**WEEK 1: DESIGN PATTERNS AND PRINCIPLES - DATA STRUCTURES AND ALGORITHMS**

**DESIGN PATTERNS**:

Question 1: Exercise 1: Implementing the Singleton Pattern

Logger.cs

using System;

namespace Week\_1

{

internal class Logger

{

private static Logger? singleInstance = null;

private Logger()

{

Console.WriteLine("Updated");

}

public static Logger GetInstance()

{

if (singleInstance == null)

{

singleInstance = new Logger();

}

return singleInstance;

}

public void Log(string name)

{

Console.WriteLine("Name: " + name);

}

}

}

Program.cs

using System;

using Week\_1;

namespace Week\_1

{

class Program

{

static void Main(string[] arg)

{

Logger l1 = Logger.GetInstance();

l1.Log("Anu");

Logger l2 = Logger.GetInstance();

l2.Log("Manu");

if (l1 == l2)

{

Console.WriteLine("Both are same Instance");

}

else

{

Console.WriteLine("Different Instances are created");

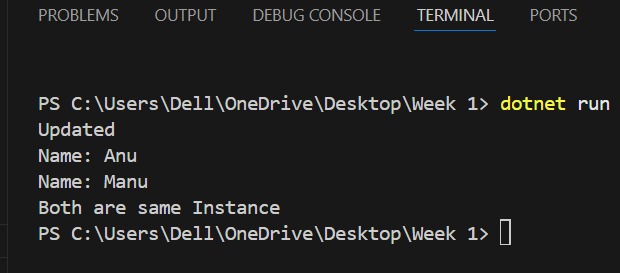
}

}

}

}

OUTPUT:



Question 2: Exercise 2: Implementing the Factory Method Pattern

Document.cs

namespace FactoryMethodPatternExample.Documents

{

public interface Document

{

void Open();

}

}

ExcelDocument.cs

using System;

namespace FactoryMethodPatternExample.Documents

{

public class ExcelDocument : Document

{

public void Open()

{

Console.WriteLine("Opening the Excel");

}

}

}

PDFDocument.cs

using System;

namespace FactoryMethodPatternExample.Documents

{

public class PDFDocument : Document

{

public void Open()

{

Console.WriteLine("Opening PDF Document");

}

}

}

WordDocument.cs

using System;

namespace FactoryMethodPatternExample.Documents

{

public class WordDocument : Document

{

public void Open()

{

Console.WriteLine("Opening Word Document");

}

}

}

DocumentFactory.cs

using FactoryMethodPatternExample.Documents;

namespace FactoryMethodPatternExample.Factories

{

public abstract class DocumentFactory

{

public abstract Document CreateDocument();

}

}

ExcelFactory.cs

using FactoryMethodPatternExample.Documents;

namespace FactoryMethodPatternExample.Factories

{

public class ExcelFactory : DocumentFactory

{

public override Document CreateDocument()

{

return new ExcelDocument();

}

}

}

PDFFactory.cs

using FactoryMethodPatternExample.Documents;

namespace FactoryMethodPatternExample.Factories

{

public class PDFFactory : DocumentFactory

{

public override Document CreateDocument()

{

return new PDFDocument();

}

}

}

WordFactory.cs

using FactoryMethodPatternExample.Documents;

namespace FactoryMethodPatternExample.Factories

{

public class WordFactory : DocumentFactory

{

public override Document CreateDocument()

{

return new WordDocument();

}

}

}

Program.cs

using System;

using FactoryMethodPatternExample.Documents;

using FactoryMethodPatternExample.Factories;

class Program

{

static void Main(string[] args)

{

DocumentFactory factory;

Console.WriteLine("Enter document type (pdf/word/excel): ");

string input = (Console.ReadLine() ?? "").ToLower();

switch (input)

{

case "pdf":

factory = new PDFFactory();

break;

case "word":

factory = new WordFactory();

break;

case "excel":

factory = new ExcelFactory();

break;

default:

Console.WriteLine("Invalid type. Defaulting to PDF.");

factory = new PDFFactory();

break;

}

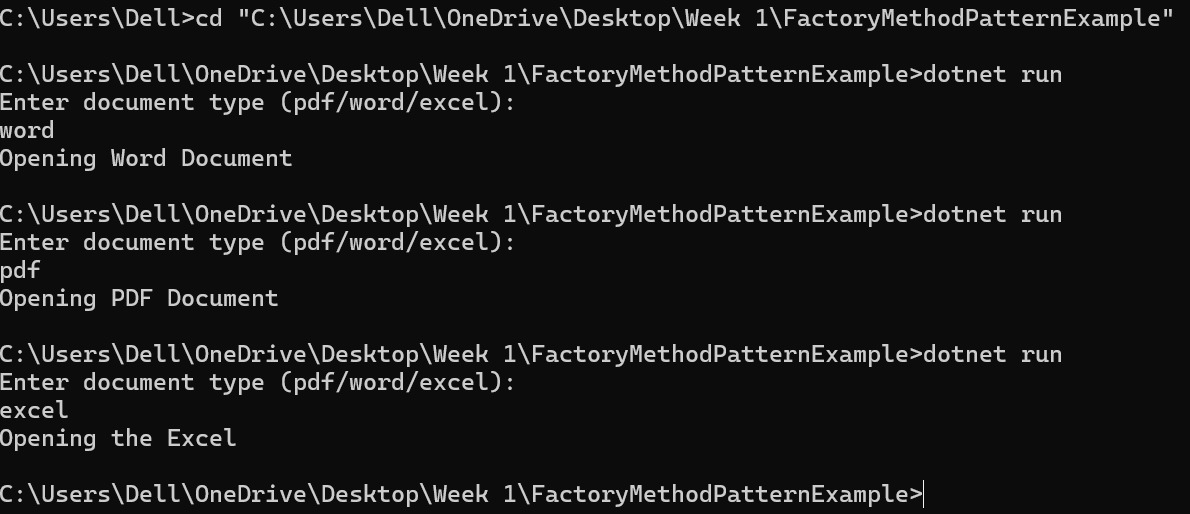
Document document = factory.CreateDocument();

document.Open();

}

}

OUTPUT:



**DATA STRUCTRES AND ALGORITHMS**

Question 3: Exercise 2: Ecommerce Platform Search Function

Binary Search Algorithm:

using System;

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 category)

{

ProductId = id;

ProductName = name;

Category = category;

}

}

class Program

{

public static void Main()

{

Product[] products = new Product[]

{

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shoes", "Fashion"),

new Product(3, "Phone", "Electronics"),

new Product(4, "Shirt", "Fashion"),

new Product(5, "Headphones", "Electronics"),

new Product(6, "Watch", "Accessories")

};

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

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

string input = Console.ReadLine();

Product foundProduct = BinarySearch(products, input);

if (foundProduct != null)

{

Console.WriteLine($"\nProduct found:");

Console.WriteLine($"ID: {foundProduct.ProductId}");

Console.WriteLine($"Name: {foundProduct.ProductName}");

Console.WriteLine($"Category: {foundProduct.Category}");

}

else

{

Console.WriteLine("\nProduct not found.");

}

}

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

{

int left = 0, 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;

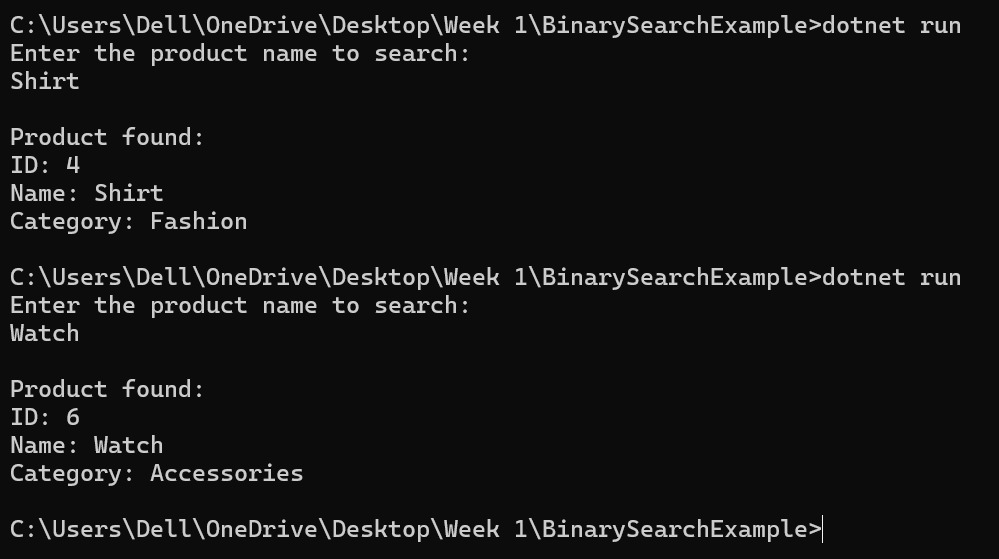
}

return null;

}

}

OUTPUT:



Linear Search Algorithm:

Program.cs

using System;

class Product

{

public int ProductId;

public string ProductName;

public string Category;

public Product(int productId, string productName, string category)

{

ProductId = productId;

ProductName = 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, "Smartphone", "Electronics"),

new Product(104, "Shoes", "Footwear"),

new Product(105, "Watch", "Accessories")

};

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

string searchName = Console.ReadLine();

bool found = false;

foreach (Product product in products)

{

if (product.ProductName.Equals(searchName, StringComparison.OrdinalIgnoreCase))

{

Console.WriteLine($"\nProduct Found:\nID: {product.ProductId}\nName: {product.ProductName}\nCategory: {product.Category}");

found = true;

break;

}

}

if (!found)

{

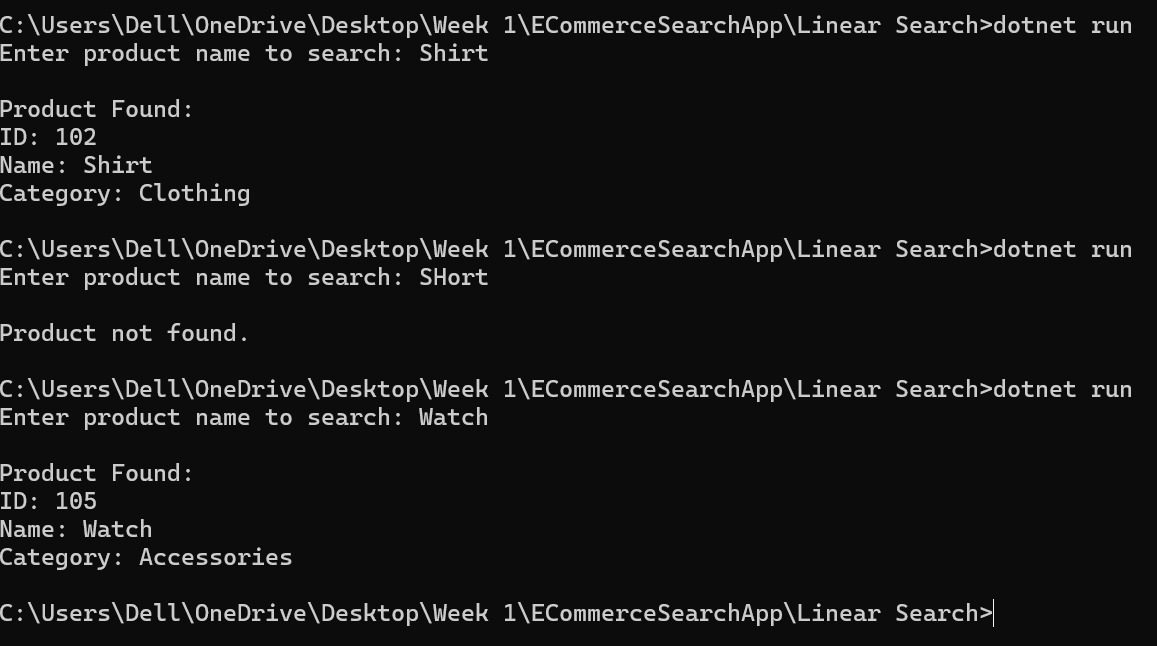
Console.WriteLine("\nProduct not found.");

}

}

}

OUTPUT:



Question 4: Exercise 7: Financial Forecasting

Program.cs

using System;

class Program

{

static double PredictFutureValue(double presentValue, double rate, int years)

{

if (years == 0)

return presentValue;

return PredictFutureValue(presentValue, rate, years - 1) \* (1 + rate);

}

static void Main(string[] args)

{

Console.Write("Enter present value: ");

double presentValue = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter annual growth rate (e.g. 0.05 for 5%): ");

double rate = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter number of years: ");

int years = Convert.ToInt32(Console.ReadLine());

double futureValue = PredictFutureValue(presentValue, rate, years);

Console.WriteLine($"\nFuture value after {years} years: {futureValue:F2}");

}

}

OUTPUT:

