Selected files

3 printable files

```
01_DATA_TYPES\09_CONST_READONLY\CircleConst.cs
01_DATA_TYPES\09_CONST_READONLY\CircleReadonly.cs
01_DATA_TYPES\09_CONST_READONLY\Program.cs
```

01_DATA_TYPES\09_CONST_READONLY\CircleConst.cs

```csharp
using System;

namespace _09_CONST_READONLY
{
    /*
        * What is const in C#?
        * --------------------
        * - Definition: A compile-time constant with a fixed value that cannot be changed.
        * - Initialization: Must be set at declaration with a value computable at compile time (e.g., literals like 5.0, "hello", or expressions like 2 + 3).
        * - Scope: Implicitly static, belongs to the class, not an instance (e.g., accessed as CircleConst.Radius).
        * - Modifiability: Never modifiable after declaration.
        * - Supported Types: Limited to primitive types (int, double, string, etc.), enums, or other constants.
        * - Use Case: Ideal for universal constants, like mathematical values (e.g., Math.PI) or fixed configurations.
        * - Example: In CircleConst.cs, 'public const double Radius = 5.0' defines a fixed radius for all instances.
        * --------------------
        */

    class CircleConst
    {
        // Compile-time constant for radius
        public const double Radius = 5.0;

        public double CalculateArea()
        {
            return Radius * Radius * Math.PI;
        }

        public void DisplayRadius()
        {
            Console.WriteLine($"CircleConst Radius (const): {Radius}");
        }
    }
}
```

01_DATA_TYPES\09_CONST_READONLY\CircleReadonly.cs

```csharp
using System;

namespace _09_CONST_READONLY
{

    /*
         * What is readonly in C#?
         * ----------------------
         * - Definition: A field that can be assigned a value only at declaration or in a constructor, remaining fixed thereafter.
         * - Initialization: Can be set at declaration or in a constructor (instance or static), allowing runtime values.
         * - Scope: Can be instance-level (unique per object) or static (shared across all objects).
         * - Modifiability: Modifiable only at declaration or in a constructor; cannot be changed afterward.
         * - Supported Types: Works with any type, including primitive types (int, double) and complex types (objects, arrays).
         * - Use Case: Ideal for values that vary per instance or depend on runtime conditions but must remain fixed after initialization (e.g., IDs, configurations).
         * - Example: In this class, 'public readonly double Radius' is set in the constructor, allowing each CircleReadonly object to have a unique, fixed radius.
         * ----------------------
    */

    class CircleReadonly
    {
        // Instance-level readonly field for radius
        public readonly double Radius;

        // Constructor to initialize readonly field
        public CircleReadonly(double radius)
        {
            Radius = radius; // Set readonly field in constructor
        }

        public double CalculateArea()
        {

            return Radius * Radius * Math.PI;
        }

        public void DisplayRadius()
        {
            Console.WriteLine($"CircleReadonly Radius (readonly): {Radius}");
        }
    }
}
```

01_DATA_TYPES\09_CONST_READONLY\Program.cs

```csharp
using System;

namespace _09_CONST_READONLY
{

    /*
    * Key Differences Between const and readonly:
    * ----------------------------------------------------------------------------------
    * Feature            | const                    | readonly
    * -----------------------------------------------------------------------------------
    * Initialization     | At declaration           | At declaration or in constructor
    * Scope              | Implicitly static        | Instance-level or static
    * Modifiability      | Never modifiable         | Modifiable only in constructor
    * Supported Types    | Primitive types, enums,  | Any type (primitive or complex)
    *                    | strings                  |
    * Value Source       | Compile-time only        | Runtime (e.g., constructor params)
    * Flexibility        | Same value for all       | Different values per instance
    * Use Case           | Universal constants      | Instance-specific fixed values
    *                    | (e.g., Math.PI)          | (e.g., ID, config)
    * ------------------------------------------------------------------------------------
    */

    class Program
    {
        static void Main()
        {
            // Using CircleConst (const)
            CircleConst circleConst = new CircleConst();
            circleConst.DisplayRadius();
            Console.WriteLine($"CircleConst Area: {circleConst.CalculateArea()}");

            // Compilation error if uncommented:
            // CircleConst.Radius = 10.0; // const cannot be modified

            Console.WriteLine(); // Separator for clarity

            // Using CircleReadonly (readonly)
            CircleReadonly circleReadonly1 = new CircleReadonly(5.0);
            CircleReadonly circleReadonly2 = new CircleReadonly(3.0);
            circleReadonly1.DisplayRadius();
            Console.WriteLine($"CircleReadonly1 Area: {circleReadonly1.CalculateArea()}");
            circleReadonly2.DisplayRadius();
            Console.WriteLine($"CircleReadonly2 Area: {circleReadonly2.CalculateArea()}");

            // Compilation error if uncommented:
            // circleReadonly1.Radius = 10.0; // readonly cannot be modified outside constructor
        }
    }
}
```