PROGRAMA PATRÓN SINGLETON

ASIGNATURA:

Programación avanzada orientada a objetos

CATEDRÁTICO:

José Juan Hernández Mora

ALUMNO:

José Montoya Guzmán

FECHA DE ASIGNACIÓN:

7 de noviembre de 2016

FECHA DE ENTREGA:

28 de noviembre de 2016

Índice

1. DESCRIPCIÓN DEL PROBLEMA

Planteamiento del problema

Desarrollar un programa que represente el patrón de Singleton en java.

Objetivo general

Implementar un programa que limite la creación de objetos de una clase a un solo objeto, tal como trabaja el patrón de Singleton.

Objetivos específicos

- El programa solo deberá permitir una única instanciación de la clase.
- No se puede generar un clon del objeto que ya se encuentre creado, ya que este es único.

2. DEFINICIÓN DE LA SOLUCIÓN

Análisis del problema

Primero se analizó cuáles son los fundamentos teóricos de patrón de Singleton. Estos se mencionan a continuación.

En ingeniería de software, el patrón Singleton (instancia única) es un patrón de diseño utilizado para restringir la creación de objetos pertenecientes a una clase. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

El patrón Singleton se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor con modificadores de acceso como protegido o privado.

La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, sólo uno de ellos debe lograr crear el objeto. La solución clásica para este problema es utilizar exclusión mutua en el método de creación de la clase que implementa el patrón.

El patrón Singleton provee una única instancia global gracias a que:

- La propia clase es responsable de crear la única instancia.
- Permite el acceso global a dicha instancia mediante un método de clase.
- Declara el constructor de clase como privado para que no sea instanciable directamente.

Resultados esperados

Por lo tanto, el objetivo deseado es que el programa tenga generadas estas condiciones, cabe mencionar que para fines de este programa no se realiza la consideración de hilos y únicamente se asegurará de cumplir los requerimientos del patrón Singleton.

De lo anterior de definen las siguientes especificaciones.

Historias de usuario

ID	Descripción	Valor
HU1	Clase principal encargada de la ejecución y flujo.	3
HU2	Clase Singleton	3
HU3	Privatización de constructor de Singleton	3
HU4	No crear más de un objeto ni clonar a raíz de uno creado	3

3. DISEÑO

Pseudocódigo

INICIO

Creación de objeto único

Llamar a método que crea objeto de la clase Singleton

Si no hay ninguna instancia

Crear objeto único

Si ya hay una instancia

No crear objeto e indicar error

Además si se intenta crear un objeto a raíz del primero

Llamar al método clone

Si ya existe un objeto

No se puede crear un objeto clon

Si no existe

Enviar mensaje de que no hay objeto del cuál clonar uno nuevo

FIN

Diagrama de flujo (ACTIVIDADES)

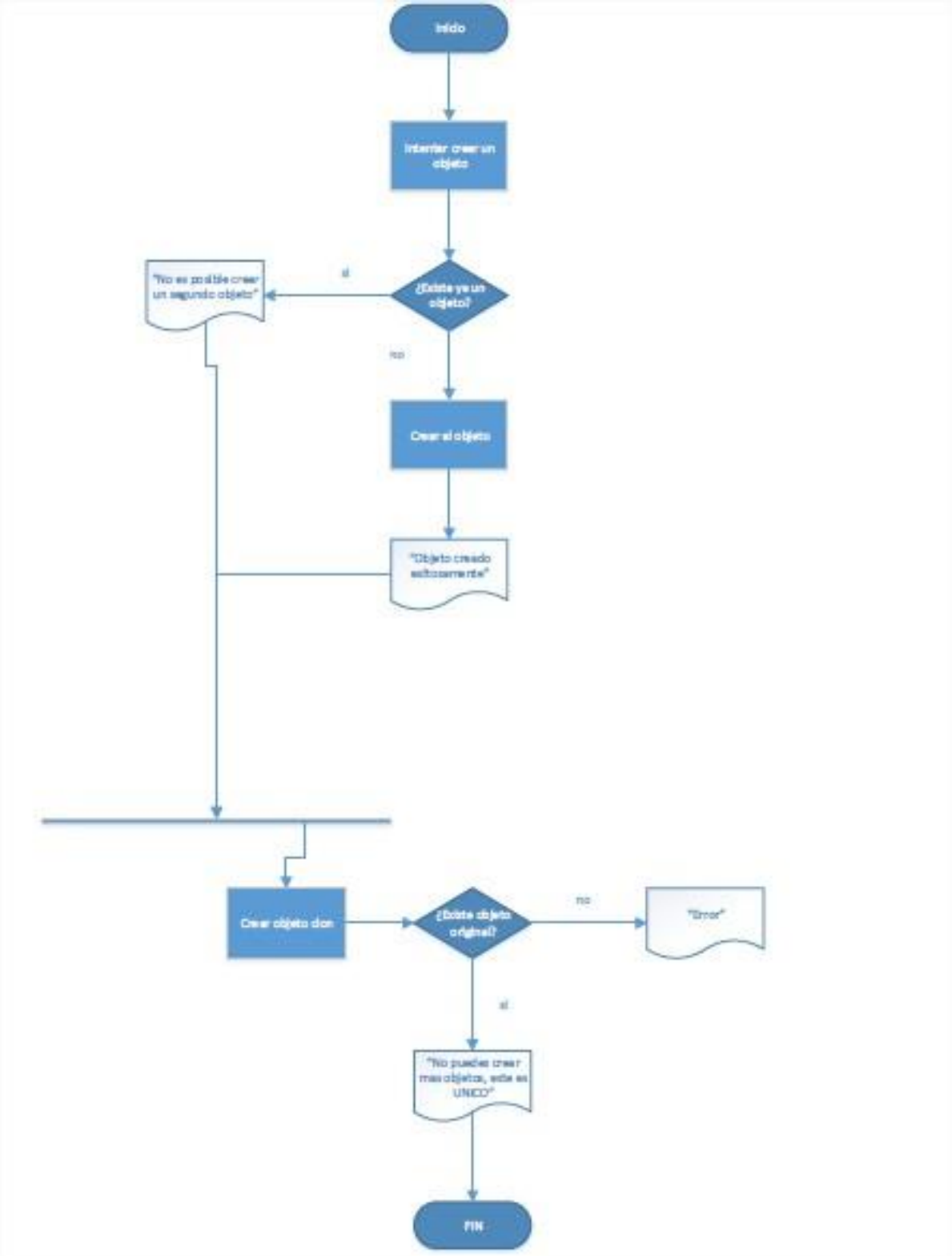
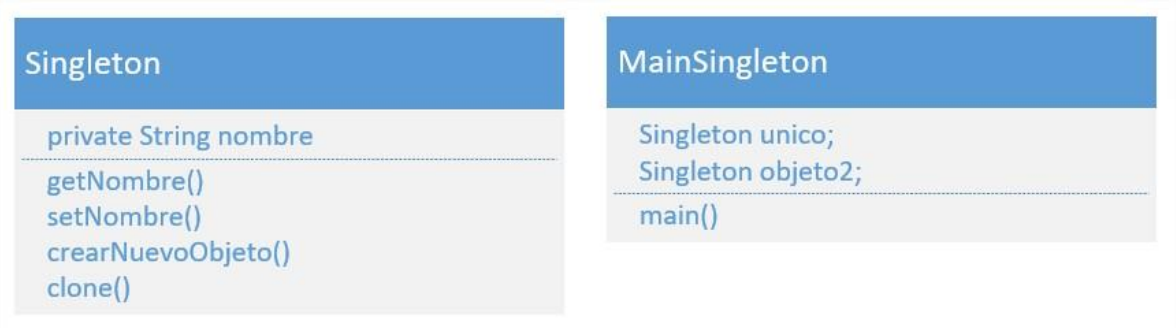


Diagrama de clases



4. DESARROLLO

Codificación

Clase Singleton.java

```
1 public class Singleton{
 2
 3    private String nombre;
 4
 5    private static Singleton objetoUnico;
 6
 7    private Singleton(String nombre) {
 8       this.nombre = nombre;
 9       System.out.println("Mi nombre es: " + this.nombre);
10    }
11
12    public static Singleton crearNuevoObjeto(String nombre) {
13       if (objetoUnico == null){
14          objetoUnico = new Singleton(nombre);
15       }
16       else{
17          System.out.println("No se puede crear el objeto "+ nombre + ", solo puede haber un
objeto unico y ya existe");
18       }
19          return objetoUnico;
20    }
21
22    //Sobreescribimos el mÃ©todo clone, para que no se pueda clonar un objeto de esta clase
23    @Override
24    public Singleton clone(){
25       try {
26          throw new CloneNotSupportedException();
27       } catch (CloneNotSupportedException ex) {
28          System.out.println("No se puede clonar un objeto de clase Singleton");
```

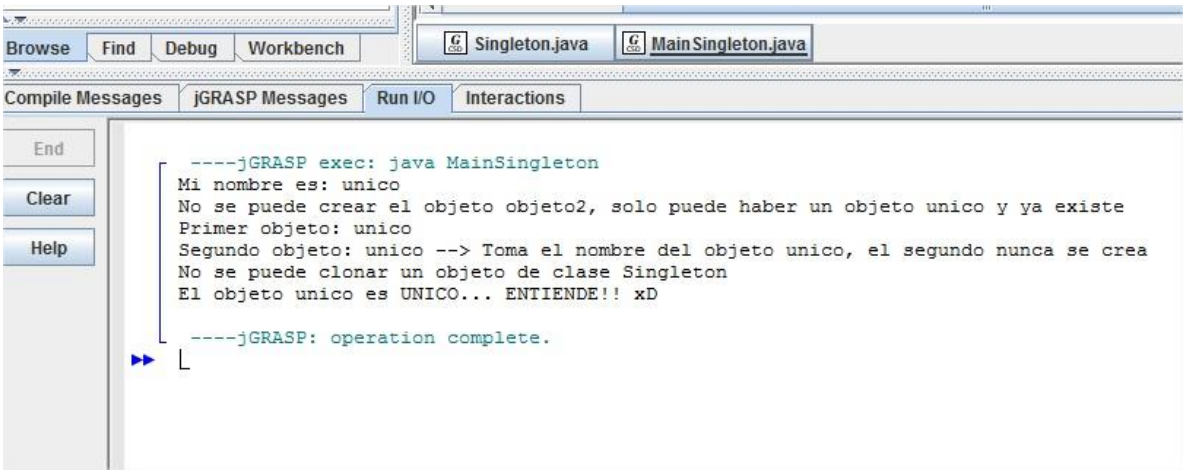
```
29        }
30        return null;
31    }
32
33    public String getNombre() {
34        return nombre;
35    }
36
37    public void setNombre(String nombre) {
38        this.nombre = nombre;
39    }
40
41 }
```

Clase MainSingleton.java

```
1 public class MainSingleton{
 2
 3   public static void main(String[] args){
 4
 5     Singleton unico = Singleton.crearNuevoObjeto("unico");
 6     Singleton objeto2 = Singleton.crearNuevoObjeto("objeto2");
 7
 8     System.out.println("Primer objeto: "+unico.getNombre());
 9     System.out.println("Segundo objeto: "+objeto2.getNombre()+" --> Toma el nombre del
objeto unico, el segundo nunca se crea");
10
11      try{
12        Singleton clon = unico.clone();
13        System.out.println("El objeto "+unico.getNombre() +" es UNICO... ENTIENDE!! xD");
14      }catch (NullPointerException ex){
15        System.out.println(ex);
16      }
17    }
18
19 }
```

Implementación

La ejecución del programa, es presentada a continuación.



Las instrucciones que va mostrando el programa son las siguientes:

1. Se muestra el nombre del objeto creado, que es “unico”.
2. Se crea un segundo objeto, pero como el método de la clase Singleton determina que ya existe uno, se indica que no es posible crearlo.

3. Se muestra el nombre del primer (y único) objeto.
4. El segundo objeto recibe el nombre del primero a pesar de intentar ser creado con un nombre propio. Debido a que solo hay un unico objeto, el método get() devuelve el nombre de este (el primero).
5. Al intentar crear un nuevo objeto a raíz del primero se indica que no es posible debido a la restricción impuesta en el método clone().
6. Finalmente se indica nuevamente que no se puede crear un segundo objeto.

NOTA: El mensaje del punto 5 corresponde a la respuesta que da el método clone() de la clase Singleton, mientras que el mensaje del punto 6 pertenece a la secuencia de instrucciones del main() que se generar posterior a la llamada al método.

5. DEPURACIÓN

La etapa de pruebas fue relativamente muy sencilla ya que solo se agregó una restricción debido a que inicialmente no se consideró que sucedía si se intentaba crear más objetos a raíz del primero. Al darse cuenta se implementó el cambio y se realizaron los cambios pertinentes en la documentación.

Bugs

Nombre	Descripción	Tiempo de solución
Clonación de objetos	Si se intenta crear un objeto a raíz del primero si lo permite, para ello se implementó el método clone()	15 minutos

6. DOCUMENTACIÓN

Manual de usuario

1. La clase Singleton es a partir de la cual se crea un objeto único.
2. La clase Singleton contiene:
   Un atributo privado para identificar el objeto.
   Sus métodos set y get correspondientes.

Un constructor privado que evita la creación de objetos desde otra clase.
Un método para la creación de objetos.
Un método sobreescrito para evitar clonación de objetos.

3. La clase MainSingleton es la que debe ejecutarse para iniciar el programa.
4. La clase MainSingleton contiene:
   La creación de los dos objetos a través de la llamada al método crearObjetoNuevo() de la clase Singleton.
   La llamada al método que clona objetos.