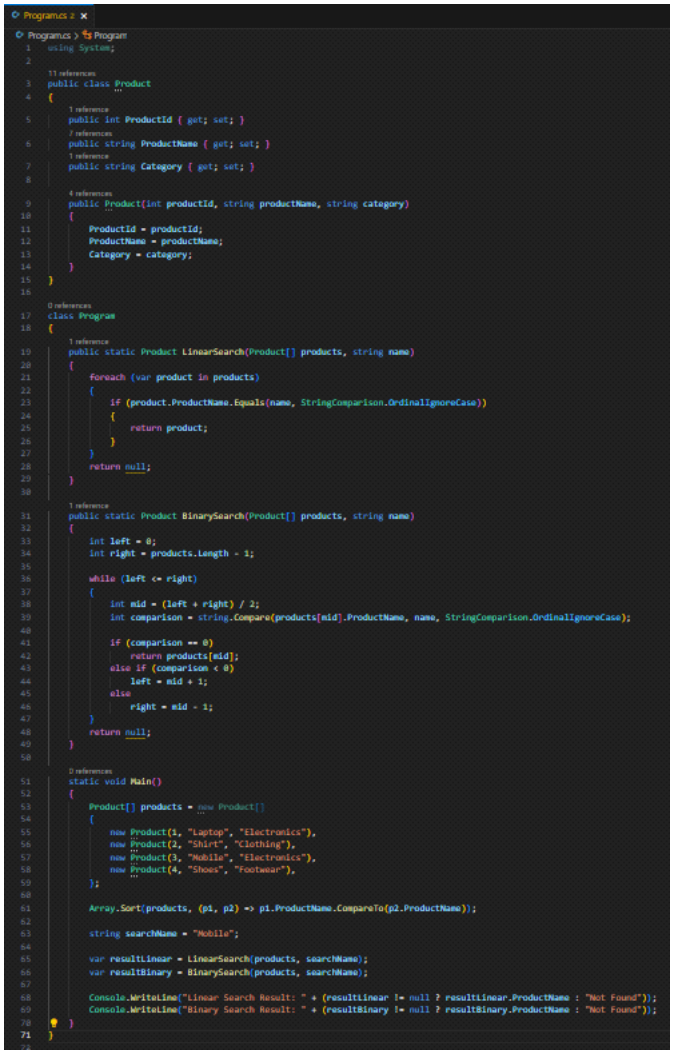
E-commerce Platform Search Function



```
using System;

public class Product
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public string Category { get; set; }

    public Product(int productId, string productName, string category)
    {
        ProductId = productId;
        ProductName = productName;
        Category = category;
    }
}

class Program
{
    public static Product LinearSearch(Product[] products, string name)
    {
        foreach (var product in products)
        {
            if (product.ProductName.Equals(name, StringComparison.OrdinalIgnoreCase))
            {
                return product;
            }
        }
        return null;
    }

    public static Product BinarySearch(Product[] products, string name)
    {
        int left = 0;
        int right = products.Length - 1;

        while (left <= right)
        {
            int mid = (left + right) / 2;
            int comparison = string.Compare(products[mid].ProductName, name, StringComparison.OrdinalIgnoreCase);

            if (comparison == 0)
                return products[mid];
            else if (comparison < 0)
                left = mid + 1;
            else
                right = mid - 1;
        }
        return null;
    }

    static void Main()
    {
        Product[] products = new Product[]
        {
            new Product(1, "Laptop", "Electronics"),
            new Product(2, "Shirt", "Clothing"),
            new Product(3, "Mobile", "Electronics"),
            new Product(4, "Shoes", "Footwear"),
        };

        Array.Sort(products, (p1, p2) => p1.ProductName.CompareTo(p2.ProductName));

        string searchName = "Mobile";

        var resultLinear = LinearSearch(products, searchName);
        var resultBinary = BinarySearch(products, searchName);

        Console.WriteLine("Linear Search Result: " + (resultLinear != null ? resultLinear.ProductName : "Not Found"));
        Console.WriteLine("Binary Search Result: " + (resultBinary != null ? resultBinary.ProductName : "Not Found"));
    }
}
```

OUTPUT:

```
Build succeeded with 2 warning(s) in 4.1s
Linear Search Result: Mobile
Binary Search Result: Mobile
```

FactoryMethodPatternExample

DocumentFactory.cs > DocumentFactory

```
6 references
public abstract class DocumentFactory
{
    6 references
    public abstract IDocument CreateDocument();
}
```

ExcelDocument.cs > ...

```
using System;

1 reference
public class ExcelDocument : IDocument
{
    4 references
    public void Open()
    {
        Console.WriteLine("Opening an Excel document.");
    }
}
```

ExcelDocumentFactory.cs > ExcelDocumentFactory

```
1 reference
public class ExcelDocumentFactory : DocumentFactory
{
    4 references
    public override IDocument CreateDocument()
    {
        return new ExcelDocument();
    }
}
```

IDocument.cs > ...

```
10 references
public interface IDocument
{
    6 references
    void Open();
}
```

PdfDocument.cs > ...

```
using System;

1 reference
public class PdfDocument : IDocument
{
    4 references
    public void Open()
    {
        Console.WriteLine("Opening a PDF document.");
    }
}
```

PdfDocumentFactory.cs > PdfDocumentFactory

```
1 reference
public class PdfDocumentFactory : DocumentFactory
{
    4 references
    public override IDocument CreateDocument()
    {
        return new PdfDocument();
    }
}
```

Program.cs > Program > Main

```
using System;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        DocumentFactory wordFactory = new WordDocumentFactory();
        IDocument wordDoc = wordFactory.CreateDocument();
        wordDoc.Open();

        DocumentFactory pdfFactory = new PdfDocumentFactory();
        IDocument pdfDoc = pdfFactory.CreateDocument();
        pdfDoc.Open();

        DocumentFactory excelFactory = new ExcelDocumentFactory();
        IDocument excelDoc = excelFactory.CreateDocument();
        excelDoc.Open();
    }
}
```

WordDocument.cs > ...

```
using System;

1 reference
public class WordDocument : IDocument
{
    4 references
    public void Open()
    {
        Console.WriteLine("Opening a Word document.");
    }
}
```

WordDocumentFactory.cs > WordDocumentFactory

```
1 reference
public class WordDocumentFactory : DocumentFactory
{
    4 references
    public override IDocument CreateDocument()
    {
        return new WordDocument();
    }
}
```

OUTPUT:

```
PS C:\Users\Lenovo\OneDrive\Desktop\C#\FactoryMethodPatternExample> dotnet run
Opening a Word document.
Opening a PDF document.
Opening an Excel document.
```

Financial Forecasting

```
Program.cs > FinancialForecast > Main
1   using System;
2
    0 references
3   class FinancialForecast
4   {
5
        2 references
6       public static double PredictFutureValue(double principal, double rate, int years)
7       {
8           if (years == 0)
9               return principal;
10
11          return (1 + rate) * PredictFutureValue(principal, rate, years - 1);
12      }
13
        0 references
14      static void Main()
15      {
16          double principal = 10000;
17          double rate = 0.05;
18          int years = 5;
19
20          double futureValue = PredictFutureValue(principal, rate, years);
21
22          Console.WriteLine($"Future Value after {years} years: {futureValue:C}");
23      }
24  }
```

OUTPUT:

```
PS C:\Users\Lenovo\OneDrive\Desktop\C#\Financial Forecasting> dotnet run
Future Value after 5 years: ₹ 12,762.82
PS C:\Users\Lenovo\OneDrive\Desktop\C#\Financial Forecasting>
```

SingletonPatternExample

Logger.cs > Logger > GetInstance

```
using System;

4 references
public class Logger
{
    4 references
    private static Logger _instance;
    1 reference
    private static readonly object _lock = new object();

    1 reference
    private Logger()
    {
        Console.WriteLine("Logger Instance Created");
    }

    0 references
    public static Logger GetInstance()
    {
        if (_instance == null)
        {
            lock (_lock)
            {
                if (_instance == null)
                {
                    _instance = new Logger();
                }
            }
        }
        return _instance;
    }

    0 references
    public void Log(string message)
    {
        Console.WriteLine("Log Message: " + message);
    }
}

```

Program.cs > Program > Main

```
using System;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Logger logger1 = Logger.GetInstance();
        Logger logger2 = Logger.GetInstance();

        logger1.Log("This is the first log message.");
        logger2.Log("This is the second log message.");

        if (object.ReferenceEquals(logger1, logger2))
        {
            Console.WriteLine("Both logger instances are the same (Singleton verified).");
        }
        else
        {
            Console.WriteLine("Different logger instances exist (Singleton failed).");
        }
    }
}
```

```
PS C:\Users\Lenovo\OneDrive\Desktop\C#\SingletonPatternExample> dotnet run
C:\Users\Lenovo\OneDrive\Desktop\C#\SingletonPatternExample\Logger.cs(5,27): warning CS8618: Non-nullable field '_instance' must contain a non-null value when exiting constructor. Consider adding the 'required' modifier or declaring the f
ield as nullable.
Logger Instance Created
Log Message: This is the first log message.
Log Message: This is the second log message.
Both logger instances are the same (Singleton verified).
```