C# Assignments on Abstract Class, Interface and

Partial Class

PROGRAMS:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Reflection;
using System.Runtime.ConstrainedExecution;
using System.Runtime.InteropServices;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace assignmenton29abstractetc
{
    internal class Class1
    {
        //Assignment 1. Abstract Class
        //Create an abstract class Vehicle that has an abstract method StartEngine() and a
concrete method StopEngine(). Create derived classes Car and
        //Motorcycle that implement the StartEngine() method and override it to show specific
behavior for each type of vehicle.

        //abstract class Vehicle
        //{
        //    public virtual void StopEngine()
        //    {
        //        Console.WriteLine("Engine stopped.");
        //    }

        //    public abstract void StartEngine();
        //}

        //class Car : Vehicle
        //{
        //    public override void StartEngine()
        //    {
```

```
        //        Console.WriteLine("Car engine started. Vroom!");
        //    }
        //}

        //class Motorcycle : Vehicle
        //{
        //    public override void StartEngine()
        //    {
        //        Console.WriteLine("Motorcycle engine started. Brrm!");
        //    }
        //}

//-------------------------------------------------------------------------------------------------------------------------
---------------


        //Assignment 2. Virtual Functions
        //Create a base class Animal with a virtual method MakeSound(). Derive classes Dog and
Cat that override the MakeSound() method to provide
        //their specific implementation.
        // public class Animal
        //{
        //    // Virtual method
        //    public virtual void MakeSound()
        //    {
        //        Console.WriteLine("Animal sound");
        //    }
        //}

        //public class Dog : Animal
        //{
        //    // Override the MakeSound method
        //    public override void MakeSound()
        //    {
        //        Console.WriteLine("Bark");
        //    }
        //}

        //public class Cat : Animal
        //{
        //    // Override the MakeSound method
        //    public override void MakeSound()
        //    {
        //        Console.WriteLine("Meow");
```

```
//    }
//}
```

//----------

```
//Assignment 3. Interface
//Create an interface IDrive with a method Drive(). Implement this interface in a Car and Truck class, with each class having its own implementation of Drive().

// Define the IDrive interface
//public interface IDrive
//{
//    void Drive();
//}

//// Implement the Car class
//public class Car : IDrive
//{
//    public void Drive()
//    {
//        Console.WriteLine("The car is driving.");
//    }
//}

//// Implement the Truck class
//public class Truck : IDrive
//{
//    public void Drive()
//    {
//        Console.WriteLine("The truck is hauling a load.");
//    }
//}
```

//----------

```
//Assignment 4. Interface vs.Abstract Class
//Write a program that demonstrates the difference between an abstract class and an interface by creating an abstract class Bird with an
//abstract method Fly(), and an interface ISwim with a method Swim().
```

```
//public abstract class Bird
//{
//    // Abstract method
//    public abstract void Fly();

//    // A regular method
//    public void Eat()
//    {
//        Console.WriteLine("Bird is eating.");
//    }
//}

//public interface ISwim
//{
//    // Method in the interface
//    void Swim();
//}

//public class Duck : Bird, ISwim
//{
//    // Implement the abstract method from Bird
//    public override void Fly()
//    {
//        Console.WriteLine("Duck is flying.");
//    }

//    // Implement the method from ISwim
//    public void Swim()
//    {
//        Console.WriteLine("Duck is swimming.");
//    }
//}

//public class Penguin : Bird
//{
//    // Implement the abstract method from Bird
//    public override void Fly()
//    {
//        Console.WriteLine("Penguins cannot fly.");
//    }
//}

//------------------------------------------------------------------------------------------------------------------------
----------
```

```
//Assignment 5. Static Class
//Create a static class MathOperations with a static method Add() and Multiply(). Demonstrate calling these methods without creating an instance of the class.


//public static class MathOperations
// {
//     // Static method for addition
//     public static int Add(int a, int b)
//     {
//         return a + b;
//     }

//     // Static method for multiplication
//     public static int Multiply(int a, int b)
//     {
//         return a * b;
//     }
// }

//------------------------------------------------------------------------------------------------------------------------------------------

//Assignment 6. Extension Methods
//Create an extension method IsEven() for the int type that returns true if the number is even and false if it is odd.

//public static class IntExtensions
//{
//    // Extension method to check if an integer is even
//    public static bool IsEven(this int number)
//    {
//        return number % 2 == 0;
//    }
//}

//------------------------------------------------------------------------------------------------------------------------------------------
```

```
//Assignment 7. Partial Class
//Create a partial class Person that is defined in two files.One file should have the property Namee and the other file should have the method ShowDetails().


//public partial class Person
//{
//    // Property for Name
//    public string Name { get; set; }
//}
//// PersonDetails.cs
//public partial class Person
//{
//    // Method to show details
//    public void ShowDetails()
//    {
//        Console.WriteLine($"Name: {Name}");
//    }
//}

//-------------------------------------------------------------------------------------------------------------------------------

//Assignment 8. Partial Methods

//Create a partial class Employee with a partial method CalculateSalary(). Implement the partial method in another part of the class and demonstrate its usage.


//public partial class Employee
//{
//    // Declaration of the partial method
//    partial void CalculateSalary();

//    // Method to call the partial method
//    public void ShowSalary()
//    {
//        CalculateSalary(); // Calls the partial method
//    }
//}

//// EmployeeDetails.cs
//public partial class Employee
```

```
    //{
    //    // Implementation of the partial method
    //    partial void CalculateSalary()
    //    {
    //        // Example salary calculation
    //        double baseSalary = 50000;
    //        double bonus = 5000;
    //        double totalSalary = baseSalary + bonus;
    //        Console.WriteLine($"Total Salary: {totalSalary}");
    //    }
    //}
```

//----------

```
    //Assignment 9. Indexer
    //Create a Library class that contains an array of Book objects.Implement an indexer that allows accessing the books by index.
    //Write a method to display all the books in the library.

    //public class Book
    //{
    //    public string Title { get; set; }
    //    public string Author { get; set; }
    //    public int Year { get; set; }

    //    public Book(string title, string author, int year)
    //    {
    //        Title = title;
    //        Author = author;
    //        Year = year;

    //    }
    //}

    //public class Library
    //{
    //    private Book[] books;

    //    public Library(Book[] books)
    //    {
    //        this.books = books;
    //    }
```

```
    //    public Book this[int index]
    //    {
    //        get
    //        {
    //            if (index >= 0 && index < books.Length)
    //            {
    //                return books[index];
    //            }
    //            else
    //            {
    //                throw new IndexOutOfRangeException();
    //            }
    //        }
    //    }

    //    public void DisplayBooks()
    //    {
    //        foreach (Book book in books)
    //        {
    //            Console.WriteLine($"Title: {book.Title}, Author: {book.Author}, Year: {book.Year}");
    //        }
    //    }
    //}
```

//----------

```
    //Assignment 10. Exception Handling
    //Write a method Divide that takes two integers as input and returns their division. If a
division by zero occurs, catch the exception and display a custom error message.
    //Demonstrate exception handling with a try-catch-finally block.

    //public static int Divide(int dividend, int divisor)
    //{
    //    try
    //    {
    //        return dividend / divisor;
    //    }
    //    catch (DivideByZeroException ex)
    //    {
    //        Console.WriteLine("Error:   Division by zero is not allowed.");
    //        return 0; // Or throw a custom exception
```

```
        //    }
        //    finally
        //    {
        //        Console.WriteLine("Division operation completed.");
        //    }
        //}

//-------------------------------------------------------------------------------------------------------------------------
---------------

        //Assignment 11. Enum
        //Create an enum CarType with values Sedan, SUV, Truck, and Coupe.Write a Car class
with a property Type of type CarType.Write a method that takes a CarType value and displays a
        //message specific to that type of car.

        //define the CarType enum
        //public enum CarType
        //{
        //    SUV,
        //    MiniSUV,
        //    CompactSUV,
        //    Sedan

        //}
        //public class Car3
        //{
        //    //property
        //    public CarType Type { get; set; }

        //    //constructor
        //    public Car3(CarType type)
        //    {
        //        Type = type;
        //    }

        //    public void DisplayCarInfo()
        //    {
        //        switch (Type)
        //        {
        //            case CarType.SUV:
        //                Console.WriteLine("This is a SUV");
        //                break;
        //            case CarType.MiniSUV:
```

```
//              Console.WriteLine("This is a Mini SUV");
//              break;
//          case CarType.CompactSUV:
//              Console.WriteLine("This is a Compact SUV");
//              break;
//          case CarType.Sedan:
//              Console.WriteLine("This is a Sedan");
//              break;
//          default:
//              Console.WriteLine("Not a CarType");
//              break;
//      }
//  }
//}
```

```
//-----------------------------------------------------------------------------------------------------------------------
---------------
```

```
    //Assignment 12. Attributes
    //Define a custom attribute DeveloperAttribute that takes the name of the developer and
the date when the code was last modified.Apply this attribute to a class Calculator and its
method Add.
    //Retrieve and display the attribute information at runtime.

    //public class DeveloperAttribute : Attribute
    //{
    //  public string DeveloperName { get; }
    //  public string LastModifiedDate { get; }

    //  public DeveloperAttribute(string developerName, string lastModifiedDate)
    //  {
    //      DeveloperName = developerName;
    //      LastModifiedDate = lastModifiedDate;
    //  }
    //}
    //// Apply the DeveloperAttribute to the Calculator class and its Add method
    //[Developer("John Doe", "2024-10-29")]
    //public class Calculator
    //{
    //  [Developer("Jane Smith", "2024-10-29")]
    //  public int Add(int a, int b)
    //  {
```

```
        //        return a + b;
        //    }
        //}



    }
}
```

MAIN PROGRAMS:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Reflection;
using System.Runtime.ConstrainedExecution;
using System.Security.AccessControl;
using System.Text;
using System.Threading.Tasks;
using static assignmenton29abstractetc.Class1;

namespace assignmenton29abstractetc
{
    internal class Program
    {
        static void Main(string[] args)
        //Assignment 1. Abstract Class
        //{
        //    Car car = new Car();
        //    Motorcycle motorcycle = new Motorcycle();

        //    car.StartEngine();
        //    car.StopEngine();

        //    motorcycle.StartEngine();
        //    motorcycle.StopEngine();

        //}

//-------------------------------------------------------------------------------------------------------------------------
----------
```

```
        //Assignment 2. Virtual Functions
        //{
        //    Animal myDog = new Dog();
        //    Animal myCat = new Cat();

        //    myDog.MakeSound(); // Output: Bark
        //    myCat.MakeSound(); // Output: Meow
        //}

//------------------------------------------------------------------------------------------------------------------------
----------------


        //Assignment 3. Interface
        //{
        //    IDrive myCar = new Car();
        //    IDrive myTruck = new Truck();

        //    myCar.Drive();  // Output: The car is driving.
        //    myTruck.Drive(); // Output: The truck is hauling a load.
        //}

//------------------------------------------------------------------------------------------------------------------------
----------------


        //Assignment 4. Interface vs.Abstract Class
        //{
        //    Duck myDuck = new Duck();
        //    myDuck.Fly(); // Output: Duck is flying.
        //    myDuck.Swim(); // Output: Duck is swimming.
        //    myDuck.Eat(); // Output: Bird is eating.

        //    Penguin myPenguin = new Penguin();
        //    myPenguin.Fly(); // Output: Penguins cannot fly.
        //    myPenguin.Eat(); // Output: Bird is eating.
        //}

//------------------------------------------------------------------------------------------------------------------------
----------------
```

```
        //Assignment 5. Static Class
        //{
        //    // Calling static methods directly without creating an instance
        //    int sum = MathOperations.Add(5, 10);
        //    int product = MathOperations.Multiply(5, 10);

        //    Console.WriteLine($"Sum: {sum}"); // Output: Sum: 15
        //    Console.WriteLine($"Product: {product}"); // Output: Product: 50
        //}

//------------------------------------------------------------------------------------------------------------------------
---------------



        //Assignment 6. Extension Methods
        //{
        //int num1 = 4;
        //int num2 = 7;

        //// Using the extension method
        //Console.WriteLine($"{num1} is even: {num1.IsEven()}"); // Output: 4 is even: True
        //Console.WriteLine($"{num2} is even: {num2.IsEven()}"); // Output: 7 is even: False
        //}

//------------------------------------------------------------------------------------------------------------------------
---------------


        //Assignment 7. Partial Class
        //{
        //    Person person = new Person();
        //    person.Name = "Alice";
        //    person.ShowDetails();
        //}

//------------------------------------------------------------------------------------------------------------------------
---------------



        //Assignment 8. Partial Methods
```

```
        //{
        //    Employee employee = new Employee();
        //    employee.ShowSalary(); // Output: Total Salary: 55000
        //}

//-------------------------------------------------------------------------------------------------------------------------------

        //Assignment 9. Indexer
        //{
        //    Book[] libraryBooks = { new Book("The Lord of the Rings", "J.R.R. Tolkien", 1954), new
Book("Pride and Prejudice", "Jane Austen", 1813),
        //    new Book("To Kill a Mockingbird", "Harper Lee", 1960)};

        //    Library library = new Library(libraryBooks);

        //    Console.WriteLine("Accessing books using indexer:");
        //    Console.WriteLine(library[0].Title);
        //    Console.WriteLine(library[1].Author);

        //    Console.WriteLine("\nDisplaying all books:");
        //    library.DisplayBooks();
        //}

//-------------------------------------------------------------------------------------------------------------------------------

        //Assignment 10. Exception Handling

        //{
        //    int result = Divide(10, 2);
        //    Console.WriteLine("Result: " + result);

        //    result = Divide(10, 0);
        //    Console.WriteLine("Result: " + result);
        //}

//-------------------------------------------------------------------------------------------------------------------------------

        //Assignment 11. Enum
```

```
        //{
        //    //creating instances of Car3 class with each CarType value
        //    Car3 suv = new Car3(CarType.SUV);
        //    Car3 msuv = new Car3(CarType.MiniSUV);
        //    Car3 csuv = new Car3(CarType.CompactSUV);
        //    Car3 sedan = new Car3(CarType.Sedan);

        //    //calling methods
        //    suv.DisplayCarInfo();
        //    msuv.DisplayCarInfo();
        //    csuv.DisplayCarInfo();
        //    sedan.DisplayCarInfo();
        //    Console.ReadLine();
        //}

//-------------------------------------------------------------------------------------------------------------------------------

        //Assignment 12. Attributes
        //{
        //    // Get the type of the Calculator class
        //    Type calculatorType = typeof(Calculator);

        //    // Retrieve and display class-level attribute
        //    var classAttribute = (DeveloperAttribute)Attribute.GetCustomAttribute(calculatorType,
typeof(DeveloperAttribute));
        //    if (classAttribute != null)
        //    {
        //        Console.WriteLine($"Calculator Class Developer: {classAttribute.DeveloperName},
Last Modified: {classAttribute.LastModifiedDate}");
        //    }

        //    // Retrieve and display method-level attribute
        //    MethodInfo addMethod = calculatorType.GetMethod("Add");
        //    var methodAttribute = (DeveloperAttribute)Attribute.GetCustomAttribute(addMethod,
typeof(DeveloperAttribute));
        //    if (methodAttribute != null)
        //    {
        //        Console.WriteLine($"Add Method Developer: {methodAttribute.DeveloperName},
Last Modified: {methodAttribute.LastModifiedDate}");
        //    }
        //}
```

```
    }

}
```