**Task 1: Logger**

Context: Your team is working on a project where you need to log various events and errors.

You are asked to create a simple logging function that writes messages to a text file with a timestamp.

Example usage:

log\_message(""application.log"", ""User logged in"", ""INFO"")

log\_message(""application.log"", ""Failed login attempt"", ""WARNING"")

Expected Output in application.log:

[2023-04-24 12:34:56] [INFO] User logged in

[2023-04-24 12:35:10] [WARNING] Failed login attempt

using System;

using System.IO;

public class Logger

{

private readonly string \_filePath;

public static void log\_message(string filePath, string message, string logLevel)

{

try {

\_filePath = filePath;

using (var log = File.CreateText(\_filePath))

{

string timestamp = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");

string logEntry = $"[{timestamp}] [{logLevel}] {message}";

log.WriteLine(logEntry);

Console.WriteLine($"Logged: {logEntry}");

}

}

catch (Exception e)

{

Console.WriteLine($"Error logging message: {e.Message}");

}

}

**Task 1.1: Tests**

Write tests scenarios for Logger

using NUnit.Framework;

using System;

using System.IO;

[TestFixture]

public class LoggerTests: BaseLoggerTest // *there are implemented SetUp,TearDown*

*\_logFilePath etc.*

{

[Test]

public void LogMessage\_InfoLevelMsg()

{

//Add the INFO message to LogFile

log\_message(logFilePath, "User logged in", "INFO")

//Read the Log File

string logContent = GetLogContent(logFilePath);

//Check the file content.

Assert.IsTrue(logContent.Contains("[INFO] User logged in"));

}

[Test]

public void LogMessage\_WarningLevelMsg()

{

log\_message(logFilePath, "Failed login attempt", "WARNING")

string logContent = GetLogContent(logFilePath);

Assert.IsTrue(logContent.Contains("[WARNING] Failed login attempt"));

}

// Method reads the file content and returns it.

private string GetLogContent(string filePath)

{

return File.ReadAllText(filePath);

}

}

**Task 2: Inventory Management**

Context: You are developing a simple inventory management system for a small store.

You need to create a function that takes a list of products with their names, prices, and stock levels, and returns a sorted list of products based on a given sort key (name, price, or stock) and order (ascending or descending).

Example Input:

products = [

{""name"": ""Product A"", ""price"": 100, ""stock"": 5},

{""name"": ""Product B"", ""price"": 200, ""stock"": 3},

{""name"": ""Product C"", ""price"": 50, ""stock"": 10}

]

sort\_key = ""price""

ascending = False

Expected Output:

[

{""name"": ""Product B"", ""price"": 200, ""stock"": 3},

{""name"": ""Product A"", ""price"": 100, ""stock"": 5},

{""name"": ""Product C"", ""price"": 50, ""stock"": 10}

]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.Linq;

public class Product

{

public string Name { get; set; }

public int Price { get; set; }

public int Stock { get; set; }

}

public class HelloWorld

{

public static void Main(string[] args)

{

List<Product> products = new List<Product>

{

new Product { Name = "Product A", Price = 100, Stock = 5 },

new Product { Name = "Product B", Price = 200, Stock = 3 },

new Product { Name = "Product C", Price = 50, Stock = 10 }

};

string sortKey = "price";

bool ascending = false;

// Sorting the products

List<Product> sortedProducts = SortProducts(products, sortKey, ascending);

// Printing the sorted products

foreach (var product in sortedProducts)

{

Console.WriteLine($"Name: {product.Name}, Price: {product.Price}, Stock: {product.Stock}");

}

List<Product> SortProducts(List<Product> products, string sort\_key, bool ascending = true)

{

switch (sortKey.ToLower())

{

case "name":

return ascending ? products.OrderBy(p => p.Name).ToList() :

products.OrderByDescending(p => p.Name).ToList();

case "price":

return ascending ? products.OrderBy(p => p.Price).ToList() :

products.OrderByDescending(p => p.Price).ToList();

case "stock":

return ascending ? products.OrderBy(p => p.Stock).ToList() :

products.OrderByDescending(p => p.Stock).ToList();

default:

throw new ArgumentException("Invalid sort key.");

}

}

}

}