PARTE 3. GoF

DAGSS\_2

Currás Rodríguez, Alexandre

Vila Fernández, David

Índice

[Ejercicio 1 2](#_Toc23880564)

[Diagrama de clases 2](#_Toc23880565)

[Justificación patrones 2](#_Toc23880566)

[Manual de uso 2](#_Toc23880567)

[Ejercicio 2 2](#_Toc23880568)

[Diagrama de clases 2](#_Toc23880569)

[Justificación patrones 2](#_Toc23880570)

# Ejercicio 1

## Diagrama de clases

## Justificación patrones

**Singleton:**

Utilizamos el patrón Singleton en la clase Logger para crear solo una instancia de este objeto, de manera que cada vez que se necesite crear *logs* no haga falta instanciar de nuevo la clase. Esto nos permite extender la clase por herencia, de manera que el usuario pueda utilizar la nueva versión sin cambiar su código.

**Chain of Responsibility:**

Este patrón lo utilizamos para evitar el acoplamiento entre la clase Aplicación y la clase Logger (y sus subclases), creando una cadena de responsabilidad encadenando Loggers.

De este modo no se especifica que Logger debe manejar el mensaje, si no que se establece dinámicamente en función de la prioridad que se le asigna a cada log.

**Template Method:**

Aplicamos este patrón en el método “log()” de Logger, creando el método plantilla “\_log()”. Este método nos permite evitar duplicar código en las subclases, por lo que la superclase contendrá ese código compartido, mientras que las subclases definen los pasos concretos a seguir.

**Command:**

En este framework podemos identificar las distintas subclases de Operation como operaciones que pueden ser almacenadas y utilizadas por el usuario para ser ejecutadas en diferentes momentos, crear macros, guardar registros, etc.

**Observer:**

*(Operation)*

**Factory Method:**

La clase Application del framework define una lista de operaciones, la cual debe ser instanciada por el programador en una subclase, por lo que la clase Application no necesita conocer la clase concreta de los objetos con los que va a trabajar.

## Manual de uso

# Ejercicio 2

## Diagrama de clases

## Justificación patrones

Utilizamos el patrón Proxy mediante FullVersionOperation para representar a la clase Raíz. Concretamente utilizamos el proxy protector por motivos de derecho de acceso, el cual comprobamos llamando al método checkIsFullVersion de la clase LicenseManager.