# DotNet-FSE Mandatory Hands-On

**WEEK-1** **NAME:** Sri Ranjani Priya P

**EXERCISE 1:**

Implementing the Singleton Pattern

**What I Did:**

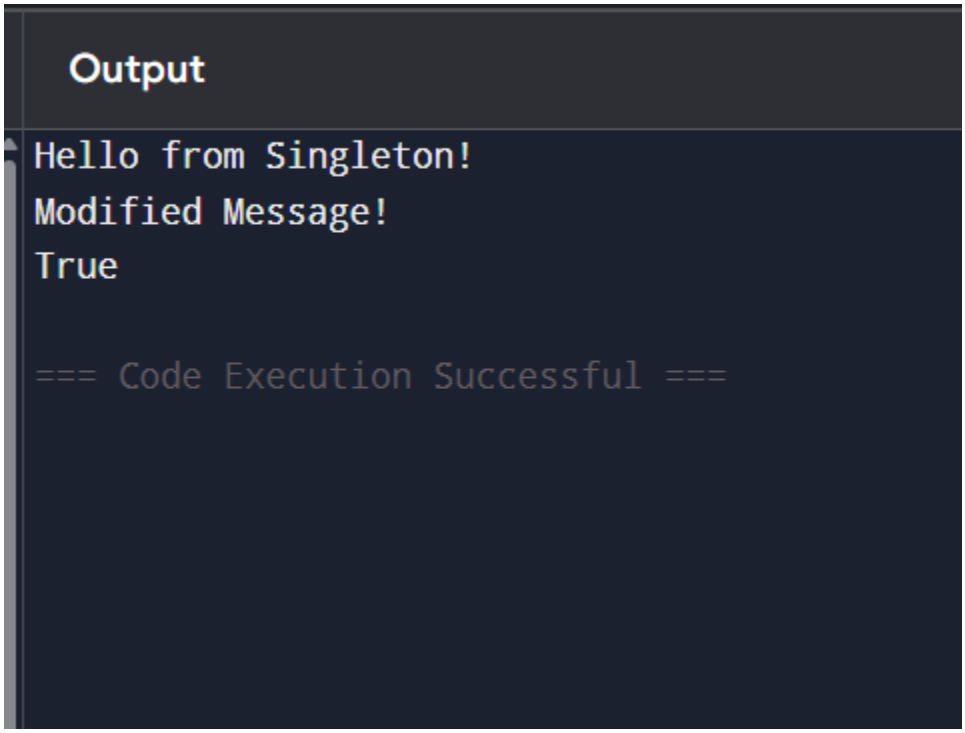
In this exercise, I wrote a program to create only one object of a class using the Singleton pattern. Even if I tried to create the object more than once, it still used the same one. I showed that changing data through one variable also changed it in another, because they both pointed to the same object.

**CODE:(Using C#)**

```
using System;

public sealed class Singleton
{
    private static Singleton instance = null;
    private static readonly object padlock = new object();

    public string Message { get; set; }

    private Singleton()
    {
        Message = "Hello from Singleton!";
    }

    public static Singleton Instance
    {
        get
        {
            lock (padlock)
            {
                if (instance == null)
                    instance = new Singleton();
                return instance;
            }
        }
    }
}
```

```
class Program
{
    static void Main()
    {
        Singleton s1 = Singleton.Instance;
        Singleton s2 = Singleton.Instance;

        Console.WriteLine(s1.Message);
        s2.Message = "Modified Message!";

        Console.WriteLine(s1.Message);
        Console.WriteLine(Object.ReferenceEquals(s1, s2));
    }
}
```

**OUTPUT:**



**EXERCISE 2:**

Implementing the Factory Method pattern

**What I Did:**

This program shows how to create different types of objects using a common method. I created a main class called ‘Product’ and two types: ‘Book’ and ‘Laptop’.Then I made special creator classes that created each type. In the main method, I used these creators to make and show the product details.

**CODE:**

```
using System;

public abstract class Product
{
    public abstract string GetDetails();
}

public class Book : Product
{
    public override string GetDetails() => "Book: C# Programming Guide";
}

public class Laptop : Product
{
    public override string GetDetails() => "Laptop: Dell XPS 13";
}

public abstract class Creator
{
    public abstract Product CreateProduct();
}

public class BookCreator : Creator
{
    public override Product CreateProduct() => new Book();
}

public class LaptopCreator : Creator
{
    public override Product CreateProduct() => new Laptop();
}
```

```
class Program
{
    static void Main()
    {
        Creator creator;

        creator = new BookCreator();
        Console.WriteLine(creator.CreateProduct().GetDetails());

        creator = new LaptopCreator();
        Console.WriteLine(creator.CreateProduct().GetDetails());
    }
}
```

**OUTPUT:**

```
Output
Book: C# Programming Guide
Laptop: Dell XPS 13

=== Code Execution Successful ===
```

**EXERCISE 2:**

E-commerce Platform Search Function

**What I Did:**

In this program, I created a list of products and allowed the user to search by typing a keyword. The program showed all the products whose names matched the keyword. If no match was found, it showed “No products found.”

**CODE:**

```
using System;
using System.Collections.Generic;
using System.Linq;

class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
}

class ECommerceSearch
{
    static void Main()
    {
        List<Product> products = new List<Product>
        {
            new Product { Id = 1, Name = "Laptop" },
            new Product { Id = 2, Name = "Smartphone" },
            new Product { Id = 3, Name = "Smartwatch" },
            new Product { Id = 4, Name = "Tablet" },
            new Product { Id = 5, Name = "Camera" }
        };

        Console.WriteLine("Enter search keyword:");
        string keyword = Console.ReadLine().ToLower();

        var results = products.Where(p => p.Name.ToLower().Contains(keyword)).ToList();

        if (results.Any())
        {
            Console.WriteLine("Search Results:");
            foreach (var product in results)
```

```
            Console.WriteLine($"Id: {product.Id}, Name: {product.Name}");
        }
        else
        {
            Console.WriteLine("No products found.");
        }
    }
}
```

**OUTPUT:**

```
Output
Enter search keyword:
smart
Search Results:
Id: 2, Name: Smartphone
Id: 3, Name: Smartwatch

=== Code Execution Successful ===
```

```
Output
Enter search keyword:
Ipad
No products found.

=== Code Execution Successful ===
```

**EXERCISE 7:**

Financial Forecasting

**What I Did:**

In this exercise, I stored profit amounts for the past 6 months and calculated the average. Then I predicted the profit for the next 3 months by adding ₹500 more each month to the average. I displayed all profits and forecasts.

**CODE:**

```
using System;
using System.Collections.Generic;
using System.Linq;

class FinancialForecast
{
    static void Main()
    {
        List<decimal> monthlyProfits = new List<decimal> { 10000, 12000, 11000, 13000,
14000, 12500 };
        decimal avg = monthlyProfits.Average();

        Console.WriteLine("Last 6 months' profits:");
        foreach (var profit in monthlyProfits)
            Console.WriteLine($"Rs.{profit}");

        Console.WriteLine($"\nAverage Monthly Profit: Rs.{avg}");

        Console.WriteLine("\nForecast for next 3 months:");
        for (int i = 1; i <= 3; i++)
            Console.WriteLine($"Month {i}: Rs.{avg + i * 500}");
    }
}
```

**OUTPUT:**

**Output**

```
Last 6 months' profits:
Rs.10000
Rs.12000
Rs.11000
Rs.13000
Rs.14000
Rs.12500

Average Monthly Profit: Rs.12083.3333333333333333333333333

Forecast for next 3 months:
Month 1: Rs.12583.3333333333333333333333333
Month 2: Rs.13083.3333333333333333333333333
Month 3: Rs.13583.3333333333333333333333333

=== Code Execution Successful ===
```