JAVA - HOW TO USE SINGLETON CLASS ?

The Singleton's purpose is to control object creation, limiting the number of obejcts to one only. Since there is only one Singleton instance, any instance fields of a Singleton will occur only once per class, just like static fields. Singletons often control access to resources such as database connections or sockets.

For example, if you have a license for only one connection for your database or your JDBC driver has trouble with multithreading, the Singleton makes sure that only one connection is made or that only one thread can access the connection at a time.

Implementing Singletons

Example 1

The easiest implementation consists of a private constructor and a field to hold its result, and a static accessor method with a name like getInstance().

The private field can be assigned from within a static initializer block or, more simply, using an initializer. The getInstance( ) method (which must be public) then simply returns this instance −

```
// File Name: Singleton.java
public class Singleton {

   private static Singleton singleton = new Singleton( );

   /* A private Constructor prevents any other
    * class from instantiating.
    */
   private Singleton(){ }

   /* Static 'instance' method */
   public static Singleton getInstance( ) {
      return singleton;
   }
   /* Other methods protected by singleton-ness */
   protected static void demoMethod( ) {
      System.out.println("demoMethod for singleton");
   }
}
```

Here is the main program file where we will create singleton object:

```
// File Name: SingletonDemo.java
public class SingletonDemo {
   public static void main(String[] args) {
      Singleton tmp = Singleton.getInstance( );
      tmp.demoMethod( );
   }
}
```

This would produce the following result −

```
demoMethod for singleton
```

Example 2

Following implementation shows a classic Singleton design pattern:

```
public class ClassicSingleton {

   private static ClassicSingleton instance = null;
   protected ClassicSingleton() {
      // Exists only to defeat instantiation.
```

```
    }
    public static ClassicSingleton getInstance() {
       if(instance == null) {
          instance = new ClassicSingleton();
       }
       return instance;
    }
}
```

The ClassicSingleton class maintains a static reference to the lone singleton instance and returns that reference from the static getInstance() method.

Here ClassicSingleton class employs a technique known as lazy instantiation to create the singleton; as a result, the singleton instance is not created until the getInstance() method is called for the first time. This technique ensures that singleton instances are created only when needed.

java_object_classes.htm