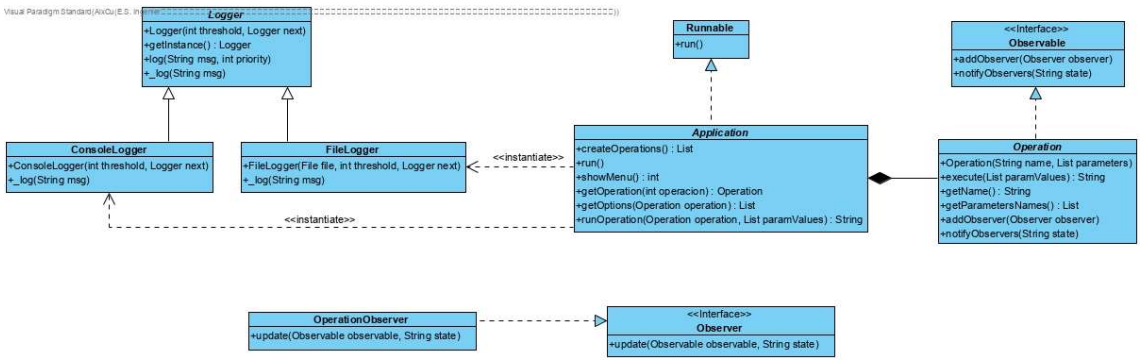
PARTE 1. GoF-3

DAGSS_2

Currás Rodríguez, Alexandre
Vila Fernández, David

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:

Aplicamos este patrón sobre la clase abstracta Operation (que implementa Observable) para poder informar sobre el estado de las operaciones. Basta con invocar en sus subclases al método *notifyObservers(String porcentaje),* que se encarga de llamar al método *update* del OperationObserver asociado a esa operación, el cual muestra el estado por consola.

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

Para utilizar el framework lo primero es extender la clase *Operation* proporcionada con las distintas operaciones que el programador necesite en su aplicación. Las distintas operaciones deben definir en su interior el nombre de los parámetros a solicitar al usuario. Dichos parámetros se almacenan en una lista de *Strings* que se pasará al constructor de la superclase junto con el nombre de la operación.

A continuación, se debe de implementar el método *execute*, que recibe como parámetro una lista de *Strings* que contiene los parámetros solicitados al usuario. Este método debe devolver el resultado generado al aplicar la operación a los parámetros.

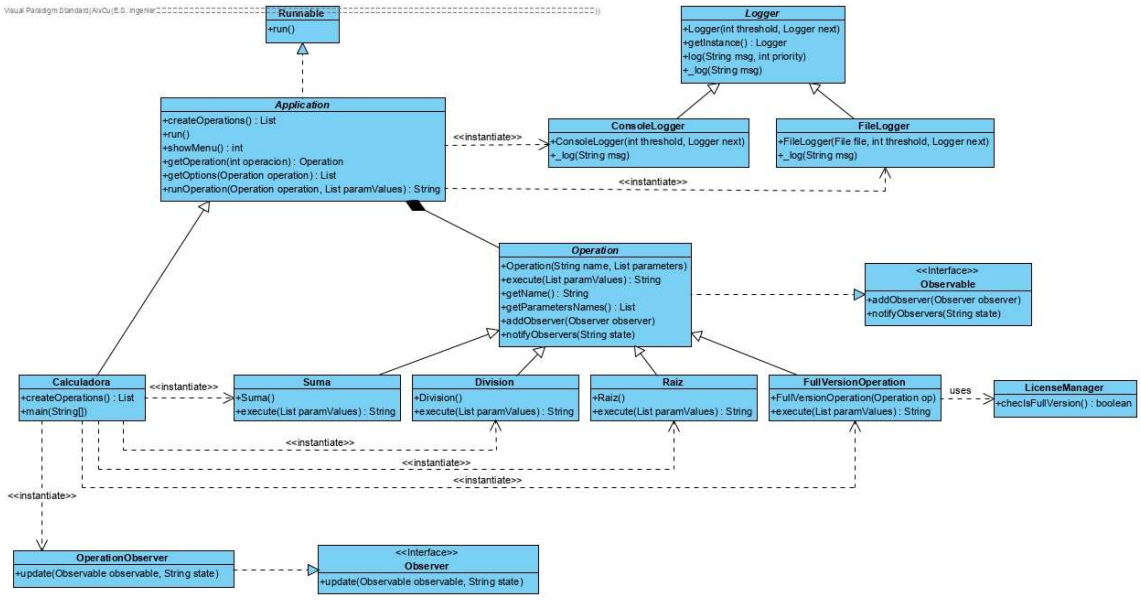
Tras haber definido las operaciones, hay que extender la clase Application del framework, definiendo el método *createOperations()*. Este debe devolver a la superclase una lista con las operaciones implementadas para mostrárselas por pantalla al usuario y que pueda utilizarlas.

Finalmente, para ejecutar el framework es necesario instanciar la subclase creada en el último paso y llamar al método *run()* proporcionado por la interfaz *Runnable* para ejecutar el programa.

Adicionalmente se pueden añadir *OperationObservers* a las operaciones para comunicar su estado al usuario. Esto se consigue llamando al método *notifyObservers(String porcentaje).*

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.