Сингълтън е дизайн патърн, който ограничава инстанцирането на един клас до само един обект. Това е удобно, когато в задачата, която имплементирате трябва да се използва точно един обект от даден клас. Концепцията понякога се обобщава към системи, които функционират по-добре, когато съществува само един обект или има ограничения за броя обекти. Терминът произлиза от математическата концепция за сингълтън. Има критики към патърна, понеже се смята за твърде използван и някои хора казват, че е анти-патърн. Други казват, че тази задача може да се изпълни и без патърна чрез статични методи.

<http://upload.wikimedia.org/wikipedia/commons/b/bd/Singleton.png-диаграма>.-Диаграма.

Код.

|  |
| --- |
| // Singleton pattern -- Real World example |
| using System;  using System.Collections.Generic;  using System.Threading;    namespace DoFactory.GangOfFour.Singleton.RealWorld  {    /// <summary>    /// MainApp startup class for Real-World    /// Singleton Design Pattern.    /// </summary>    class MainApp    {      /// <summary>      /// Entry point into console application.      /// </summary>      static void Main()      {        LoadBalancer b1 = LoadBalancer.GetLoadBalancer();        LoadBalancer b2 = LoadBalancer.GetLoadBalancer();        LoadBalancer b3 = LoadBalancer.GetLoadBalancer();        LoadBalancer b4 = LoadBalancer.GetLoadBalancer();          // Same instance?        if (b1 == b2 && b2 == b3 && b3 == b4)        {          Console.WriteLine("Same instance\n");        }          // Load balance 15 server requests        LoadBalancer balancer = LoadBalancer.GetLoadBalancer();        for (int i = 0; i < 15; i++)        {          string server = balancer.Server;          Console.WriteLine("Dispatch Request to: " + server);        }          // Wait for user        Console.ReadKey();      }    }      /// <summary>    /// The 'Singleton' class    /// </summary>    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        // 'Double checked locking' pattern which (once        // the instance exists) avoids locking each        // time the method is invoked        if (\_instance == null)        {          lock (syncLock)          {            if (\_instance == null)            {              \_instance = new LoadBalancer();            }          }        }          return \_instance;      }        // Simple, but effective random load balancer      public string Server      {        get        {          int r = \_random.Next(\_servers.Count);          return \_servers[r].ToString();        }      }    }  } |