

1.

```
using System;
```

```
namespace SingletonPatternExample
```

```
{
```

```
// Singleton Class
```

```
public sealed class Logger
```

```
{
```

```
// Private static readonly instance
```

```
private static readonly Lazy<Logger> _instance = new Lazy<Logger>(() => new Logger());
```

```
// Private constructor to prevent instantiation
```

```
private Logger()
```

```
{
```

```
    Console.WriteLine("Logger instance created.");
```

```
}
```

```
// Public property to access the instance
```

```
public static Logger Instance => _instance.Value;
```

```
// Example method
```

```
public void Log(string message)
```

```
{
```

```
    Console.WriteLine($"Log: {message}");
```

```
}
```

```
}
```

```
// Main Program

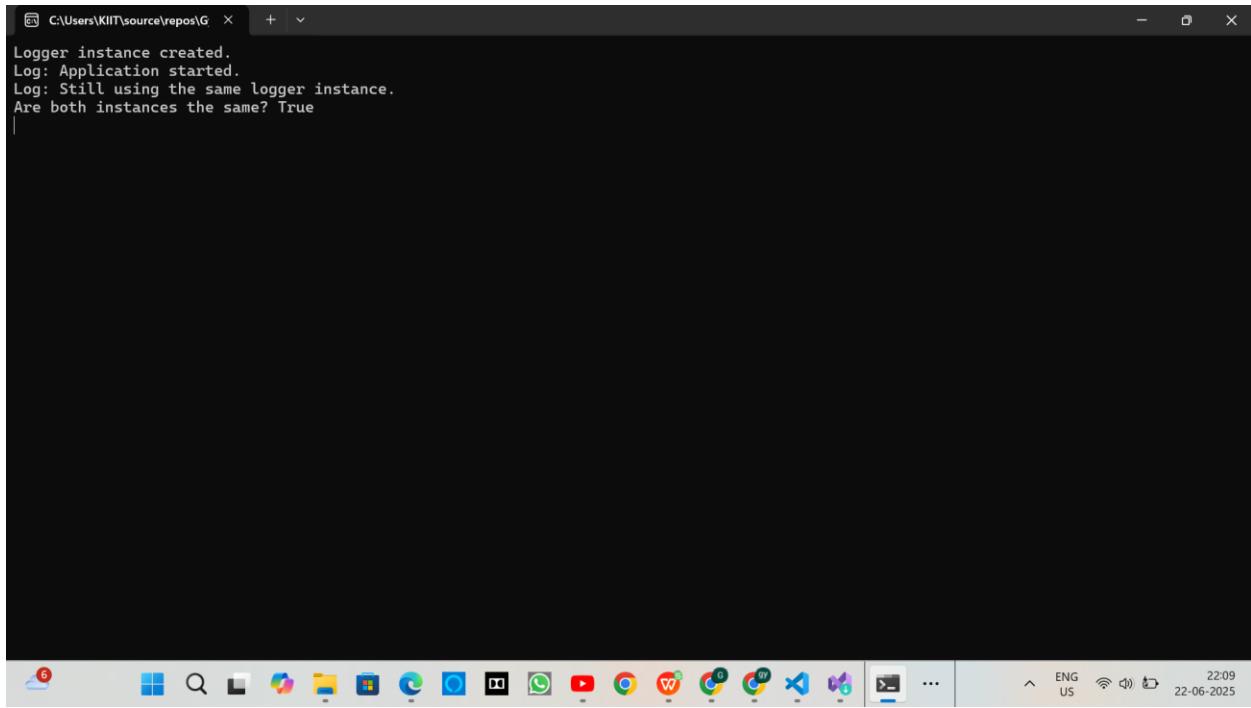
class Program

{
    static void Main(string[] args)
    {
        Logger logger1 = Logger.Instance;
        Logger logger2 = Logger.Instance;

        logger1.Log("Application started.");
        logger2.Log("Still using the same logger instance.");

        Console.WriteLine($"Are both instances the same? {ReferenceEquals(logger1,
logger2)}");

        Console.ReadLine();
    }
}
```



C:\Users\KITT\source\repos\G > + v

```
Logger instance created.  
Log: Application started.  
Log: Still using the same logger instance.  
Are both instances the same? True
```

2.

```
using System;
```

```
namespace FactoryPatternExample
```

```
{
```

```
// Product Interface
```

```
public interface IShape
```

```
{
```

```
    void Draw();
```

```
}
```

```
// Concrete Products
```

```
public class Circle : IShape
```

```
{
```

```
    public void Draw()
```

```
{  
    Console.WriteLine("Drawing a Circle.");  
}  
}  
  
  
public class Rectangle : IShape  
{  
    public void Draw()  
    {  
        Console.WriteLine("Drawing a Rectangle.");  
    }  
}  
  
  
// Abstract Factory  
  
public abstract class ShapeFactory  
{  
    public abstract IShape CreateShape();  
}  
  
  
// Concrete Factories  
  
public class CircleFactory : ShapeFactory  
{  
    public override IShape CreateShape()  
    {  
        return new Circle();  
    }  
}
```

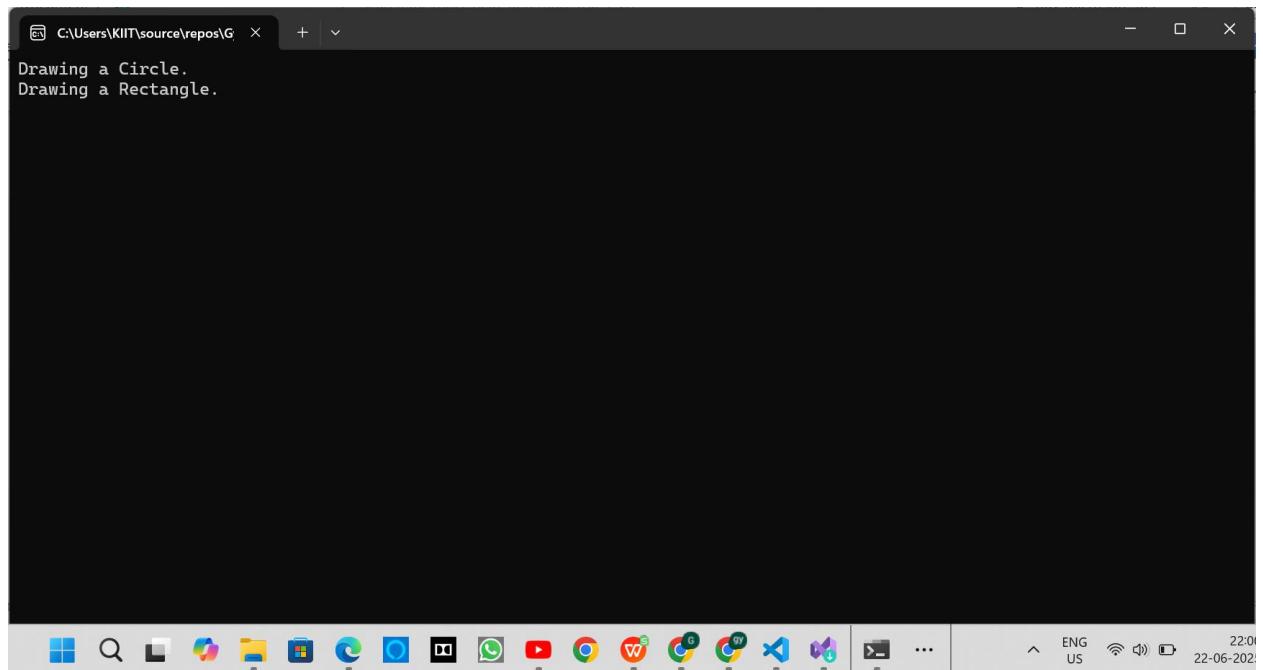
```
}
```

```
public class RectangleFactory : ShapeFactory  
{  
    public override IShape CreateShape()  
    {  
        return new Rectangle();  
    }  
}
```

```
class Program
```

```
{  
    static void Main(string[] args)  
    {  
        ShapeFactory factory;  
  
        // Use CircleFactory  
        factory = new CircleFactory();  
        IShape circle = factory.CreateShape();  
        circle.Draw();  
  
        // Use RectangleFactory  
        factory = new RectangleFactory();  
        IShape rectangle = factory.CreateShape();  
        rectangle.Draw();  
}
```

```
        Console.ReadLine();  
    }  
}  
}
```



3.

```
using System;  
  
using System.Collections.Generic;  
  
using System.Linq;  
  
namespace EcommerceSearchApp  
{  
    // Product class to represent each product  
    public class Product
```

```
{  
    public int Id { get; set; }  
    public string Name { get; set; }  
    public string Category { get; set; }  
    public decimal Price { get; set; }  
}  
  
// Search Engine class implementing search features using DSA  
public class ProductSearchEngine  
{  
    private List<Product> products;  
    private Dictionary<string, List<Product>> categoryMap;  
  
    public ProductSearchEngine()  
{  
        products = new List<Product>();  
        categoryMap = new Dictionary<string, List<Product>>();  
        LoadSampleProducts();  
        BuildCategoryMap();  
    }  
  
    // Load static products into the list  
    private void LoadSampleProducts()  
{  
        products.Add(new Product { Id = 1, Name = "iPhone 14", Category = "Electronics",  
        Price = 79999 });  
    }  
}
```

```
    products.Add(new Product { Id = 2, Name = "Samsung Galaxy S23", Category =
"Electronics", Price = 69999 });

    products.Add(new Product { Id = 3, Name = "Nike Air Max", Category = "Footwear",
Price = 12000 });

    products.Add(new Product { Id = 4, Name = "Adidas Running Shoes", Category =
"Footwear", Price = 10000 });

    products.Add(new Product { Id = 5, Name = "Sony Headphones", Category =
"Electronics", Price = 15000 });

}

// Build a category map for efficient lookup

private void BuildCategoryMap()

{

    categoryMap = products

        .GroupBy(p => p.Category.ToLower())

        .ToDictionary(g => g.Key, g => g.ToList());

}

// Search products by partial name match (case-insensitive)

public List<Product> SearchByName(string keyword)

{

    return products

        .Where(p => p.Name.ToLower().Contains(keyword.ToLower()))

        .ToList();

}

// Search products by category using dictionary
```

```
public List<Product> SearchByCategory(string category)
{
    string key = category.ToLower();
    return categoryMap.ContainsKey(key) ? categoryMap[key] : new List<Product>();
}

// Search by price range

public List<Product> SearchByPriceRange(decimal min, decimal max)
{
    return products
        .Where(p => p.Price >= min && p.Price <= max)
        .ToList();
}

}

class Program
{
    static void Main(string[] args)
    {
        ProductSearchEngine searchEngine = new ProductSearchEngine();

        Console.WriteLine("== E-commerce Product Search ==");
        Console.WriteLine("1. Search by Name");
        Console.WriteLine("2. Search by Category");
        Console.WriteLine("3. Search by Price Range");
        Console.Write("Choose an option (1-3): ");
    }
}
```

```
string choice = Console.ReadLine();

List<Product> results = new List<Product>();

switch (choice)
{
    case "1":
        Console.Write("Enter product name keyword: ");
        string name = Console.ReadLine();
        results = searchEngine.SearchByName(name);
        break;

    case "2":
        Console.Write("Enter product category (e.g., Electronics): ");
        string category = Console.ReadLine();
        results = searchEngine.SearchByCategory(category);
        break;

    case "3":
        Console.Write("Enter minimum price: ");
        decimal min = decimal.Parse(Console.ReadLine());

        Console.Write("Enter maximum price: ");
        decimal max = decimal.Parse(Console.ReadLine());

        results = searchEngine.SearchByPriceRange(min, max);
}
```

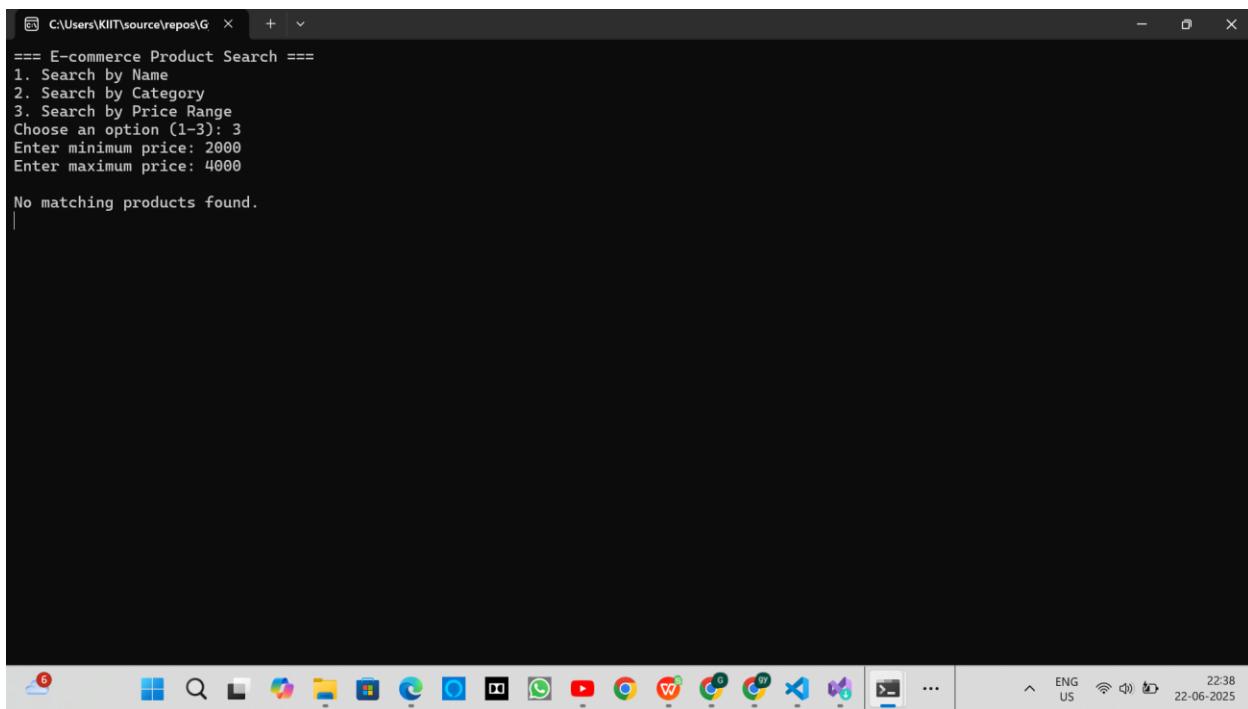
```
        break;

    default:
        Console.WriteLine("Invalid choice.");
        break;
    }

    // Display search results
    if (results.Any())
    {
        Console.WriteLine("\nSearch Results:");
        foreach (var product in results)
        {
            Console.WriteLine($"#{product.Id} | {product.Name} | {product.Category} |
${product.Price}");
        }
    }
    else
    {
        Console.WriteLine("\nNo matching products found.");
    }

    Console.ReadLine();
}

}
```



```
C:\Users\KITT\source\repos\G X + v
==== E-commerce Product Search ====
1. Search by Name
2. Search by Category
3. Search by Price Range
Choose an option (1-3): 3
Enter minimum price: 2000
Enter maximum price: 4000
No matching products found.
```

4.

```
using System;
```

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

```
namespace FinancialForecastApp
```

```
{
```

```
    // Stores forecast for each month
```

```
    public class ForecastRecord
```

```
{
```

```
    public int Month { get; set; }
```

```
    public decimal MonthlySavings { get; set; }
```

```
    public decimal TotalWithInterest { get; set; }
```

```
    public ForecastRecord(int month, decimal savings, decimal total)
```

```

    {
        Month = month;
        MonthlySavings = savings;
        TotalWithInterest = total;
    }
}

// Forecasting logic using List (DSA)
public class FinancialForecaster
{
    public List<ForecastRecord> GenerateForecast(decimal income, decimal expenses,
int months, decimal annualInterest)

    {
        List<ForecastRecord> records = new List<ForecastRecord>();
        decimal monthlySavings = income - expenses;
        decimal monthlyRate = annualInterest / 100 / 12;
        decimal total = 0;

        for (int i = 1; i <= months; i++)
        {
            total = (total + monthlySavings) * (1 + monthlyRate);
            records.Add(new ForecastRecord(i, monthlySavings, total));
        }
    }

    return records;
}

```

```
}
```

```
class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
{
```

```
        Console.WriteLine("==== Financial Forecasting Tool ====");
```

```
        decimal income = ReadDecimal("Enter your monthly income: ₹");
```

```
        decimal expenses = ReadDecimal("Enter your monthly expenses: ₹");
```

```
        int months = ReadInt("Enter number of months to forecast: ");
```

```
        decimal interest = ReadDecimal("Enter expected annual interest rate (e.g., 6.5): ");
```

```
        var forecaster = new FinancialForecaster();
```

```
        var forecast = forecaster.GenerateForecast(income, expenses, months, interest);
```

```
        Console.WriteLine("\nMonth\tSavings\t\tTotal with Interest");
```

```
        foreach (var record in forecast)
```

```
{
```

```
            Console.WriteLine($"{record.Month}\t{record.MonthlySavings:F2}\t\t{record.TotalWithInterest:F2}");
```

```
}
```

```
        var final = forecast[forecast.Count - 1];
```

```
        Console.WriteLine($"\\n Final projected savings: ₹{final.TotalWithInterest:F2}");
```

```
        Console.ReadLine();  
    }  
  
    static decimal ReadDecimal(string prompt)  
    {  
        decimal value;  
        Console.Write(prompt);  
        while (!decimal.TryParse(Console.ReadLine(), out value))  
        {  
            Console.WriteLine("Invalid number. Try again: ");  
        }  
        return value;  
    }  
  
    static int ReadInt(string prompt)  
    {  
        int value;  
        Console.Write(prompt);  
        while (!int.TryParse(Console.ReadLine(), out value))  
        {  
            Console.WriteLine("Invalid number. Try again: ");  
        }  
        return value;  
    }  
}
```

}

```
C:\Users\KII\source\repos\G X + v
== Financial Forecasting Tool ==
Enter your monthly income: ?20000
Enter your monthly expenses: ?17000
Enter number of months to forecast: 7
Enter expected annual interest rate (e.g., 6.5): 7.5

Month    Savings      Total with Interest
1        ?3000.00      ?3018.75
2        ?3000.00      ?6056.37
3        ?3000.00      ?9112.97
4        ?3000.00      ?12188.68
5        ?3000.00      ?15283.60
6        ?3000.00      ?18397.88
7        ?3000.00      ?21531.61

?? Final projected savings: ?21531.61
```

