized way.

Program 4

```
using System;

class AlRafidain
{
    private string name;
    private int balance;

    public AlRafidain(string n, int b) { name = n; balance = b; }

    public int GetBalance() => balance;
    public void Deposit(int amount) => balance += amount;
    public void Withdraw(int amount) => balance -= amount;
```

```
    // internal method to allow transfer logic safely
    internal void Decrease(int amount) => balance -= amount;
}

class AlRasheed
{
    private string name;
    private int balance;

    public AlRasheed(string n, int b) { name = n; balance = b; }

    public int GetBalance() => balance;

    public void Transfer(AlRafidain client)
    {
        Console.Write("How much to transfer?: ");
        int amount = int.Parse(Console.ReadLine()!);

        if (amount <= client.GetBalance())
        {
            balance += amount;
            client.Decrease(amount);
        }
        else
        {
            Console.WriteLine("Not enough balance");
        }
    }
}

class Program
{
    static void Main()
    {
        AlRafidain ahmed = new AlRafidain("Ahmed", 1000);
        ahmed.Deposit(500);
        Console.WriteLine("Ahmed balance : " + ahmed.GetBalance() + "$");
        ahmed.Withdraw(700);
        Console.WriteLine("Ahmed balance now : " + ahmed.GetBalance() +"$");

        AlRasheed ali = new AlRasheed("Ali", 500);
        Console.WriteLine("Ali balance : " + ali.GetBalance() + "$");
        ali.Transfer(ahmed);
        Console.WriteLine("Ali balance : " + ali.GetBalance() + "$");
        Console.WriteLine("Ahmed balance : " + ahmed.GetBalance() + "$");
    }
}
```

In this example, two classes represent two banks: `AlRafidain` and `AlRasheed`.

A client transfers money between the two banks using method calls and object interaction.

The transfer process is implemented through public methods and an internal helper method, ensuring that account balances are modified safely without direct access to private data members.

This example demonstrates **encapsulation**, **controlled data access**, and **collaboration between classes** in C#.

H.W) Write an object-oriented program in C# that represents a **Car**. The class should contain appropriate data members, and a **constructor** that reads values for all data members from the user. Create a **public method** inside the class to display the car price. Then, create a **separate method** that receives a `Car` object as a parameter and increases the car price by