

# **CS 635 Advanced Object-Oriented Design & Programming Spring Semester, 2001**

## **Doc 11 Memento, Command, Command Processor Contents**

Singleton.....	2
Intent.....	2
Motivation.....	2
Applicability .....	2
Implementation .....	4
Java.....	4
C++ .....	6
Smalltalk.....	7
Singletons and Static.....	10
Consequences .....	11

## **References**

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

Pattern-Oriented Software Architecture: A System of Patterns (POSA  
1), Buschman, Meunier, Rohnert, Sommerlad, Stal, 1996,

The Design Patterns Smalltalk Companion, Alpert, Brown, Woolf,  
Addision-Wesley, 1998,pp. 91-101

Copyright ©, All rights reserved. 2001 SDSU & Roger Whitney, 5500 Campanile Drive, San  
Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/opl.shtml>) license  
defines the copyright on this document.

## **Singleton Intent**

Insure a class only has one instance, and provide a global point of access to it

## **Motivation**

There are times when a class can only have one instance

## **Applicability**

Use the Singleton pattern when

- There must be only one instance of a class, and it must be accessible to clients from a well-known access point
- When the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code

## **Examples of Using a Singleton**

### **Java Security manager**

All parts of a program must access the same security manager

Once set a security manager cannot be changed in a program

### **Logging the activity of a server**

All parts of the server should use the same instance of the logging system

The server should not be able to change the instance of the logging system was it has been set

### **Null Object**

Null object does not have state, so only need one instance

A binary tree with  $N$  non-null nodes will contain  $N+1$  NullNodes

### **Command Processor**

If Command Processor is to control all commands:

- Can have only one Command Processor
- Entire program needs access to Command Processor

## Implementation Java

// Only one object of this class can be created

class Singleton

```
{  
    private static Singleton _instance = null;  
  
    private Singleton()    { fill in the blank }
```

```
    public static Singleton getInstance()  
    {
```

```
        if ( _instance == null )  
            _instance = new Singleton();  
        return _instance;  
    }
```

```
    public void otherOperations() { blank; }  
}
```

class Program

```
{  
    public void aMethod()  
    {  
        X = Singleton.getInstance();  
    }  
}
```

## **Java Singletons, Classes, Garbage Collection**

Classes can be garbage collected in Java

Only happens when there are

- No references to instances of the class
- No references to the class

If a singleton's state is modified and its class is garbage collected, its modified state is lost

To avoid having singletons garbage collected:

- Disable class garbage collection with -Xnoclassgc flag
- Insure singleton or class always has a reference

Store singleton or class in system property

## Implementation C++

// Only one object of this class can be created

class Singleton

{

private:

static Singleton\* \_instance;

void otherOperations();

protected:

Singleton();

public:

static Singleton\* getInstance();

}

Singleton\* Singleton::\_instance = 0;

Singleton\* Singleton::getInstance()

{

if ( \_instance == 0 )

\_instance = new Singleton;

return \_instance;

}

## Implementation Smalltalk

```
Object subclass: #Singleton
  instanceVariableNames: ''
  classVariableNames: 'UniqueInstance '
  poolDictionaries: ''
  category: 'Whitney-Examples'
```

```
current
  UniqueInstance ifNil: [UniqueInstance := Singleton basicNew].
  ^UniqueInstance
```

```
new
  self error: 'Use current to get an instance of Class ' , self name
```

## Overriding new in Smalltalk

Since can control what new returns one might be tempted to use:

new

```
UniqueInstance ifNil: [UniqueInstance := Singleton basicNew].
```

```
^UniqueInstance
```

This can be misleading, user might think they are getting different objects when calling new

Do we have two different windows below or not?

```
| left right |
```

```
left := SingleWindow new.
```

```
Right := SingleWindow new.
```

```
left position: 100@ 100.
```

```
right position: 500@100.
```



## Naming the Access Method

GOF uses: **instance()**

POSA 1 uses: **getInstance()**

Smalltalk uses **default** and **current**

Selecting names is one of the more difficult problems in object-oriented analysis. No name is perfect<sup>1</sup>

---

<sup>1</sup> Fowler pp. 9, Alpert pp. 98

## **Singletons and Static**

If one needs only one instance of a class why not just implement all methods as static?

- Classes do not inherit Object's protocol
- Hard to modify design if need more than one instance
- Builds bad habits in beginners

## **Consequences**

- Controlled access to sole instance
- Reduced name space
- Permits subclassing
- Permits a variable number of instances
- More flexible than class operations
- Leads to improper use of globals