**Section06 Singleton Design Pattern**

**Notes:-**

**1-Singleton is component which is instantiated only once**

**2-singelton is used for**

**A-some components it only make sense to have one in the system.**

**\*Database repository: you want single instance of context to access to the database**

**\*Object factory**

**B-constructor call is expensive**

**\*we only do it once**

**\*we provide everyone with the same instance**

**C-Need to take care of lazy initiate and thread safety**

**Lesson01 Singleton Implementation**

**Notes:-**

**1-Singleton is used to initialize by set the constructor private and using static property of the same class with using lazy to initiate whenever it call**

**//we define interface with implement inside the SingletonDatabase**

**namespace SingletonPatternSol.Interfaces{**

**public interface IDatabase{**

**int GetPopulation(string name);}}**

**using System;**

**using System.Collections.Generic;**

**using System.IO;**

**using System.Linq;**

**using System.Security.Cryptography;**

**using System.Text;**

**using System.Threading.Tasks;**

**using Autofac;**

**using MoreLinq;**

**using SingletonPatternSol.Interfaces;**

**using static System.Console;**

**namespace SingletonPatternSol.Models{**

**public class SingletonDatabase : IDatabase{**

**private Dictionary<string, int> capitals;**

**private static int instanceCount;**

**public static int Count => instanceCount;**

**private SingletonDatabase(){**

**WriteLine("Initializing database");**

**//by using the MoreLinq**

**//we read the capital.txt file into key , value pairs as dictionary**

**capitals = File.ReadAllLines(**

**Path.Combine(**

**new FileInfo(typeof(IDatabase).Assembly.Location).DirectoryName, "capitals.txt"))**

**.Batch(2).ToDictionary(**

**list => list.ElementAt(0).Trim(),**

**list => int.Parse(list.ElementAt(1)));}**

**public int GetPopulation(string name){return capitals[name];}**

**// laziness + thread safety not calling until it call through static property**

**private static Lazy<SingletonDatabase> instance = new Lazy<SingletonDatabase>(() =>{**

**instanceCount++;**

**return new SingletonDatabase();});**

**public static IDatabase Instance => instance.Value;}}**

**Lesson02 Problem of Singleton Design Pattern**

**Notes:-**

**1-the problem of the singleton is that you have to use the same static property instance as below**

**//we install the following nugget pacakges**

* **NUnit (3.12.0)**
* **NUnit3TestAdapter (3.17.0)**
* **Microsoft.NET.Test.Sdk (16.7.1)**

**using SingletonPatternSol.Interfaces;**

**using System.Collections.Generic;**

**namespace SingletonPatternSol.Testing{**

**public class DummyDatabase : IDatabase{**

**public int GetPopulation(string name){**

**return new Dictionary<string, int>{**

**["alpha"] = 1,**

**["beta"] = 2,**

**["gamma"] = 3**

**}[name];}}}**

**using SingletonPatternSol.Models;**

**using System.Collections.Generic;**

**namespace SingletonPatternSol.Testing{**

**public class SingletonRecordFinder{**

**public int TotalPopulation(IEnumerable<string> names){**

**int result = 0;**

**foreach (var name in names)**

**result += SingletonDatabase.Instance.GetPopulation(name);**

**return result;}}}**

**using SingletonPatternSol.Interfaces;**

**using System.Collections.Generic;**

**namespace SingletonPatternSol.Testing{**

**public class ConfigurableRecordFinder{**

**private IDatabase database;**

**public ConfigurableRecordFinder(IDatabase database){this.database = database;}**

**public int GetTotalPopulation(IEnumerable<string> names){**

**int result = 0;**

**foreach (var name in names)**

**result += database.GetPopulation(name);**

**return result;}}}**

**using NUnit.Framework;**

**using SingletonPatternSol.Models;**

**namespace SingletonPatternSol.Testing{**

**class DatabaseTesting{**

**/// <summary>**

**/// IMPORTANT: be sure to turn off shadow copying for unit tests in R#!**

**/// </summary>**

**[TestFixture]**

**public class SingletonTests{**

**[Test]**

**public void IsSingletonTest(){**

**var db = SingletonDatabase.Instance;**

**var db2 = SingletonDatabase.Instance;**

**Assert.That(db, Is.SameAs(db2));**

**Assert.That(SingletonDatabase.Count, Is.EqualTo(1));}**

**[Test]**

**public void SingletonTotalPopulationTest(){**

**// testing on a live database**

**var rf = new SingletonRecordFinder();**

**var names = new[] { "Seoul", "Mexico City" };**

**int tp = rf.TotalPopulation(names);**

**Assert.That(tp, Is.EqualTo(17500000 + 17400000));}**

**[Test]**

**public void DependantTotalPopulationTest(){**

**var db = new DummyDatabase();**

**var rf = new ConfigurableRecordFinder(db);**

**Assert.That(**

**rf.GetTotalPopulation(new[] { "alpha", "gamma" }),**

**Is.EqualTo(4));}}}}**

**Lesson03 Singleton using Dependency Injection**

**Notes:-**

**1-**