SEMINARIO

3

Ejercicio de diseño

Logger

Diseño del Software

Grado en Ingeniería Informática del Software

Curso 2022-2023

Escuela de Ingeniería Informática
Universidad de Oviedo

César Acebal

¿Cómo garantizar que una clase tenga una única instancia?

Logger

- Se quiere una clase Logger que permita mostrar mensajes de traza en una aplicación

```
logger.log("¡Hola, mundo!");
```

- ¿Cómo podríamos garantizar que de dicha clase sólo exista un único objeto?

Única instancia

- **¿Qué necesitamos?**
  1. Evitar que se pueda llamar a `new` desde fuera de la clase
     - Haciéndolo privado (o protegido)

```
public class Logger {

  private Logger() {
    ...
  }

}
```

Única instancia

- **Pero entonces... ¿quién crea esa única instancia?**
  - Necesariamente tendrá que ser desde dentro de la clase

Única instancia

- **Así pues...**
  2. Crear el único objeto desde dentro de la clase
     - Y guardarlo

```
public class Logger {

   private static final Logger instance = new Logger();

   ...

}
```

Única instancia

- **Ya tenemos el objeto creado y almacenado en una variable estática**
  - (Lógicamente, tiene que ser estática)
- **Pero ahora nos surge un nuevo problema**
  - ¿Cómo acceder a dicha instancia única desde fuera de la clase?

Única instancia

- **Por lo que, finalmente...**
  3. Hay que proporcionar un punto de acceso a dicha única instancia

```
public class Logger {

  public static Logger getInstance() {
    return instance;
  }

}
```

```
public class Logger
{
    private static Logger instance = new Logger();

    public static Logger getInstance()
    {
        return instance;
    }

    private Logger()
    {
    }

    public void log(String message)
    {
        System.out.println(message);
    }
}
```

Patrón Singleton

- **Pues bien, esto que hemos hecho es el patrón Singleton**
  - Ojo con cómo lo usemos
    - Al poder acceder a ellos desde cualquier lugar de la aplicación se comportan como variables globales
    - Que sean *singletons* no quiere decir que tengamos que obtener la única instancia necesariamente a través de su método estático `getInstance`
      - También podemos pasarlos como parámetros a las clases cliente que lo necesiten, como haríamos con cualquier otro objeto

Patrón Singleton

- **Otras consideraciones**
  - Inicialización perezosa

```
public class Logger {

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

Patrón Singleton

- **Otras consideraciones**
  - La misma técnica valdría para un número limitado de objetos

El problema del Singleton y las pruebas

The Clean Code Talks - "Global State and Singletons"

Firefox File Edit View History Bookmarks Tools Window Help

https://docs.google.com/a/google.com/Present?docID=adzqh2m4hq_174gcphdcgh&fs=true&revision=_latest&start=0&theme=blank&cwj=true

Singleton: Good vs Bad

```
class AppSettings {
   private static AppSettings instance =
      new AppSettings();
   Object state1;
   Object state2;
   Object state3;
   e AppSettings() { ... }

   public static AppSettings getInstance(){
     return instance;
   }
   ...
}
```

Test can never access internal state of the singleton.

14:00 / 54:08

https://www.youtube.com/watch?v=-FRm3VPhseI

Dos tipos de logger

Introducimos una primera variante: ahora nos surge la necesidad de tener dos tipos de logger (de consola y fichero), manteniendo la misma restricción de que sólo haya un único objeto.

Dos tipos de logger

- De consola y fichero
- Una vez decidido el tipo de logger y creado éste, ya no se podrá cambiar
- Se mantiene el requisito de que la instancia del logger creado sea única durante toda la ejecución del programa
- Plantead el diseño

(Lo terminaremos el próximo día.)