**Implementing the Singleton Pattern**

**Solution:**

using System;

public class Logger{

private static Logger instance;

private Logger(){

Console.WriteLine("Logger initialized");

}

public static Logger GetInstance(){

if (instance == null)

{

instance = new Logger();

}

return instance;

}

public void Log(string message){

Console.WriteLine("LOG " + message);

}

}

class SingletonPatternExample{

static void Main(string[] args){

Logger logger1 = Logger.GetInstance();

logger1.Log("This is my first log message");

Logger logger2 = Logger.GetInstance();

logger2.Log("This is my second log message");

if (logger1 == logger2)

{

Console.WriteLine("Both loggers are in same instance");

}

else

{

Console.WriteLine("Loggers are in different instances. Singleton pattern failed");

}

}

}

OUTPUT:

**Implementing the Factory Method Pattern**

**Solution:**

using System;

namespace FactoryMethodPatternExample{

interface Document{

void Open();

}

class WordDocument : Document{

public void Open(){

Console.WriteLine("Opening Word Document");

}

}

class PdfDocument : Document{

public void Open(){

Console.WriteLine("Opening PDF Document");

}

}

class ExcelDocument : Document{

public void Open()

{

Console.WriteLine("Opening Excel Document");

}

}

abstract class DocumentFactory{

public abstract Document CreateDocument();

}

class WordDocumentFactory : DocumentFactory{

public override Document CreateDocument()

{

return new WordDocument();

}

}

class PdfDocumentFactory : DocumentFactory{

public override Document CreateDocument(){

return new PdfDocument();

}

}

class ExcelDocumentFactory : DocumentFactory{

public override Document CreateDocument(){

return new ExcelDocument();

}

}

class FactoryMethodPatternExample{

static void Main(string[] args){

DocumentFactory wordFactory = new WordDocumentFactory();

Document word = wordFactory.CreateDocument();

word.Open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdf = pdfFactory.CreateDocument();

pdf.Open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excel = excelFactory.CreateDocument();

excel.Open();

}

}

}

**OUTPUT:**

**E-commerce Platform Search Function**

**Solution:**

using System;

using System.Collections.Generic;

class Product{

public int ProductId;

public string ProductName;

public string Category;

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

{

ProductId = id;

ProductName = name;

Category = category;

}

}

class ECommerceSearch{

static Product LinearSearch(List<Product> products, string value){

foreach (var i in products)

{

if (i.ProductName.ToLower() == value.ToLower())

return i;

}

return null;

}

static Product BinarySearch(List<Product> products, string value)

{

int low = 0, high = products.Count - 1;

while (low <= high)

{

int mid = (low + high) / 2;

string midVal = products[mid].ProductName.ToLower();

value = value.ToLower();

if (midVal == value)

return products[mid];

else if (value.CompareTo(midVal) < 0)

high = mid - 1;

else

low = mid + 1;

}

return null;

}

static void Main(){

var products = new List<Product>{

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

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

new Product(3, "Chair", "Furniture"),

new Product(4, "Camera", "Electronics")

};

var result1 = LinearSearch(products, "Camera");

Console.WriteLine("Linear Search: " + (result1 != null ? result1.ProductName + " found" : "Not found"));

products.Sort((a, b) => a.ProductName.CompareTo(b.ProductName));

var result2 = BinarySearch(products, "Laptop");

Console.WriteLine("Binary Search: " + (result2 != null ? result2.ProductName + " found" : "Not found"));

}

}

**OUTPUT:**

**Financial Forecasting:**

**Solution:**

using System;

class FinancialForecasting{

static double FutureVal(double presentVal, double growthRate, int years){

if (years == 0)

return presentVal;

return FutureVal(presentVal, growthRate, years - 1) \* (1 + growthRate);

}

static void Main()

{

double presentVal = 5000;

double growthRate = 0.30;

int years = 7;

double futureVal = FutureVal(presentVal, growthRate, years);

Console.WriteLine($"Future Value after {years} years: Rs.{futureVal:F2}");

}

}

**OUTPUT:**