**Do Factory Singleton Design Pattern**

**Notes: -**

**1-Singleton: - Ensure a class has only one instance and provide a global point access.**

**(with using private constructor and using static property assign to the class itself)**

**namespace DoFactorySingeltonPro{**

**public class Singleton{**

**private static Singleton \_instance;**

**// Constructor is 'protected' so you cannot create instance directly but through static field**

**protected Singleton(){}**

**public static Singleton Instance(){**

**// Uses lazy initialization.**

**// Note: this is not thread safe.**

**if (\_instance == null){\_instance = new Singleton();}**

**return \_instance;}}}**

**using System;**

**using System.Collections.Generic;**

**namespace DoFactorySingeltonPro{**

**class LoadBalancer{**

**private static LoadBalancer \_instance;**

**private List<string> \_servers = new List<string>();**

**private Random \_random = new Random();**

**// Lock synchronization object**

**private static object syncLock = new object();**

**// Constructor (protected)**

**protected LoadBalancer(){**

**// List of available servers**

**\_servers.Add("ServerI");**

**\_servers.Add("ServerII");**

**\_servers.Add("ServerIII");**

**\_servers.Add("ServerIV");**

**\_servers.Add("ServerV");}**

**public static LoadBalancer GetLoadBalancer(){**

**// Support multithreaded applications through**

**// the instance exists) avoids locking each**

**if (\_instance == null){**

**lock (syncLock){**

**if (\_instance == null){**

**\_instance = new LoadBalancer();}}}**

**return \_instance;}**

**public string Server{**

**get{int r = \_random.Next(\_servers.Count);**

**return \_servers[r].ToString();}}}}**