Singleton

- Definition
- UML diagram
- Participants
- Structural code in C#
- Real-world code in C#
- .NET Optimized code in C#

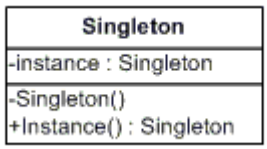
Definition

Ensure a class has only one instance and provide a global point of access to it.

Frequency of use: 

Medium high

UML class diagram



Participants

The classes and objects participating in this pattern are:

- **Singleton (LoadBalancer)**
  - defines an Instance operation that lets clients access its unique instance. Instance is a class operation.
  - responsible for creating and maintaining its own unique instance.

Structural code in C#

This structural code demonstrates the Singleton pattern which assures only a single instance (the singleton) of the class can be created.

```
1.
2.
3.  using System;
4.
5.  namespace DoFactory.GangOfFour.Singleton.Structural
6.  {
7.    /// <summary>
8.    /// MainApp startup class for Structural
9.    /// Singleton Design Pattern.
10.   /// </summary>
11.   class MainApp
12.   {
13.     /// <summary>
14.     /// Entry point into console application.
15.     /// </summary>
16.     static void Main()
17.     {
18.       // Constructor is protected -- cannot use new
19.       Singleton s1 = Singleton.Instance();
20.       Singleton s2 = Singleton.Instance();
21.
22.       // Test for same instance
23.       if (s1 == s2)
24.       {
25.         Console.WriteLine("Objects are the same instance");
26.       }
27.
28.       // Wait for user
29.       Console.ReadKey();
30.     }
31.   }
32.
33.   /// <summary>
34.   /// The 'Singleton' class
35.   /// </summary>
36.   class Singleton
37.   {
38.     private static Singleton _instance;
39.
40.     // Constructor is 'protected'
41.     protected Singleton()
42.     {
43.     }
44.
45.     public static Singleton Instance()
46.     {
47.       // Uses lazy initialization.
48.       // Note: this is not thread safe.
49.       if (_instance == null)
50.       {
51.         _instance = new Singleton();
52.       }
53.
```

```
54.         return _instance;
55.       }
56.     }
57.   }
58.
59.
60.
61.
```

Output

```
Objects are the same instance
```

Real-world code in C#

This real-world code demonstrates the Singleton pattern as a LoadBalancing object. Only a single instance (the singleton) of the class can be created because servers may dynamically come on- or off-line and every request must go throught the one object that has knowledge about the state of the (web) farm.

```
1.
2.
3.  using System;
4.  using System.Collections.Generic;
5.  using System.Threading;
6.
7.  namespace DoFactory.GangOfFour.Singleton.RealWorld
8.  {
9.    /// <summary>
10.   /// MainApp startup class for Real-World
11.   /// Singleton Design Pattern.
12.   /// </summary>
13.   class MainApp
14.   {
15.     /// <summary>
16.     /// Entry point into console application.
17.     /// </summary>
18.     static void Main()
19.     {
20.       LoadBalancer b1 = LoadBalancer.GetLoadBalancer();
21.       LoadBalancer b2 = LoadBalancer.GetLoadBalancer();
22.       LoadBalancer b3 = LoadBalancer.GetLoadBalancer();
23.       LoadBalancer b4 = LoadBalancer.GetLoadBalancer();
24.
25.       // Same instance?
26.       if (b1 == b2 && b2 == b3 && b3 == b4)
27.       {
28.         Console.WriteLine("Same instance\n");
29.       }
30.
31.       // Load balance 15 server requests
32.       LoadBalancer balancer = LoadBalancer.GetLoadBalancer();
33.       for (int i = 0; i < 15; i++)
34.       {
35.         string server = balancer.Server;
36.         Console.WriteLine("Dispatch Request to: " + server);
37.       }
38.
39.       // Wait for user
40.       Console.ReadKey();
41.     }
42.   }
43.
44.   /// <summary>
45.   /// The 'Singleton' class
46.   /// </summary>
47.   class LoadBalancer
48.   {
49.     private static LoadBalancer _instance;
50.     private List<string> _servers = new List<string>();
51.     private Random _random = new Random();
52.
53.     // 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();
      }
    }
  }


```

Output

```
Same instance
```

```
ServerIII
ServerII
ServerI
ServerII
ServerI
ServerIII
ServerI
ServerIII
ServerIV
ServerII
ServerII
ServerIII
ServerIV
ServerII
ServerIV
```

.NET Optimized code in C#

The .NET optimized code demonstrates the same code as above but uses more modern, built-in .NET features.

Here an elegant .NET specific solution is offered. The Singleton pattern simply uses a private constructor and a static readonly instance variable that is lazily initialized. Thread safety is guaranteed by the compiler.

```
1.
2.
3.  using System;
4.  using System.Collections.Generic;
5.
6.  namespace DoFactory.GangOfFour.Singleton.NETOptimized
7.  {
8.    /// <summary>
9.    /// MainApp startup class for .NET optimized
10.   /// Singleton Design Pattern.
11.   /// </summary>
12.   class MainApp
13.   {
14.     /// <summary>
15.     /// Entry point into console application.
16.     /// </summary>
17.     static void Main()
18.     {
19.       LoadBalancer b1 = LoadBalancer.GetLoadBalancer();
20.       LoadBalancer b2 = LoadBalancer.GetLoadBalancer();
21.       LoadBalancer b3 = LoadBalancer.GetLoadBalancer();
22.       LoadBalancer b4 = LoadBalancer.GetLoadBalancer();
23.
24.       // Confirm these are the same instance
25.       if (b1 == b2 && b2 == b3 && b3 == b4)
26.       {
27.         Console.WriteLine("Same instance\n");
28.       }
29.
30.       // Next, load balance 15 requests for a server
31.       LoadBalancer balancer = LoadBalancer.GetLoadBalancer();
32.       for (int i = 0; i < 15; i++)
33.       {
34.         string serverName = balancer.NextServer.Name;
35.         Console.WriteLine("Dispatch request to: " + serverName);
36.       }
37.
38.       // Wait for user
39.       Console.ReadKey();
40.     }
41.   }
42.
43.   /// <summary>
44.   /// The 'Singleton' class
45.   /// </summary>
46.   sealed class LoadBalancer
47.   {
48.     // Static members are 'eagerly initialized', that is,
49.     // immediately when class is loaded for the first time.
50.     // .NET guarantees thread safety for static initialization
51.     private static readonly LoadBalancer _instance =
52.       new LoadBalancer();
53.
```

```
54.       // Type-safe generic list of servers
55.       private List<Server> _servers;
56.       private Random _random = new Random();
57.
58.       // Note: constructor is 'private'
59.       private LoadBalancer()
60.       {
61.         // Load list of available servers
62.         _servers = new List<Server>
63.           {
64.            new Server{ Name = "ServerI", IP = "120.14.220.18" },
65.            new Server{ Name = "ServerII", IP = "120.14.220.19" },
66.            new Server{ Name = "ServerIII", IP = "120.14.220.20" },
67.            new Server{ Name = "ServerIV", IP = "120.14.220.21" },
68.            new Server{ Name = "ServerV", IP = "120.14.220.22" },
69.           };
70.       }
71.
72.       public static LoadBalancer GetLoadBalancer()
73.       {
74.         return _instance;
75.       }
76.
77.       // Simple, but effective load balancer
78.       public Server NextServer
79.       {
80.         get
81.         {
82.           int r = _random.Next(_servers.Count);
83.           return _servers[r];
84.         }
85.       }
86.     }
87.
88.     /// <summary>
89.     /// Represents a server machine
90.     /// </summary>
91.     class Server
92.     {
93.       // Gets or sets server name
94.       public string Name { get; set; }
95.
96.       // Gets or sets server IP address
97.       public string IP { get; set; }
98.     }
99.   }
100.
101.
102.
```

Output

```
Same instance

ServerIV
ServerIV
ServerIII
ServerV
ServerII
ServerV
ServerII
ServerII
ServerI
ServerIV
ServerIV
ServerII
ServerI
ServerV
ServerIV
```

- .NET Design Patterns (/net/design-patterns)
- JavaScript Design Patterns (/javascript/design-patterns)

- JavaScript Tutorial (/tutorial/javascript)
- SQL Tutorial (/sql/tutorial)

- Connection Strings (/reference/connection-strings)
- Visual Studio Shortcuts (/reference/visual-studio-shortcuts)
- C# Coding Standards (/reference/csharp-coding-standards)
- HTML Colors (/reference/html-color-codes)

Our Products

- .NET Design Pattern Framework (/products/net-design-pattern-framework) TM
- PRO .NET Design Pattern Framework (/products/pro-net-design-pattern-framework) TM

- JavaScript + jQuery Pattern Framework (/products/javascript-jquery-design-pattern-framework) TM
- SQL + Database Pattern Framework (/products/sql-database-design-pattern-framework) TM

- Products and Pricing (/products)
