**PRESCRIPTION COST MANAGEMENT CONNECTING WITH DB AND UNIT TESING WITH LOG4NET**

**Task 1**

**The week 4 coding first question –**

Problem Statement: Prescription Cost Management

- Define a class: `PrescriptionCost` with the following properties:

- `PrescriptionID` (integer)

- `PatientName` (string)

- `Medication` (string)

- `Cost` (double, in dollars)

- Tasks:

1. Data Input:

- Read N `prescriptionCosts` from the keyboard.

2. Find Lowest Cost Prescription:

- Display the prescription with the lowest cost.

Solve in time complexity of O(N).

Dont use built-in C# sorting or LINQ.

3. Find Second Highest Cost Prescription:

- Display the prescription with the second highest cost.

Solve in time complexity of O(N).

Dont use built-in C# sorting or LINQ.

4. Sort by Medication Name:

- Implement and call your own sorting algorithm.

Dont use built-in C# sorting or LINQ.

**read from SQL, then apply exception handler, log it using log4net, unit tests using MSTest.**

**Program.cs**

using System;

using System.Data.SqlClient;

using System.Runtime.Remoting;

using log4net;

namespace Week4AssessmentApp

{

public class PrescriptionCost

{

public int PrescriptionID { get; set; }

public string PatientName { get; set; }

public string Medication { get; set; }

public double Cost { get; set; }

public PrescriptionCost(int prescriptionID, string patientName, string medication, double cost)

{

PrescriptionID = prescriptionID;

PatientName = patientName;

Medication = medication;

Cost = cost;

}

public override string ToString()

{

return $"PrescriptionID: {PrescriptionID}, PatientName: {PatientName}, Medication: {Medication}, Cost: ${Cost:F2}";

}

}

public class PrescriptionCostService

{

//public static void Read(PrescriptionCost[] prescriptions)

//{

// Console.Write("Enter the number of prescriptions: ");

// int N = int.Parse(Console.ReadLine());

// for (int i = 0; i < N; i++)

// {

// Console.WriteLine($"Enter details for prescription {i + 1}:");

// Console.Write("PrescriptionID: ");

// int id = int.Parse(Console.ReadLine());

// Console.Write("Patient Name: ");

// string patientName = Console.ReadLine();

// Console.Write("Medication: ");

// string medication = Console.ReadLine();

// Console.Write("Cost: ");

// double cost = double.Parse(Console.ReadLine());

// prescriptions[i] = new PrescriptionCost

// {

// PrescriptionID = id,

// PatientName = patientName,

// Medication = medication,

// Cost = cost

// };

// }

//}

private static string connectionString = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Week4AssessmentDb;Integrated Security=True;";

public static void Read(PrescriptionCost[] prescriptions)

{

try

{

using (SqlConnection conn = new SqlConnection(connectionString))

{

string query = "SELECT PrescriptionID, PatientName, Medication, Cost FROM PrescriptionCost";

SqlCommand cmd = new SqlCommand(query, conn);

conn.Open();

SqlDataReader reader = cmd.ExecuteReader();

for (int i = 0; i < prescriptions.Length; i++)

{

if (!reader.Read())

{

throw new ServerException("[0101]Server Errror.");//throw error

}

prescriptions[i] = new PrescriptionCost(

(int)reader["PrescriptionID"],

reader["PatientName"].ToString(),

reader["Medication"].ToString(),

(double)reader["Cost"]

);

}

}

}

catch (SqlException ex)

{

// Handle SQL exceptions

//Console.WriteLine($"SQL Error: {ex.Message}");

throw new ServerException($"[0102]Server Errror.{ex.Message}");//throw Error

}

catch (ServerException ex)

{

throw ex;

}

catch (Exception ex)

{

// Handle other exceptions

//Console.WriteLine($"Error: {ex.Message}");

throw new ServerException($"[0103]Server Errror.{ex.Message}");//throw Error

}

}

public static PrescriptionCost FindLowestCostPrescription(PrescriptionCost[] prescriptions)

{

PrescriptionCost lowestCostPrescription = null;

foreach (var prescription in prescriptions)

{

if (prescription == null) continue;

if (lowestCostPrescription == null || prescription.Cost < lowestCostPrescription.Cost)

{

lowestCostPrescription = prescription;

}

}

return lowestCostPrescription;

}

public static PrescriptionCost FindSecondHighestCostPrescription(PrescriptionCost[] prescriptions)

{

PrescriptionCost highestCost = null;

PrescriptionCost secondHighestCost = null;

foreach (var prescription in prescriptions)

{

if (prescription == null) continue;

if (highestCost == null || prescription.Cost > highestCost.Cost)

{

secondHighestCost = highestCost;

highestCost = prescription;

}

else if (secondHighestCost == null || prescription.Cost > secondHighestCost.Cost)

{

secondHighestCost = prescription;

}

}

return secondHighestCost;

}

public static void SortByMedicationName(PrescriptionCost[] prescriptions)

{

int n = prescriptions.Length;

for (int i = 0; i < n - 1; i++)

{

if (prescriptions[i] == null) continue;

int minIndex = i;

for (int j = i + 1; j < n; j++)

{

if (prescriptions[j] == null) continue;

if (string.Compare(prescriptions[j].Medication, prescriptions[minIndex].Medication, StringComparison.Ordinal) < 0)

{

minIndex = j;

}

}

if (minIndex != i)

{

PrescriptionCost temp = prescriptions[i];

prescriptions[i] = prescriptions[minIndex];

prescriptions[minIndex] = temp;

}

}

}

}

public class Program

{

private static readonly ILog log = LogManager.GetLogger(typeof(Program));

static void Main(string[] args)

{

PrescriptionCost[] prescriptionCosts = new PrescriptionCost[10];

try

{

PrescriptionCostService.Read(prescriptionCosts);

}

catch (ServerException ex)

{

log.Error($"{ex.Message}");

//Console.WriteLine($"{ex.Message}");

}

PrescriptionCost min = PrescriptionCostService.FindLowestCostPrescription(prescriptionCosts);

//Console.WriteLine($"MIN COST : {min}");

log.Info($"MIN COST : {min}");

PrescriptionCost secondMax = PrescriptionCostService.FindSecondHighestCostPrescription(prescriptionCosts);

if (secondMax != null)

{

//Console.WriteLine($"Second Max COST : {secondMax}");

log.Info($"Second Max COST : {secondMax}");

}

else

{

log.Info("Not enough prescriptions to determine the second highest cost.");

// Console.WriteLine("Not enough prescriptions to determine the second highest cost.");

}

PrescriptionCostService.SortByMedicationName(prescriptionCosts);

log.Info($"After SortByMedicationName");

//Console.WriteLine($"After SortByMedicationName");

foreach (var prescription in prescriptionCosts)

{

if (prescription != null)

{

log.Info(prescription);

// Console.WriteLine(prescription);

}

}

}

}

}

**SQL QUERY**

CREATE DATABASE Week4AssessmentDb;

USE Week4AssessmentDb;

CREATE TABLE PrescriptionCost (

PrescriptionID INT PRIMARY KEY,

PatientName NVARCHAR(100),

Medication NVARCHAR(100),

Cost FLOAT

);

INSERT INTO PrescriptionCost

(PrescriptionID, PatientName, Medication, Cost) VALUES

(1,'Rahul', 'Dolo 650', '120'),

(2,'Girish', 'Vicks', '56'),

(3, 'Abijith','Halls', '30');

SELECT \* FROM PrescriptionCost;

**AssemblyInfo.cs**

//add this line at last in AssemblyInfo.cs

[assembly: log4net.Config.XmlConfigurator]

**App.config**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />

</configSections>

<log4net>

<!-- File Appender -->

<appender name="FileAppender" type="log4net.Appender.RollingFileAppender">

<file value="week4assessment\_app\_log.log" />

<appendToFile value="true" />

<rollingStyle value="Size" />

<maxSizeRollBackups value="5" />

<maximumFileSize value="10MB" />

<staticLogFileName value="true" />

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date [%thread] %-5level %logger - %message%newline" />

</layout>

</appender>

<!-- Console Appender -->

<appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date [%thread] %-5level %logger - %message%newline" />

</layout>

</appender>

<!-- Root logger -->

<root>

<level value="ALL" />

<appender-ref ref="FileAppender" />

<appender-ref ref="ConsoleAppender" />

</root>

</log4net>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />

</startup>

</configuration>

**PrescriptionCostServiceTests.cs**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Week4AssessmentApp;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ConsoleApp2;

namespace Week4AssessmentApp.Tests

{

[TestClass()]

public class PrescriptionCostServiceTests

{

[TestMethod()]

public void FindLowestCostPrescriptionTest()

{

PrescriptionCost[] prescriptionCosts = new PrescriptionCost[3];

PrescriptionCostService.Read(prescriptionCosts);

PrescriptionCost expected = new PrescriptionCost(3, "Abijith", "Halls", 30);

PrescriptionCost actual = PrescriptionCostService.FindLowestCostPrescription(prescriptionCosts);

Assert.AreEqual(expected.ToString(), actual.ToString());

}

[TestMethod()]

public void FindSecondHighestCostPrescriptionTest()

{

PrescriptionCost[] prescriptionCosts = new PrescriptionCost[3];

PrescriptionCostService.Read(prescriptionCosts);

PrescriptionCost expected = new PrescriptionCost(2, "Girish", "Vicks", 56);

PrescriptionCost actual = PrescriptionCostService.FindSecondHighestCostPrescription(prescriptionCosts);

Assert.AreEqual(expected.ToString(), actual.ToString());

}

[TestMethod()]

public void SortByMedicationNameTest()

{

PrescriptionCost[] prescriptionCosts = new PrescriptionCost[3];

PrescriptionCostService.Read(prescriptionCosts);

PrescriptionCost expected = new PrescriptionCost(1, "Rahul", "Dolo 650", 120);

PrescriptionCostService.SortByMedicationName(prescriptionCosts);

PrescriptionCost actual = prescriptionCosts[0];

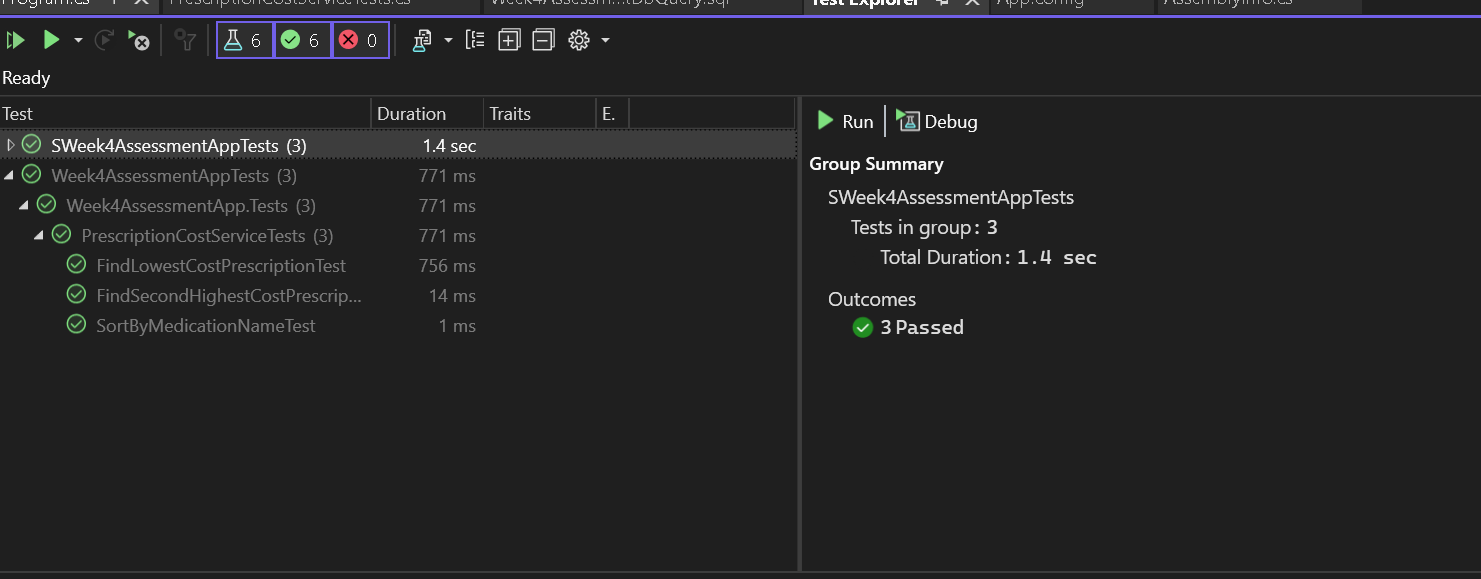
Assert.AreEqual(expected.ToString(), actual.ToString());

}

}

}

**TEST RESULT** (TestExplorer



**OUTPUT**

