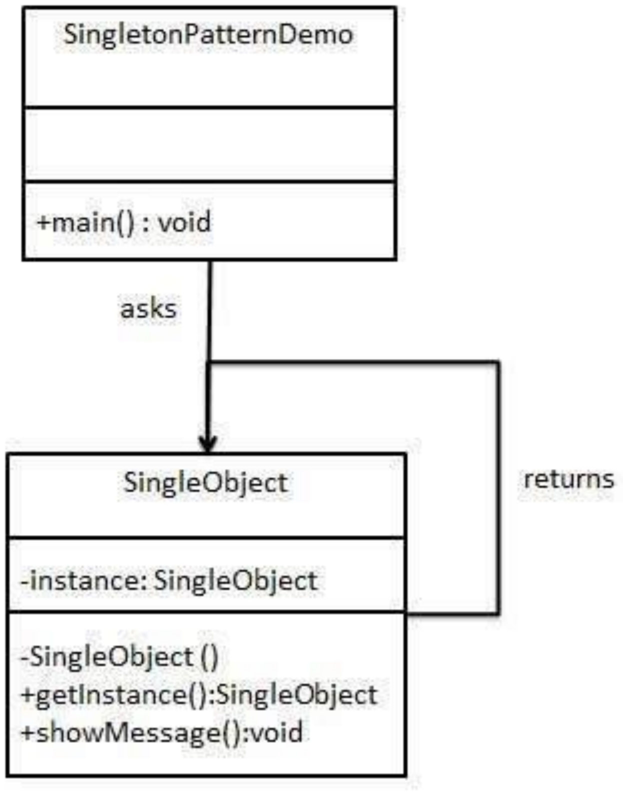
Singleton design pattern in Java

- We can make constructor as private. So that We can not create an object outside of the class.
- This property is useful to create singleton class in java.
- Singleton pattern helps us to keep only one instance of a class at any time.
- The purpose of singleton is to control object creation by keeping private constructor.

We're going to create a *SingleObject* class. *SingleObject* class have its constructor as private and have a static instance of itself.

SingleObject class provides a static method to get its static instance to outside world. *SingletonPatternDemo*, our demo class will use *SingleObject* class to get a *SingleObject* object.



Step 1

Create a Singleton Class.

SingleObject.java

```
public class SingleObject {

   //create an object of SingleObject
   private static SingleObject instance = new SingleObject();

   //make the constructor private so that this class cannot be
   //instantiated
   private SingleObject(){}

   //Get the only object available
   public static SingleObject getInstance(){
      return instance;
   }
   public void showMessage(){
      System.out.println("Hello World!");
   }
}
```

Step 2

Get the only object from the singleton class.

SingletonPatternDemo.java

```
public class SingletonPatternDemo {
   public static void main(String[] args) {

      //illegal construct
      //Compile Time Error: The constructor SingleObject() is not visible
      //SingleObject object = new SingleObject();

      //Get the only object available
      SingleObject object = SingleObject.getInstance();
      //show the message
      object.showMessage();
   }
}
```

Output

```
Hello World!
```

Following implementation shows a classic Singleton design pattern –

```
public class ClassicSingleton {

   private static ClassicSingleton instance = null;
   private 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.