SEMINARIO

3

Logger

Diseño del Software

Grado en Ingeniería Informática del Software

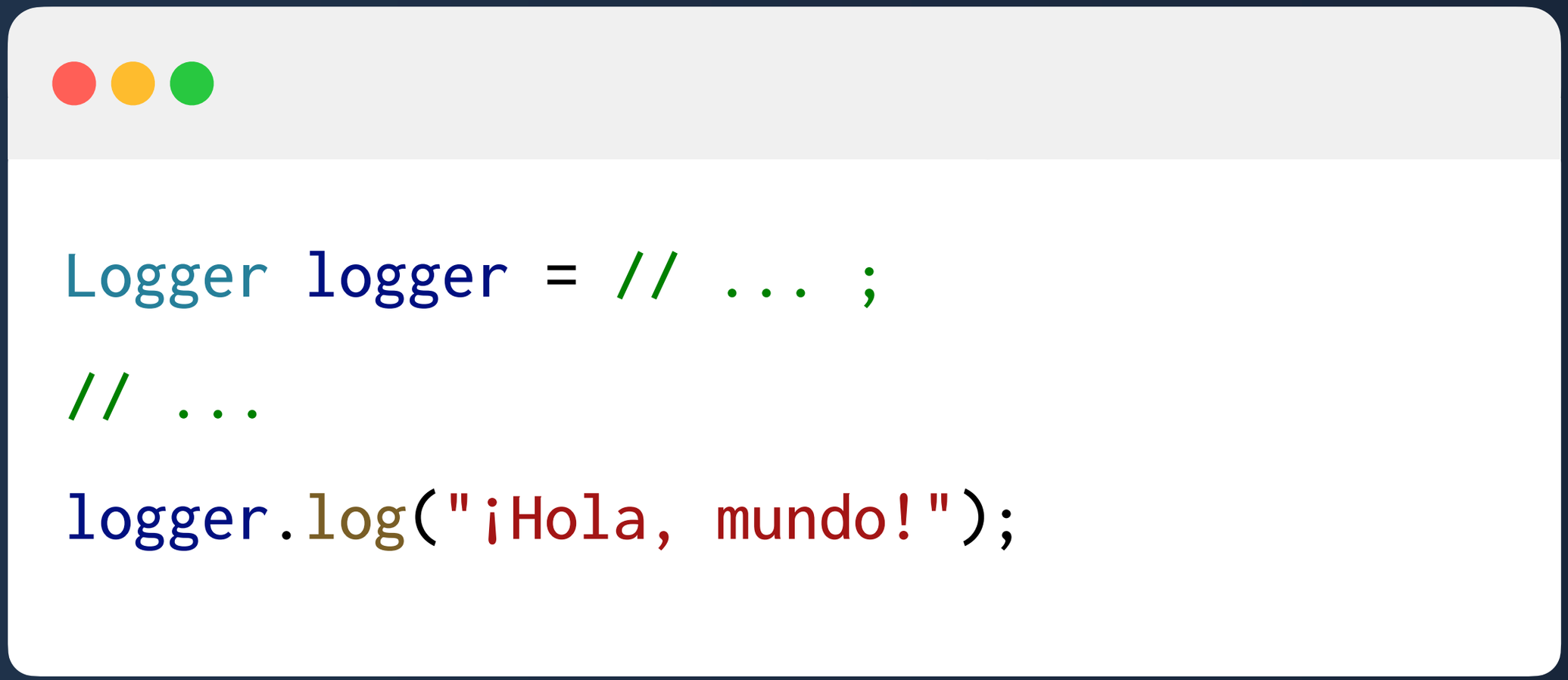
Curso 2024-2025

Escuela de Ingeniería Informática
Universidad de Oviedo

César Acebal

¿Cómo garantizar que una clase tenga una única «instancia»?

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



```
Logger logger = // ... ;

// ...

logger.log("¡Hola, mundo!");
```

```
Logger logger = // ... ;

// ...

logger.log("¡Hola, mundo!");
```

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

¿Qué necesitamos?

1

Para que solo haya un único objeto

Evitar que se pueda llamar a ‘new‘ desde fuera de la clase.

Haciendo al constructor privado (o protegido).

Logger.java

```java
public class Logger {

  private Logger() {
    // ...
  }

  // ...

}
```

Pero entonces…

¿Quién crea ese único objeto?

Necesariamente tendrá que ser desde dentro de la clase.

2

Para que **haya** dicho objeto

Crear el único objeto desde dentro de la propia clase.

Y guardarlo.

Logger.java

```java
public class Logger {

    private static final Logger instance =
        new Logger();

    // ...
}
```

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 dicho único objeto desde fuera de la clase?

3

Para que se pueda acceder a él

Proporcionar un punto de acceso global dicho objeto.

Logger.java

```java
public class Logger {
  // ...
  public static Logger getInstance() {
    return instance;
  }
  // ...
}
```

Logger.java

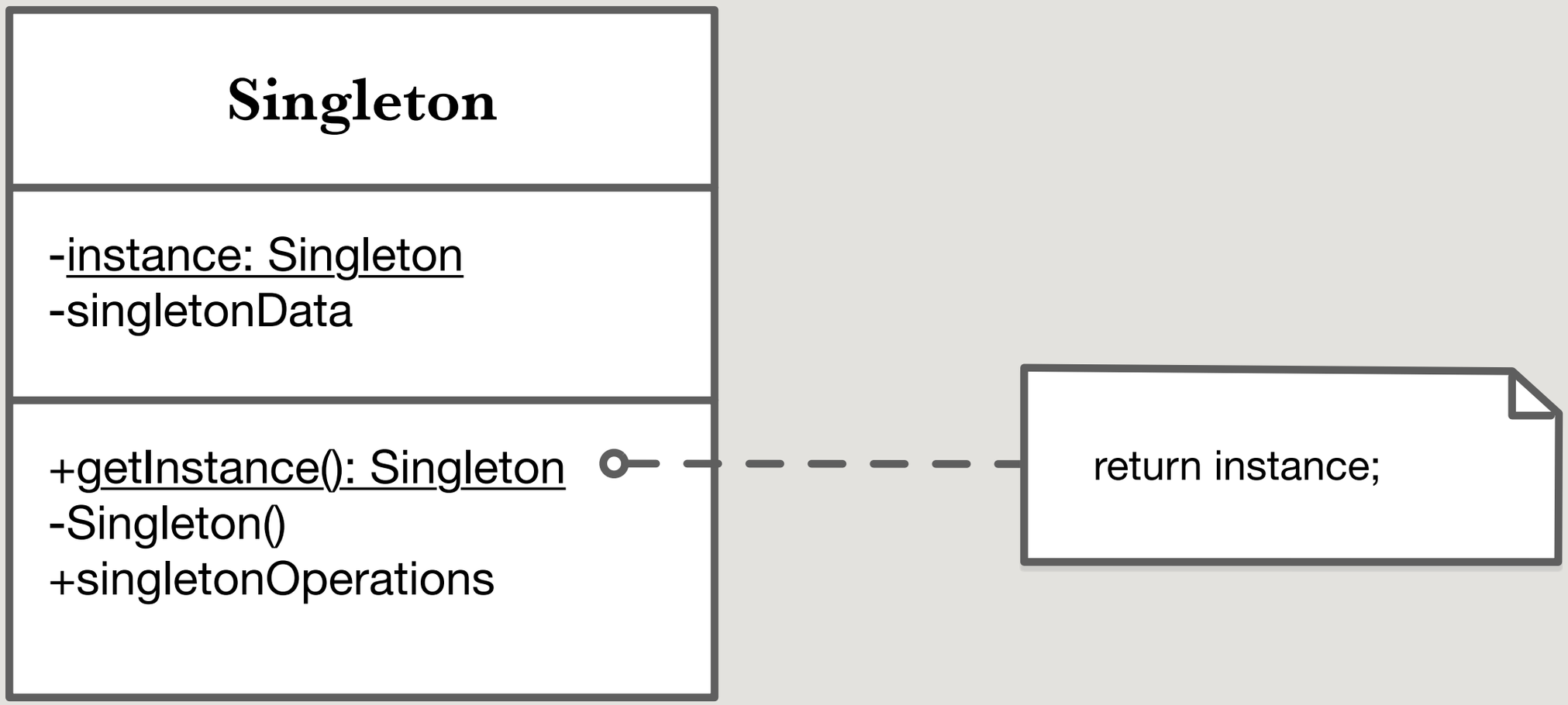
```java
public class Logger {

  private static Logger instance = new Logger();

  public static Logger getInstance() {
    return instance;
  }

  private Logger() {
  }

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

Pues bien, esto que hemos hecho es el patrón Singleton.

Patrón Singleton

Patrón de creación, de objetos.

Garantiza que una clase tiene una única instancia, y proporciona un punto de acceso global a ella.


Singleton
-instance: Singleton
-singletonData
+getInstance(): Singleton
-Singleton()
+singletonOperations
return instance;

Ojo con cómo lo usemos.

Al poder acceder a ellos desde cualquier lugar de la aplicación se comportan casi como variables globales.

¿Por qué no una clase con todos los métodos estáticos?

Es más flexible.

En primer lugar, se puede **heredar** de ellos.

Y proporcionar otras implementaciones de los métodos del «singleton».

En primer lugar, se puede **heredar** de ellos.

No es trivial, pero en la sección de **implementación** del libro se discuten diferentes enfoques de cómo hacerlo.

Y un ejemplo de uno de ellos en el código de ejemplo.

Es más fácil cambiarlos en un futuro si fuera necesario cambiar el diseño para que dejasen de ser Singleton.

Que sean «singletons» no quiere decir que tengamos que obtener siempre la única instancia necesariamente a través de su método estático **getInstance**.

También podríamos pasarlos como parámetros a los clientes que lo necesiten, como haríamos con cualquier otro objeto.

Otras consideraciones

Logger.java

```
public class Logger {

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

  // ...

}
```

Inicialización perezosa

Se podría aplicar a un número determinado de objeto.

Firefox File Edit View History Bookmarks Tools Window Help — Thu 3:17 PM

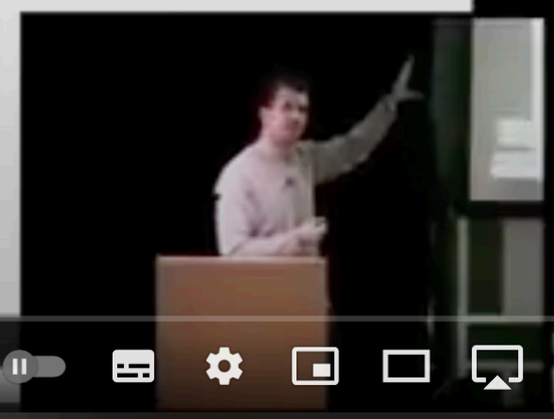
Testing Design Skills - 3 Global State - Google Docs

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.



Google Docs http://go/hxhp

13:53 / 54:08

The Clean Code Talks - "Global State and Singletons"

Google TechTalks
347 K suscriptores

Suscribirme

1,6 K | Compartir | Clip | Guardar | ...

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 solo haya un único objeto.

Dos tipos: de consola y fichero.

Dos tipos: de consola y fichero.

Una vez decidido el tipo de logger y creado este, ya no se podrá cambiar.

Dos tipos: de consola y fichero.

Una vez decidido el tipo de logger y creado este, ya no se podrá cambiar.

Se mantiene el requisito de que la instancia del logger creado sea única durante toda la ejecución del programa.

Dos tipos: de consola y fichero.

Una vez decidido el tipo de logger y creado este, 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.

Pensadlo en casa.

Lo veremos el próximo día.