Diseñ o Software
Informe de la Prá ctica de Diseñ o (2020-2021)

·Principios diseñ o:

No pudimos respetar el principio de sustitució n de Liskov porque no procede con los patrones usados en los ejercicios, ademas de que lo complicarí a excesivamente ya que necesitamos subclases que difieran en algunas cosas, como es el caso de la clase 'Modo' del primer ejercicio con las subclases para cada estado, y 'ElementEmp' del segundo con las subclases para un nodo hoja o subarbol.

En cambio el principio de inversió n de dependencia sí pudimos cumplirlo en el primer ejercicio ya que la clase 'Termostato' solo interactú a con la clase abstracta 'Modo' salvo cuando se le asigna una implementació n con el mé todo 'set{Subclase}' pero sin que el cliente conozca la implementació n especifica. En cambio en el segundo ejercicio es imprescindible que el cliente use la subclase directamente.

Tampoco cumplimos el principio de segregació n de interfaces ya que en ambos ejercicios usamos clases abstractas como 'Modo' y 'ElementEmp' que reú nen atributos y mé todos comunes entre todas las subclases lo que evita la escritura de có digo idé ntico, pero imposibilita dividir estas caracterí sticas en clases diferentes al solo poder heredar de una.

El principio de responsabilidad ú nica sí pudimos respetarlo mayormente, es cierto que reunimos varias funcionalidades en las clases abstractas padre y luego en las subclases apenas contienen las que interesan redefinir, pero de otra forma tendrí amos que repetir mucho có digo, ademá s, los mé todos está n relacionados con sus propias clases y nunca hacen el trabajo de otras por medio de getters y setters.

Tambié n cumplimos el principio de abierto-cerrado en ambos ejercicios, ya que solo habrí a que crear una nueva subclase de 'Modo' y añ adir un mé todo para cambiar a ese modo en 'Termostato' en el ejercicio 1, en el 2 solo serí a necesario crear una subclase de 'ConjuntEmp' para un nuevo grupo de elementos o una de 'LeafEmp' para un nuevo elemento, por ejemplo 'Profesora' o 'Alumno'.

Aparte de los SOLID tambié n cumplimos los principios de favorecer la inmutabilidad al comunicar las clases mediante Strings salvo cuando es imposible (mé todo para obtener los compañ eros del ejercicio 2), tambié n el Tell, don't ask al solo usar mé todos de las clases ajenas contenidas en un atributo o mé todos propios, salvo necesidades muy especí ficas, y como era de esperar el de Don't Repeat Yourself al concentrar caracterí sticas comunes en clases abstractas.

·Patrones diseñ o:

Para el primer ejercicio usamos el patró n de estados por su simplicidad al evitarnos un montó n de estructuras condicionales, tan solo necesitamos un atributo de la clase 'Modo' que contiene la instancia del modo actual y unos mé todos para cambiar el valor de este atributo a otra implementació n, de esta manera habilita la posibilidad de añ adir nuevos estados fá cilmente. Para este ejercicio usamos el diagrama de estados.

En el segundo usamos el patró n composició n ya que es perfecto para almacenar estructuras de datos en forma de á rbol como el requerido, ademá s de que tareas como añ adir un equipo repleto de trabajadores a un proyecto o recibir toda su informació n solo requieren de un mé todo ('addComp', 'getInfo'...) gracias al polimorfismo, aparte de evitarnos escrbir mucho có digo. En este usamos el diagrama de secuencia.