Beltrán Aceves Gil

Melisa Barro Hermida

Informe Práctica de Diseño de Software

Principios de diseño:

Ejercicio 1:

-Abierto-cerrado: se puede observar en la clase abstracta CSV_Estado, que permite añadir nuevas funcionalidades sin romper el contrato con la interfaz CSV_Estado_Interfaz, cualidad que obtenemos al poder aprovechar los mecanismos de herencia. A su vez la clase abstracta Field nos permite crear subclases de la misma que contengan datos de tipos diferentes.

-Sustitución de Liskov: todas las subclases de Field y CSV_Estado, ya que aunque sobrescriben métodos de sus clases padre, siguen cumpliendo los contratos a los que están sometidos (Field a la lista<Field> y los estados a la interfaz CSV_Estado_Interfaz), por lo que al programa le es indiferente que le pasemos una clase padre o una subclase.

-Inversión de Dependencia: en la lista(declarada como interfaz Lista<>, pero realizada como ArrayList), las subclases de Field y los estados, ya que funcionan como clases abstractas y podemos cambiar la implementación de tanto las subclases de Field y CSV_Estado.

Ejercicio 2:

-Abierto-cerrado: se puede observar en las clases abstractas OperaciónBinaria y OperaciónUnaria, que permiten añadir nuevas operaciones sin romper el contrato con la interfaz Expresion, ya que la responsabilidad de operar() y representación recae sobre estas operaciones y no de la clase que las llama.

-Sustitución de Liskov: todas las implementaciones de la interfaz Expresion, ya sea Constante o las subclases de OperaciónUnaria y OperacionBinaria, ya que a la clase Calculadora le es indiferente recibir una instancia de cualquiera de ellas.

-Inversión de Dependencia: como en el caso anterios, todas las implementaciones de la interfaz Expresion, ya sea Constante o las subclases de OperaciónUnaria y OperacionBinaria, lo cumplen al asumir las responsabilidades las expresiones y no Calculadora a la hora de evaluar y representar. Asi las implementaciones concretas nos dan igual, mientras cumplan el contrato.

Patrones de diseño:

Ejercicio 1: este ejercicio utiliza el Patrón Estado (realizado a través de la interfaz CSV_Estado_Interfaz), ya que el comportamiento del método interpretar(String) : List<Field>

depende de la situación en la que se encuentre el programa al ir leyendo la String de entrada. Por lo tanto, si delegamos el funcionamiento en los estados, podremos añadir en el futuro otras situaciones para las que no este preparado el programa en este momento. Se mostrará en un diagrama de estados.

Ejercicio 2: este ejercicio utiliza el Patrón Compuesto (realizado a través de la interfaz Expresión, y las clases abstractas OperacionBinaria y OperacionUnaria), ya que así podemos formar el árbol de expresiones de forma más sencilla. A su vez, nos tenemos que olvidar entre elementos primitivos y contenedores, porque lo son entre ellos. De esta manera Calculadora puede tratar de la misma manera a todo tipo de operaciones y valores, se pueden añadir nuevas operaciones. Se mostrará en diagramas de secuencia.