CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2013
Doc 13 Singleton Pattern
March 19, 2013

References

Design Patterns: Elements of Resuable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, Addison-Wesley, 1995, pp. 127-134

When is a Singleton not a Singleton, Joshua Fox, January 2001, http://java.sun.com/developer/technicalArticles/Programming/singletons/

http://en.wikipedia.org/wiki/Singleton_pattern

The "Double-Checked Locking is Broken" Declaration, http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html

Use your Singletons wisely, http://www.ibm.com/developerworks/webservices/library/co-single.html

Why Singletons are Evil, http://blogs.msdn.com/scottdensmore/archive/2004/05/25/140827.aspx

Why Singletons Are Controversial, http://code.google.com/p/google-singleton-detector/wiki/WhySingletonsAreControversial

Photographs used with permission from www.istockphoto.com

Singleton

Warning

Simplest pattern

- But has subtlest issues particularly in Java

Most controversial pattern

Intent

Ensure a class only has one instance

Provide global point of access to single instance

Singleton

```
public class Counter {
      private int count = 0;
      private static Counter instance;
      private Counter() { }

      public static Counter instance() {
            if (instance == null)
                  instance = new Counter();
            return instance;
      }

      public int increase()  {return ++count;}
}
```

One instance

Global access

This version does not work correctly all the time. See later slides

Ruby Singleton

```
class Counter
    private_class_method :new
    @@instance = nil

    def Counter.instance
        @@instance = new unless @@instance
        @@instance
    end

    def increase
        @count = 0 unless @count
        @count = @count + 1
        @count
    end
end
```

```
require 'singleton'

class Counter
   include Singleton

    def increase
        @count = 0 unless @count
        @count = @count + 1
        @count
    end
end
```

Some Uses

Java Security Manager

Logging a Server

Null Object

Globals are Evil

Why Singletons Are Controversial(Evil)

Singletons provide global access point for some service

Hidden dependencies

Is there a different design that does not need singletons

Pass a reference

Why Singletons Are Controversial(Evil)

Singletons allow you to limit creation of objects of a class

Should that be the responsibility of the class?

Class should do one thing

Use factory or builder to limit the creation

Why Singletons Are Controversial(Evil)

Singletons tightly couple you to the exact type of the singleton object

No polymorphism

Hard to subclass

Why Singletons Are Controversial(Evil)

Singletons carry state with them that last as long as the program lasts

Persistent state makes testing hard and error prone

Why Singletons Are Controversial(Evil)

A Singleton today is a multiple tomorrow

SIngleton pattern makes it hard to change to allow multiple objects

Why Singletons Are Controversial(Evil)

In Java Singletons are a lie

More on this later

Singleton Implementation

Why Not Use This?

```
public class Counter {
        private static int count = 0;

        public static int increase()  {return ++count;}
}
```

Why Not Use This?

```
public class Counter {
        private int count = 0;
        private Counter() { }

        public static Counter instance = new Counter();

        public int increase()  {return ++count;}
}
```


Very subtle the error here

Two Useful Features

Lazy

- Only created when needed

Thread safe

Recommended Implementation

```
public class Counter {
      private int count = 0;
      private Counter() { }

      private static class SingletonHolder {
     private final static Counter INSTANCE = new Counter();
    }
      public static Counter instance() {
                return SingletonHolder.INSTANCE;
      }

      public int increase()  {return ++count;}
}
```


Implementation due to Bill Pugh, found in Wikipedia, http://en.wikipedia.org/wiki/Singleton_pattern. It is thread-safe, lazy and works with all know versions of Java

Correct but not Lazy

```
public class Counter {
        private int count = 0;
        protected Counter() { }

        private final static Counter INSTANCE = new
Counter();

        public static Counter instance() {
                return INSTANCE;
        }

        public int increase()  {return ++count;}
}
```

Lazy, Thread safe with Overhead

```
public class Counter {
        private int count = 0;
        private static Counter instance;
        private Counter() { }

        public static synchronized Counter instance() {
                if (instance == null)
                        instance = new Counter();
                return instance();
        }

        public int increase()  {return ++count;}
}
```

Double-Checked Locking does not work

```
public class Counter {
        private int count = 0;
        private static Counter instance;
        private Counter() { }

        public static Counter instance() {
              if (instance == null)
            synchronize(this) {
                  if (instance == null)
                              instance = new Counter();
               }
                  return instance();
             }

        public int increase()  {return ++count;}
    }
```


See http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html

Java Templates & Singleton

Does not compile

```
public class TemplateSingleton<Type> {
        Type foo;

        public static TemplateSingleton<Type> instance =
            new TemplateSingleton<Type>();
}
```

When is a Single Singleton a Singleton?

When Java Garbage Collects Classes

Singleton class can be garbage collected
Singleton loses any value it had

Solution

Turn off garbage collection of classes (-Xnoclassgc)

Make sure there is always a reference to the class/instance

When Multiple Java Class Loaders are Used

When loaded by two different class loaders there will be two versions of the class

Some servlet engines use different class loader for each servlet

Using custom class loaders can cause this

Purposely Reloading a Java Class

Servlet engines can force a class to be reloaded

Serialize and Deserialize Singleton Object

Serialize the singleton
Deserialize the singleton
You now have two copies

One way to serialize a Java object is using ObjectOutputStream

Ruby Marshal.dump() will not marshal a singleton