SINGLETON DESIGN PATTERN

Singleton patterns are used, when you want to allow creation of only one instance(object) of a particular class. Such classes (which allow only single object to be created) are called **Singleton Classes**. For example, there is only one Window manager, so a class implementing Windows Manager, should be allowed to be instantiated only once.

One way to allow only one object creation is to have everything in the class (data and methods) static and global. Because static members are available at class level and have only a single copy. But this is not an Object Oriented Design Methodology plus this will not disallow creation of objects, we can create as many projects of a class, and the responsibility to disallow multiple objects should lie with the class and not with the user. User should be free to use the class, if an attempt is made to create multiple objects of the class, then it should result in a compile-time error.

Singleton Pattern is part of the *Creational Design Pattern* because it effects the way in which an object of a class is getting created.

Implementation of Singleton Design Pattern:

```
1   class Singleton
2   {
3     public:
4
5       static aSingletonClass *getInstance()
6       {
7           //The object is created when it is called first
8           if(!_instance)
9             _instance = new Singleton;
10          return _instance;
11      }
12
13      static void deleteInstance()
14      {
15          if(_instance)
16          {
17            delete _instance;
18            _instance = NULL; //To avoid memory leak
19          }
20      }
21
```

```
22      private:
23          static Singleton *instance_;
24
25          Singleton();  // Private Constructor
26          ~Singleton() {}; // Private Destructor, noone can delete the object explicitly
27
28         // Preventing creation of object by Copy or by assignment.
29          Singleton(const Singleton&);
30          Singleton& operator=(const Singleton&);
31      };
```

Note that the Constructor (default & Copy constructor) and overloaded assignment operator are only declared and not defined, but the destructor is defined. This is explained below:

If we define the constructor then object of the class can be created in a friend class or friend function. Though we have not yet declared any function or class as friend, but if we do , our code should not stopped functioning.

The destructor is defined, because we need to destroy the object (the only object) which is getting created.

Source: http://www.ritambhara.in/singleton-design-pattern/