Singleton Pattern

Design Patterns

Motivating Example

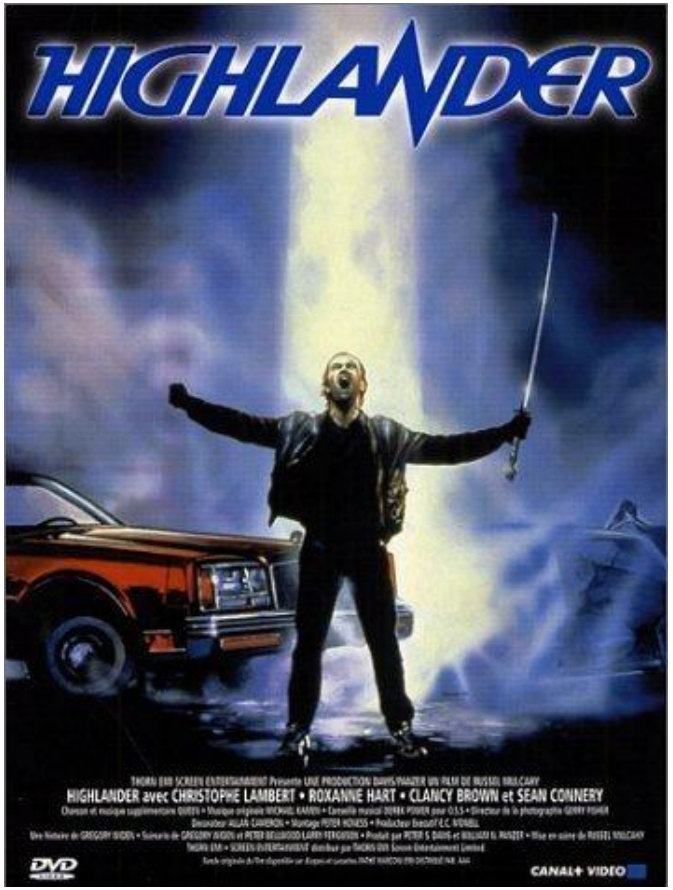
Singleton Pattern

- **Some classes should have exactly one instance**
- **Examples:**
  - Access to a computer's filesystem
  - Access to a network's printer spooler
  - Access to an operating system's windowmanager
- **Commonly, singletons should only be created when they are first needed (e.g. lazy construction)**

Intent

Singleton Pattern

- Ensure a class has only one instance
- Make the class itself responsible for keeping track of its sole instance
- "There can be only one"



http://www.imdb.com/media/rm429429248/tt0091203

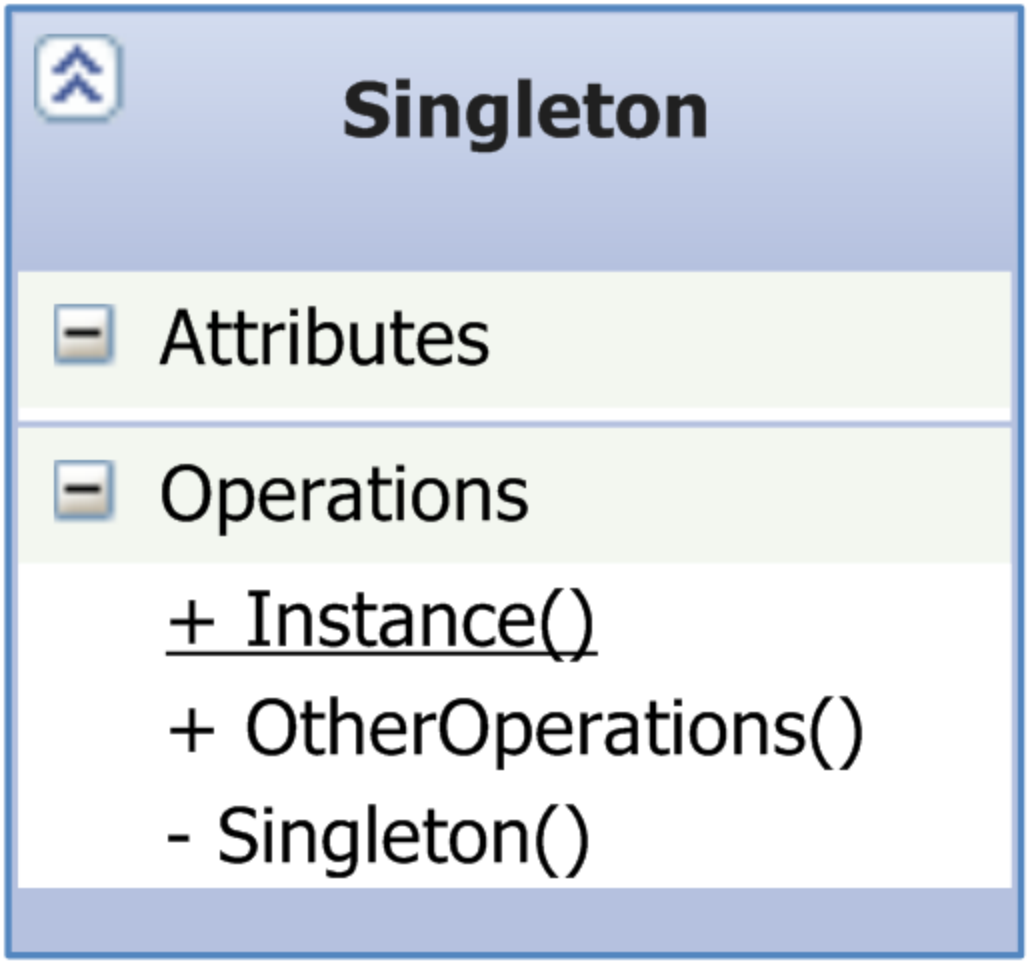
Applicability

Singleton Pattern

- **Use the Singleton Pattern when:**
  - There must be one and only one instance of a class
  - The class must be accessible to clients
  - The class should not require parameters as part of its construction
  - When creating the instance is expensive, a Singleton can improve performance

Structure

Singleton Pattern



Structure (not thread-safe)

Singleton Pattern

```
1  public class Singleton
2  {
3      private static Singleton _instance;
4
5      private Singleton()
6      {
7      }
8
9      public static Singleton Instance
10     {
11         get
12         {
13             if (_instance == null)
14             {
15                 _instance = new Singleton();
16             }
17             return _instance;
18         }
19     }
20 }
```

How It Gets Used

Singleton Pattern

- Call methods on Instance directly
- Assign variables to Instance
- Pass as Parameter

```
public class SampleUsage
{
    public void SomeMethod()
    {
        // Call Singleton's DoStuff() method
        Singleton.Instance.DoStuff();

        // Assign to another variable
        var myObject = Singleton.Instance;
        myObject.DoStuff();

        // Pass as parameter
        SomeOtherMethod(Singleton.Instance);
    }

    private void SomeOtherMethod(Singleton singleton)
    {
        singleton.DoStuff();
    }
}
```

Structure (thread-safe and fast)

Singleton Pattern

```
1  public class LazySingleton
2  {
3      private LazySingleton()
4      {
5      }
6
7      public static LazySingleton Instance
8      {
9          get { return Nested.instance; }
10     }
11
12     private class Nested
13     {
14         static Nested()
15         {
16         }
17
18         internal static readonly LazySingleton instance = new LazySingleton();
19     }
20 }
```

Collaboration

Singleton Pattern

- Classes that need to interact directly with a Singleton must refer to its Instance property (or method)
- Alternately, classes can depend on an interface or parameter of the Singleton's type

Consequences

Singleton Pattern

- The default implementation of the Singleton pattern is not thread-safe and should not be used in multi-threaded environments, including web servers (e.g. ASP.NET)
- Singletons introduce tight coupling among collaborating classes
- Singletons are notoriously difficult to test
  - Commonly regarded as an *anti-pattern*
- Using an *IOC Container* it is straightforward to avoid the coupling and testability issues

Single Responsibility

Singleton Pattern

- Management of object lifetime is a *separate responsibility*
- Adding this responsibility to a class with other responsibilities violates the *Single Responsibility Principle* (SRP)
- Using an IOC Container, a separate class can be responsible for managing object lifetimes

Learn more about the **Single Responsibility Principle** in the **Principles of Object Oriented Design** course at Pluralsight On Demand

Implementation Example

Singleton Pattern

- **Logging to a common file**
  - Using separate instance classes
  - Using a Singleton for performance reasons
  - Introducing locking
  - Introducing double-check locking
  - Introducing lazy instantiation via statics
- **Execution Modes**
  - Single-threaded
  - Multi-threaded (in Parallel)

Related Patterns

Singleton Pattern

- Abstract Factory
- Factory Method
- Builder

- See also: Static Classes

References

Singleton Pattern

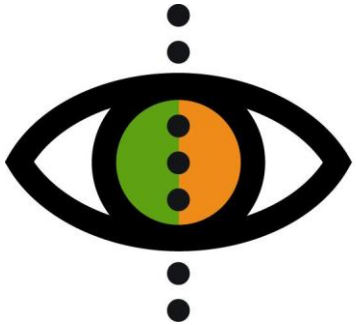
- **Books**
  - Design Patterns, http://amzn.to/95q9ux
  - Design Patterns Explained, http://amzn.to/cr8Vxb
  - Design Patterns in C#, http://amzn.to/bqJgdU
  - Head First Design Patterns, http://amzn.to/aA4RS6
  - C# In Depth, http://amzn.to/djDzZw
- **Online**
  - Implementing the Singleton Pattern in C# http://csharpindepth.com/Articles/General/Singleton.aspx
- **Related Patterns**
  - Builder
  - Abstract Factory
  - Factory Method (a.k.a. Factory pattern)

Summary

Singleton Pattern

- The Singleton Pattern is one of the simplest design patterns
- It is used to ensure that only exactly one instance of a class exists within a program
- Though simple, it is easy to get wrong, and can result in a more brittle and less testable design
- Object lifetime management is usually better handled by a container with this responsibility

For more in-depth online developer training visit



pluralsight

see what you can learn

on-demand content from authors you trust