**Exercise 2: E-commerce Platform Search Function**

*Code:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

public class Product

{

public int ProductId { get; set; }

public string? ProductName { get; set; }

public string? Category { get; set; }

}

public interface ISearchStrategy

{

Product? Search(Product[] products, int productId);

}

public class LinearSearch : ISearchStrategy

{

public Product? Search(Product[] products, int productId)

{

return products.FirstOrDefault(p => p.ProductId == productId);

}

}

public class BinarySearch : ISearchStrategy

{

public Product Search(Product[] products, int productId)

{

int left = 0, right = products.Length - 1;

while (left <= right)

{

int mid = (left + right) / 2;

if (products[mid].ProductId == productId)

return products[mid];

else if (products[mid].ProductId < productId)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

}

public class ProductSearchService

{

private readonly ISearchStrategy \_strategy;

public ProductSearchService(ISearchStrategy strategy)

{

\_strategy = strategy;

}

public Product Search(Product[] products, int productId)

{

return \_strategy.Search(products, productId);

}

}

class Program

{

static void Main()

{

var products = new Product[]

{

new Product { ProductId = 101, ProductName = "Laptop", Category = "Electronics" },

new Product { ProductId = 102, ProductName = "Mouse", Category = "Accessories" },

new Product { ProductId = 103, ProductName = "Keyboard", Category = "Accessories" }

};

Array.Sort(products, (a, b) => a.ProductId.CompareTo(b.ProductId)); // for BinarySearch

ISearchStrategy strategy = new BinarySearch(); // can switch to LinearSearch

var service = new ProductSearchService(strategy);

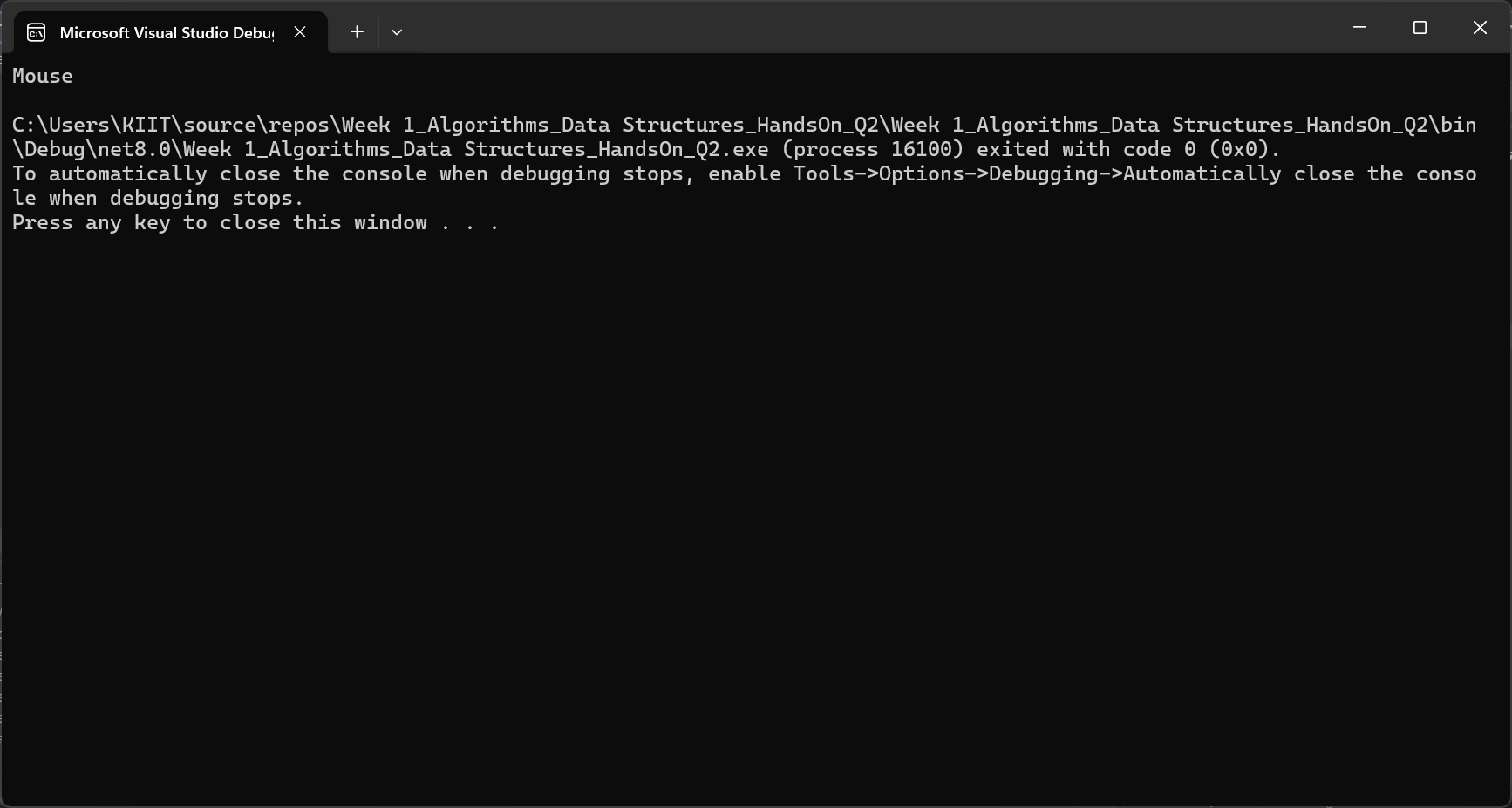
var result = service.Search(products, 102);

Console.WriteLine(result?.ProductName ?? "Product not found");

}

}

*OUTPUT:*

****

**Exercise 7: Financial Forecasting**

*Code:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

public interface IForecastStrategy

{

double Forecast(double initialValue, double growthRate, int years);

}

public class RecursiveForecast : IForecastStrategy

{

public double Forecast(double initialValue, double growthRate, int years)

{

if (years == 0) return initialValue;

return Forecast(initialValue \* (1 + growthRate), growthRate, years - 1);

}

}

public class IterativeForecast : IForecastStrategy

{

public double Forecast(double initialValue, double growthRate, int years)

{

for (int i = 0; i < years; i++)

initialValue \*= (1 + growthRate);

return initialValue;

}

}

public class ForecastService

{

private readonly IForecastStrategy \_strategy;

public ForecastService(IForecastStrategy strategy)

{

\_strategy = strategy;

}

public double Predict(double initialValue, double growthRate, int years)

{

return \_strategy.Forecast(initialValue, growthRate, years);

}

}

class Program

{

static void Main()

{

IForecastStrategy strategy = new RecursiveForecast(); // or IterativeForecast

var service = new ForecastService(strategy);

double result = service.Predict(1000, 0.10, 5);

Console.WriteLine($"Forecasted Value: {result}");

}

}

*OUTPUT:*

