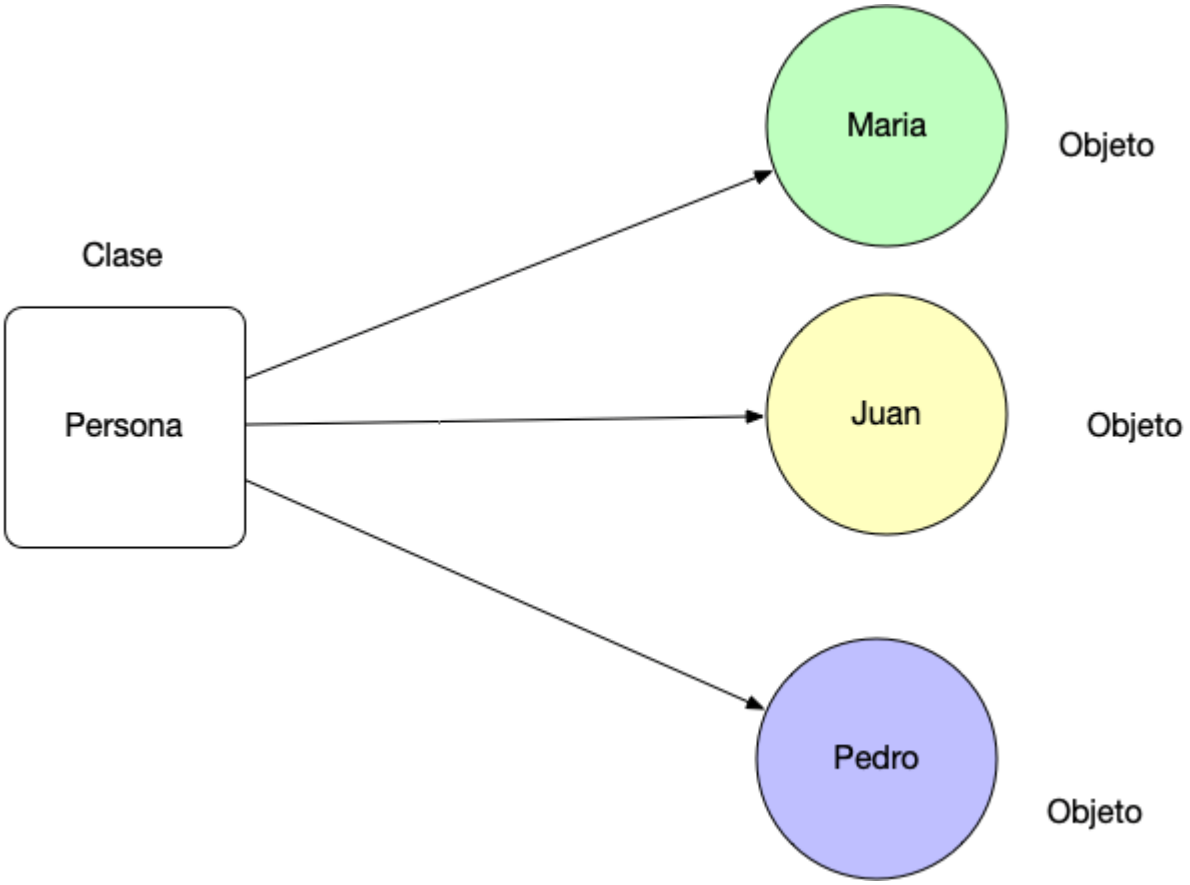
Tabla de Contenidos

- Singleton y Limitaciones
  - Java Singleton
  - Singleton y ClassLoaders
- Otros artículos relacionados

Java Singleton el patrón de diseño más básico . Como ya he comentado en varias ocasiones el conocimiento de los patrones de diseño es algo clave a la hora de abordar desarrollos y de solucionar problemas, sobre todo problemas complejos y que necesitan flexibilidad .Hoy voy a entrar a detalle en uno de los patrones de diseño mas sencillo ,el patrón Java Singleton . Este patrón de diseño se encarga de que una clase determinada únicamente pueda tener un único objeto. Normalmente una clase puede instanciar todos los objetos que necesite.



Singleton y Limitaciones

Sin embargo una clase que siga el patrón Singleton tiene la peculiaridad de que solo puede instanciar un único objeto .Este tipo de clases son habituales en temas como configurar parámetros generales de la aplicación ya que una vez instanciado el objeto los valores se mantienen y son compartidos por toda la aplicación. Uno de los ejemplos más clásicos son las clases que Spring instancia que todos son Singleton por defecto. Vamos a construir una clase con un Java Singleton. A esta clase la denominaremos configurador.



Java Singleton

Una vez que tenemos claro cual es el concepto de Configurador vamos a crearlo en código .En este caso nuestro configurador almacenará dos valores url, y base de datos que serán compartidos por el resto de Clases de la aplicación , las cuales simplemente la leerán.

```
package com.arquitecturajava;

public class Configurador {

    private String url;
    private String baseDatos;
    private static final Configurador miconfigurador;

    public static Configurador getConfigurador(String url, String
```

```
baseDatos) {

        if (miconfigurador == null) {

            miconfigurador = new Configurador(url, baseDatos);
        }
        return miconfigurador;
    }

    private Configurador(String url, String baseDatos) {

        this.url = url;
        this.baseDatos = baseDatos;

    }

    public String getUrl() {
        return url;
    }

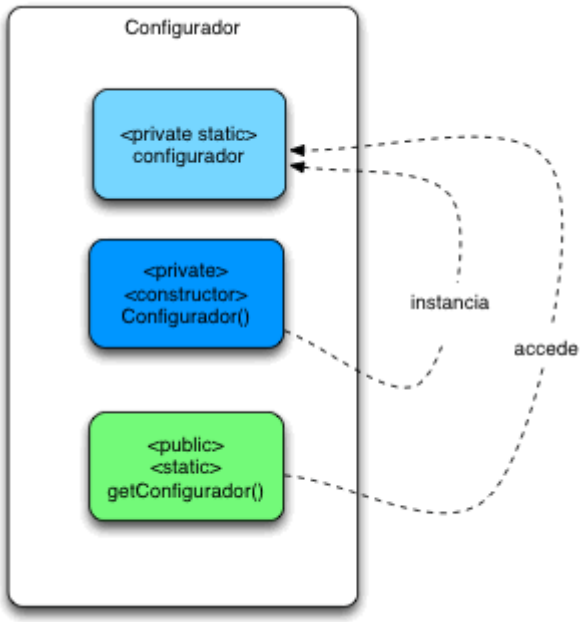
    public void setUrl(String url) {
        this.url = url;
    }

    public String getBaseDatos() {
        return baseDatos;
    }

    public void setBaseDatos(String baseDatos) {
        this.baseDatos = baseDatos;
    }
```

```
}
```

Para conseguir que una clase sea de tipo Singleton necesitamos en primer lugar que su constructor sea privado. De esa forma ningún programa será capaz de construir objetos de esta tipo y por lo tanto no podremos construir ninguno estaremos en cero . En segundo lugar necesitaremos disponer de una variable estática privada que almacene una referencia al objeto que vamos a crear a traves del constructor . Por ultimo un método estático publico que se encarga de instanciar el objeto la primera vez y almacenarlo en la variable estática apoyándose en el constructor privado que recordemos se puede llamar desde la misma clase.



Una vez aclarado como funciona un Singleton es muy sencillo utilizarle desde un programa ya que basta con invocar al método estático.

```
package com.arquitecturajava;

public class Principal {
```

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

        Configurador c = Configurador.getConfigurador("miurl",
"mibaseDatos");

        System.out.println(c.getUrl());

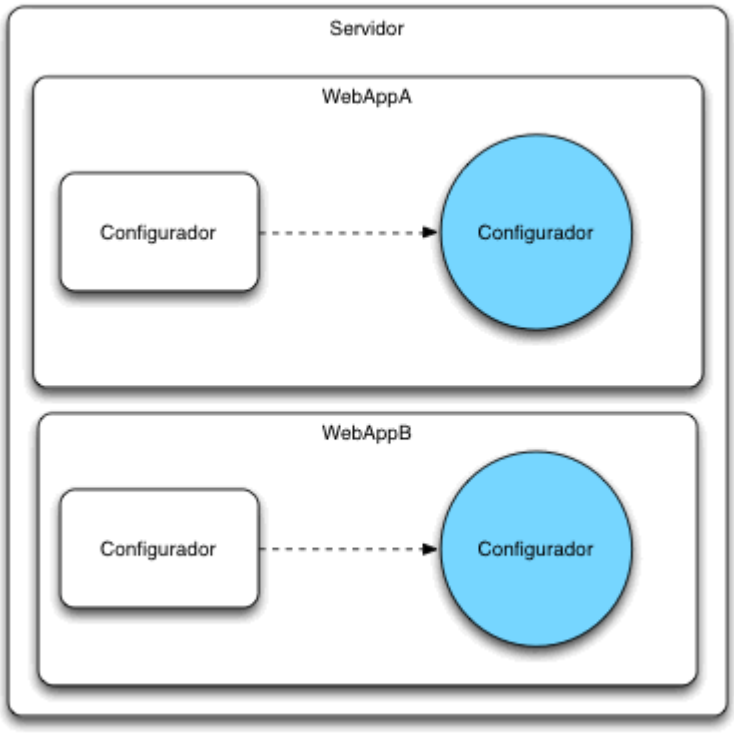
        System.out.println(c.getBaseDatos());

    }

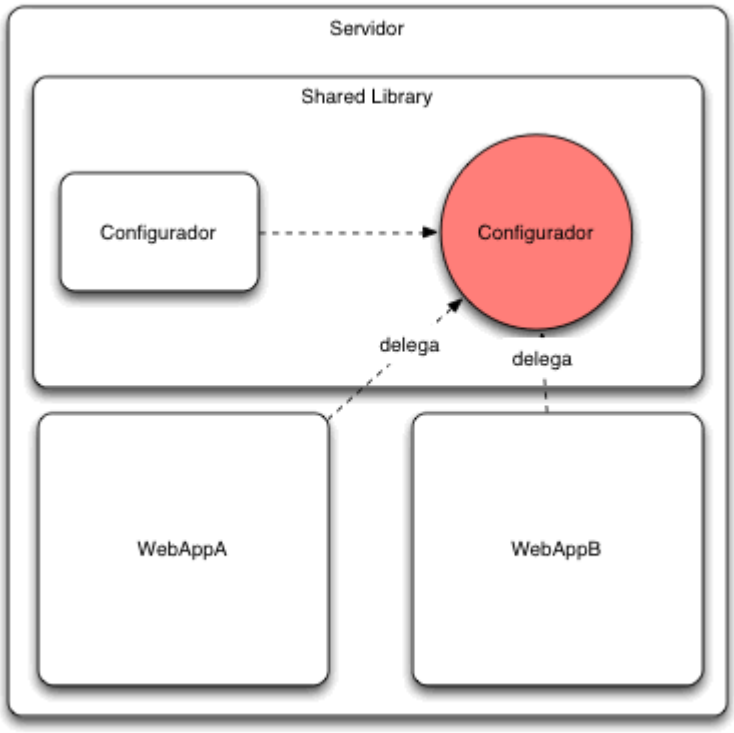
}
```

Singleton y ClassLoaders

A veces los patrones pueden sernos muy útiles y a veces pueden no serlo tanto y llevarnos a situaciones problemáticas . Por ejemplo si preguntamos lo siguiente . ¿El patrón Singleton nos genera un único objeto para una clase Java? . La respuesta mas normal es “SI” sin embargo esta no es verdad del todo. La respuesta correcta es que el patron Singleton nos genera un objeto por cada clase cargada en el mismo ClassLoader.¿ Que quiere decir esto?. Pues quiere decir que por ejemplo dos aplicaciones web que cada una tiene su propio WebClassLoader tendrán cada una so propia instancia.



Esto en principio no es problemático porque se encuentran aisladas . Ahora bien hay situaciones en las que un administrador de sistemas puede decidir compartir librerias y clases entre distintas aplicaciones.



En este caso si podemos tener problemas ya que el objeto Singleton que en principio fue diseñado para configurar una aplicación concreta estará compartido por varias. Mucho ojo sobre como gestionamos este patrón de diseño ya en algunos casos puede generar problemas y situaciones curiosas .

Otros artículos relacionados

- EJB Singleton
- Utilizando Java Singleton Properties
- Spring Singleton vs Prototype
- EntityManager, EntityManagerFactory y Singletons
- El concepto de ClassLoader

CURSO Introducción Patrones Diseño GRATIS

APUNTATE!!