**Week 1 Mandatory Hands-On**

**Design Patterns and Principles**

1. **Singleton Pattern:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

***Code:***

**Logger.cs:**

using System;

public class Logger

{

    private static Logger \_inst;

    private Logger()

    {

        Console.WriteLine("Instance created");

    }

    public static Logger getinst()

    {

*if* (\_inst == null)

        {

            \_inst = new Logger();

        }

*return* \_inst;

    }

    public void Log(string msg)

    {

        Console.WriteLine(msg);

    }

}

**Program.cs:**

using System;

class Program

{

    static void Main(String[] args)

    {

        Logger l1 = Logger.getinst();

        Logger l2 = Logger.getinst();

        l1.Log("First log message");

        l2.Log("Second log message");

*if* (ReferenceEquals(l1, l2))

        {

            Console.WriteLine("L1 and L2 are same instances");

        }

*else*

        {

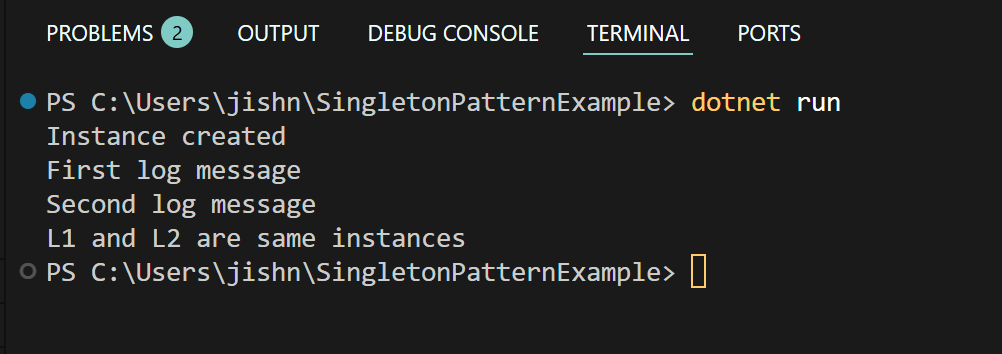
            Console.WriteLine("L1 and L2 are different instances");

        }

    }

}

**Output:**

****

1. **Factory Method Pattern:**

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

***Code:***

**IDocument.cs (Abstract Base class):**

public interface IDocument

{

    void open();

}

**Concrete Document Classes (Word, PDF, Excel)**

**WordDocument.cs:**

public class WordDocument : IDocument

{

    public void open()

    {

        Console.WriteLine("opening a word doc");

    }

}

**PDFDocument.cs :**

public class PDFDocument : IDocument

{

    public void open()

    {

        Console.WriteLine("opening a pdf");

    }

}

**ExcelDocument.cs:**

public class ExcelDocument : IDocument

{

    public void open()

    {

        Console.WriteLine("opening an excel doc");

    }

}

**Abstract Document Factory**

**DocumentFactory.cs:**

public abstract class DocumentFactory

{

    public abstract IDocument CreateDocument();

}

**Concrete Factories for each Document**

**WorddocFact.cs:**

public class WorddocFactory : DocumentFactory

{

    public override IDocument CreateDocument()

    {

*return* new WordDocument();

    }

}

**PdfDocFact.cs:**

public class PdfDocFactory : DocumentFactory

{

    public override IDocument CreateDocument()

    {

*return* new PDFDocument();

    }

}

**ExcelDocFact.cs:**

public class ExcelDocFactory : DocumentFactory

{

    public override IDocument CreateDocument()

    {

*return* new ExcelDocument();

    }

}

**Program.cs:**

using System;

class Program

{

    static void Main(String[] args)

    {

        DocumentFactory fact;

        Console.WriteLine("Choose a doc: word/pdf/excel ");

        string s = Console.ReadLine();

*switch* (s)

        {

*case* "word":

                fact = new WorddocFactory();

*break*;

*case* "pdf":

                fact = new PdfDocFactory();

*break*;

*case* "excel":

                fact = new ExcelDocFactory();

*break*;

*default*:

                Console.WriteLine("Invalid Input");

*return*;

        }

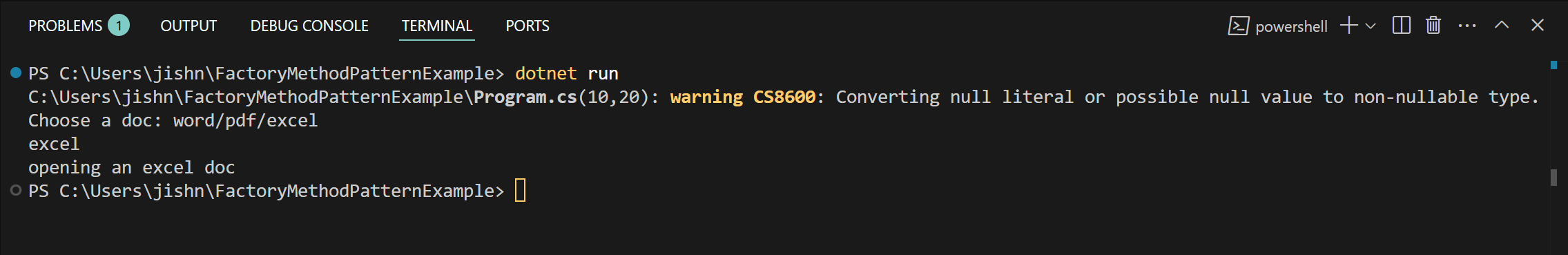
        IDocument doc = fact.CreateDocument();

        doc.open();

    }

}

**Output:**

****

**Data Structures and Algorithms**

1. **E-Commerce Platform Search Function:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

***Code:***

using System;

using System.Linq;

public class Product

{

    public int productID { get; set; }

    public string productName { get; set; }

    public string Category { get; set; }

    public Product(int id, string name, string cat)

    {

        productID = id;

        productName = name;

        Category = cat;

    }

    public override string ToString()

    {

*return* $"ID : {productID}, Name : {productName}, Category: {Category}";

    }

}

class Program

{

    static void Main(string[] args)

    {

        Product[] products = {

            new Product(101, "Laptop", "Electronics"),

            new Product(102,"Shirt","Clothing"),

            new Product(103, "Mouse","Electronics"),

            new Product(104,"Books","Stationery")

        };

        Console.WriteLine("Enter product name to search: ");

        string s = Console.ReadLine() ?? "";

        var res1 = LinearSearch(products, s);

*if* (res1 == null)

        {

            Console.WriteLine($"Not Found: {res1} [Linear Search] ");

        }

*else*

        {

            Console.WriteLine($"Found: {res1} [Linear Search]");

        }

        var sorted = products.OrderBy(p => p.productName).ToArray();

        var res2 = BinarySearch(sorted, s);

*if* (res2 == null)

        {

            Console.WriteLine($"Not Found [Binary Search] ");

        }

*else*

        {

            Console.WriteLine($"Found: {res2} [Binary Search]");

        }

    }

    static Product? LinearSearch(Product[] products, string name)

    {

*foreach* (var p *in* products)

        {

*if* (p.productName.Equals(name, StringComparison.OrdinalIgnoreCase))

            {

*return* p;

            }

        }

*return* null;

    }

    static Product? BinarySearch(Product[] products, string name)

    {

        int l = 0, r = products.Length - 1;

*while* (l <= r)

        {

            int m = l + (r - l) / 2;

            int cmp = string.Compare(products[m].productName, name, true);

*if* (cmp == 0)

            {

*return* products[m];

            }

*else* *if* (cmp < 0)

            {

                l = m + 1;

            }

*else*

            {

                r = m - 1;

            }

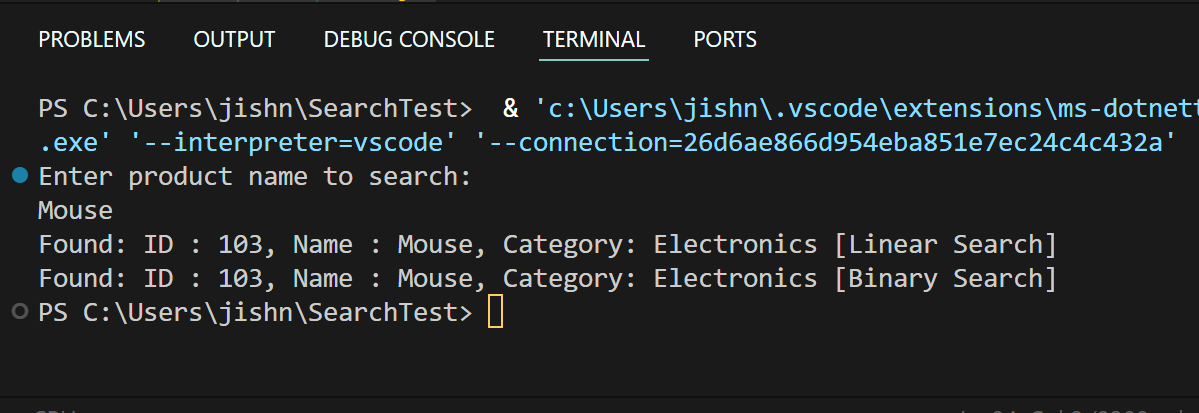
        }

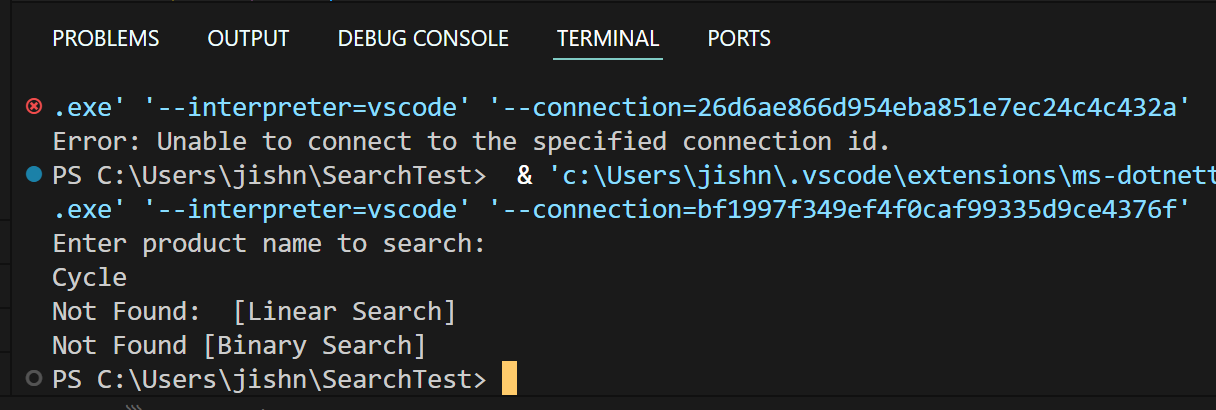
*return* null;

    }

}

**Output:**





1. **Financial Forecasting:**

You are developing a financial forecasting tool that predicts future values based on past data.

***Code:***

using System;

class Program

{

    static void Main()

    {

        Console.WriteLine("enter current investment value:");

        double pv = double.Parse(Console.ReadLine() ?? "0");

        Console.WriteLine("enter annual growth rate (as percentage):");

        double r = double.Parse(Console.ReadLine() ?? "0") / 100;

        Console.WriteLine("enter number of years to forecast:");

        int yrs = int.Parse(Console.ReadLine() ?? "0");

        double fv = ForecastFutureValue(pv, r, yrs);

        Console.WriteLine($"Predicted Future Value after {yrs} years: ₹{fv:F2}");

    }

    static double ForecastFutureValue(double pv, double r, int yrs)

    {

*if* (yrs == 0)

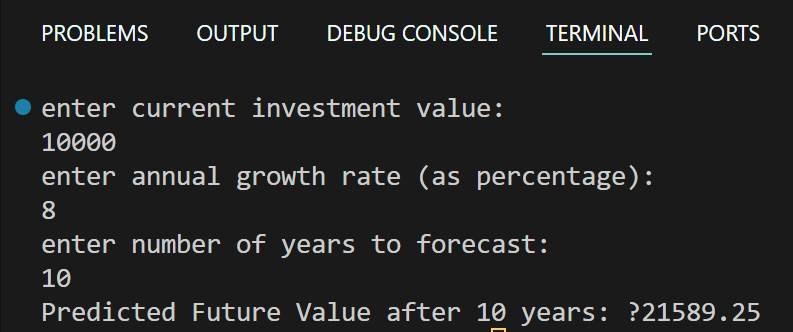
*return* pv;

*return* ForecastFutureValue(pv, r, yrs - 1) \* (1 + r);

    }

}

**Output:**

****