C# Fundamentals

9. **Reflection and Custom AttributesObjective:Requirements:**
    - **Build an application that discovers and executes methods based on custom attributes.**
    - **Define a custom attribute (e.g., [Runnable]).**
    - **Create several classes with methods decorated with the [Runnable] attribute.**
    - **Use reflection to scan the current assembly for methods marked with [Runnable].**
    - **Invoke the discovered methods dynamically and display their outputs.**

Program code:

CustomAttributeDefinition.cs

```
using System;

// AttributeUsage(AttributeTargets.Method) ensures this attribute can only be applied to methods.
[AttributeUsage(AttributeTargets.Method)]
public class RunnableAttribute : Attribute
{
}

// RunnableAttribut - A custom attribute used to mark methods that should be run.
```

SampleClass.cs

```
using System;

public class MathTasks
{
   [Runnable]
   public void Add()
   {
      Console.WriteLine("2 + 3 = " + (2 + 3));
```

```
    }

    public void Subtract()
    {
        Console.WriteLine("This won't run because it has no attribute.");
    }
}

public class GreetingTasks
{
    [Runnable]
    public void SayHello()
    {
        Console.WriteLine("Hello from GreetingTasks!");
    }

    [Runnable]
    public void SayBye()
    {
        Console.WriteLine("Goodbye from GreetingTasks!");
    }
}
```

Program.cs

```
using System;
using System.Linq;
using System.Reflection;

class Program
{
    static void Main()
```

```
{
    Console.WriteLine("Scanning for [Runnable] methods...\n");

    // Assembly.GetExecutingAssembly() - Gets the current compiled program.
    // Get all types in the current assembly
    var types = Assembly.GetExecutingAssembly().GetTypes();

    foreach (var type in types)
    {
        // GetMethods(...) - Returns all public instance methods of a type.
        var methods = type.GetMethods(BindingFlags.Public | BindingFlags.Instance);

        foreach (var method in methods)
        {
            // Check if method has [Runnable] attribute
            var isRunnable = method.GetCustomAttribute(typeof(RunnableAttribute)) != null;

            if (isRunnable)
            {
                Console.WriteLine($"Found: {type.Name}.{method.Name}");

                // Dynamically creates an object of a class.
                var instance = Activator.CreateInstance(type);

                // Dynamically calls the method found.
                method.Invoke(instance, null);
                Console.WriteLine(); // new line
            }
        }
    }
```

```
        Console.WriteLine("All runnable methods executed.");
    }
}
```

// Activator is a class in the System namespace that allows you to create instances of types (objects) dynamically at runtime.

// Assembly is a compiled code library (like .exe or .dll) that contains your application code. Includes: All classes, interfaces, structs, and methods and also Metadata (info about types, attributes, references, etc).

// An attribute is metadata that you attach to code elements like classes, methods, or properties to describe their behavior.

// C# has built-in attributes like [Obsolete], [Serializable], etc.

Output: