Patrón Singleton en Java, con ejemplos

El proyecto de este post los puedes descargar pulsando [AQUI](https://github.com/jarroba/Singleton_Java).

En este tutorial vamos a hablar del patrón de diseño "***Singleton***", que en ingeniería del software es un patrón diseñado para limitar la creación de objetos pertenecientes a una clase. El objetivo de este patrón es el de garantizar que una clase solo tenga una instancia (o ejemplar) y proporcionar un punto de acceso global a ella. Esta patrón; por ejemplo, suele ser utilizado para las conexiones a bases de datos.

Este patrón se implementa haciendo privado el constructor de la clase y creando (en la propia clase) un método que crea una instancia del objeto si este no existe. Veamos a continuación un ejemplo de este patrón en el que vamos a crear una clase llamada "***SoyUnico***" con un constructor que va a ser privado y un método llamado '*getSingletonInstance()*' que será el encargado de crear una instancia de esta clase si no se ha creado todavía:

public class SoyUnico {

private String nombre;

private static SoyUnico soyUnico;

*// El constructor es privado, no permite que se genere un constructor por defecto.*

private SoyUnico(String nombre) {

this.nombre = nombre;

System.out.println("Mi nombre es: " + this.nombre);

}

public static SoyUnico getSingletonInstance(String nombre) {

if (soyUnico == null){

soyUnico = new SoyUnico(nombre);

}

else{

System.out.println("No se puede crear el objeto "+ nombre + " porque ya existe un objeto de la clase SoyUnico");

}

return soyUnico;

}

*// metodos getter y setter*

}

Vemos como al llamar al método '*getSingletonInstance()*'  pregunta si el atributo '*soyUnico*' es null o no, y si no lo es lo crea. En el caso de que ya haya sido creado no imprimirá un mensaje por pantalla diciendonos que ya existe un objeto de esa clase.

Vamos a ver como efectivamente nos crea un solo objeto, ejecutando el siguiente código en el que llamamos dos veces el método '*getSingletonInstance()*':

public class Main {

public static void main(String[] args) {

SoyUnico ricardo = SoyUnico.getSingletonInstance("Ricardo Moya");

SoyUnico ramon = SoyUnico.getSingletonInstance("Ramón Invarato");

*// ricardo y ramon son referencias a un único objeto de la clase SoyUnico*

System.out.println(ramon.getNombre());

System.out.println(ricardo.getNombre());

}

}

Como salida a este código tenemos lo siguiente:

Mi nombre es: Ricardo Moya

No se puede crear el objeto Ramón Invarato porque ya existe un objeto de la clase SoyUnico

Ricardo Moya

Ricardo Moya

Vemos como efectivamente cuando no hay un objeto de la clase "***SoyUnico***", lo crea y si lo quiere volver a crear, el método nos dice que ya hay otro creado pero le asigna la referencia al objeto, es decir que las variable 'ricardo' y 'ramon' referencian al mismo y único objeto de la clase "***SoyUnico***"; tal y como podemos ver al preguntar por el atributo '*nombre*' de la clase.

Por último este patrón no quedaria perfectamente implementado si no controlamos la copia (o clone) de los objetos de esta clase; por ello es importante sobrescribir el método '*clone()*', para que cuando se quiera clonar este objeto se lance una [excepción](https://jarroba.com/excepciones-exception-en-java-con-ejemplos/). Este método lo sobrescribimos de la siguiente forma:

*//Sobreescribimos el método clone, para que no se pueda clonar un objeto de esta clase*

@Override

public SoyUnico clone(){

try {

throw new CloneNotSupportedException();

} catch (CloneNotSupportedException ex) {

System.out.println("No se puede clonar un objeto de la clase SoyUnico");

}

return null;

}

Veamos que pasa si intentamos clonar el objeto de la clase "***SoyUnico***":

public class Main {

public static void main(String[] args) {

SoyUnico ricardo = SoyUnico.getSingletonInstance("Ricardo Moya");

try{

SoyUnico richard = ricardo.clone();

System.out.println(richard.getNombre());

}catch (NullPointerException ex){

ex.printStackTrace();

}

}

}

Vemos que nos lanza una excepción y no nos permite clonar ese objeto.

Mi nombre es: Ricardo Moya

No se puede clonar un objeto de la clase SoyUnico

java.lang.NullPointerException

at Main.Main.main(Main.java:21)

En resumen, en este tutorial se ha puesto un ejemplo muy sencillo para ver el funcionamiento del patrón "***Singleton***" que es un patron de diseño muy utilizado (quizas el que más), aplicable en los casos en los que solo queremos tener un único objeto de la clase que implementamos.