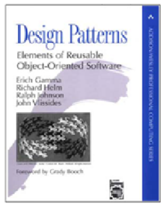


Design Pattern Definitions from the GoF Book

The Singleton Pattern

Ensures a class has only one instance, and provides a global point of access to it.

Creational Patterns

- **The Factory Method Pattern**
  Defines an interface for creating an object, but lets subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- **The Abstract Factory Pattern**
  Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
- **The Singleton Pattern**
- **The Builder Pattern**
- **The Prototype Pattern**

Structural Patterns

- **The Decorator Pattern**
  Attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.
- **The Adapter Pattern**
- **The Facade Pattern**
- **The Composite Pattern**
- **The Proxy Pattern**
- **The Bridge Pattern**
- **The Flyweight Pattern**

Behavioral Patterns

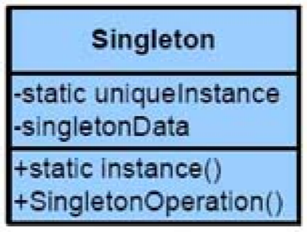
- **The Strategy Pattern**
  Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
- **The Observer Pattern**
  Defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.
- **The Command Pattern**
- **The Template Method Pattern**
- **The Iterator Pattern**
- **The State Pattern**
- **The Chain of Responsibility Pattern**
- **The Interpreter Pattern**
- **The Mediator Pattern**
- **The Memento Pattern**
- **The Visitor Pattern**

Design Patterns: The Singleton

Quick Overview

Ensures a class has only one instance, and provides a global point of access to it.

Defines the class as a Singleton and ensures only one instance can be created. $\longrightarrow$



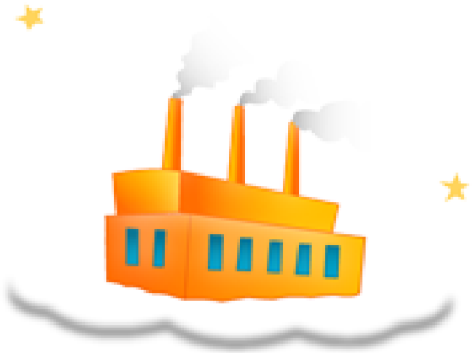
Design Patterns: The Singleton

1 and only 1

Design Patterns: The Singleton

Examples

- A factory class which instantiates instances of other classes, all of which must follow a standard.



- A print spooler which must have one and only one instance sending print jobs to a printer.

- A window manager for a GUI application.

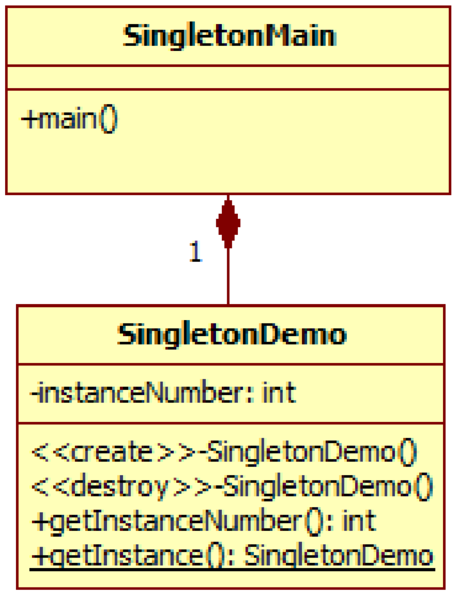
Design Patterns: The Singleton

```
//==================================
// SingletonDemo.h
//==================================
#ifndef SINGLETONDEMO_H
#define SINGLETONDEMO_H
class SingletonDemo
{
        private:
                int instanceNumber;
                SingletonDemo();
        public:
                ~SingletonDemo();
                int getInstanceNumber();
                static SingletonDemo *getInstance();
};
#endif
```

```
//================================
// SingletonDemo.cpp
//================================
#include “SingletonDemo.h”
SingletonDemo::SingletonDemo()  {}
SingletonDemo::~SingletonDemo()  {}
int SingletonDemo::getInstanceNumber()
{
   return this->instanceNumber;
}
//================================
// Return the singleton instance
//================================
SingletonDemo *SingletonDemo::getInstance()
{
   static SingletonDemo *theInstance = NULL;
   static int counter = 1;
   if(theInstance == NULL)
   {
      theInstance = new SingletonDemo();
      theInstance->instanceNumber = counter;
      counter++;
   }
   return theInstance;
}
```

Design Patterns: Singleton

Code Sample



SingletonMain
+main()
1
SingletonDemo
-instanceNumber: int
<<create>>-SingletonDemo()
<<destroy>>-SingletonDemo()
+getInstanceNumber(): int
+getInstance(): SingletonDemo

UML diagram drawn with StarUML

SingletonMain

Creates two pointers to SingletonDemo
Calls getInstance() for each pointer
Calls getInstanceNumber to show both point to the same instance.

Let's look at the code and run the demonstration.