**Taller #5 - Patrones**

**Patrón seleccionado:** Estrategia [1]

Es un patrón de propósito de comportamiento, por lo que caracteriza las maneras en que diferentes clases y objectos interactúan y distribuyen responsabilidades. Específicamente, el patrón de Estrategia contiene una familia de algoritmos intercambiables para que se puedan usar de manera intercambiable. De este modo, los algoritmos varían independientemente del cliente que los use, por lo que promueven flexibilidad y extensibilidad. Este patrón provee un mecanismo para seleccionar de manera dinámica el algoritmo según condiciones específicas [1].

**Proyecto:** jsoup

**Repositorio:** <https://github.com/jhy/jsoup>

**Descripción:** *jsoup* es una librería para el análisis (*parsing*) de HTML. Se utiliza para editar, limpiar y extraer/raspar (*scrape*) información de documentos HTML, mientras que protege contra ataques de XSS (Cross-Site Scripting) [2].

**Estructura general:** La librería se compone de una clase principal *Jsoup* que provee acceso a las funcionalidades de jsoup. La principal funcionalidad es el método *parse*, el cual utiliza los métodos de la clase *Parser* para analizar el documento HTML y armar su representación en un árbol de nodos, donde un *TreeBuilder* arma un árbol de *LeafNodes*. El documento HTML también se representa como un nodo que contiene a otros nodos, el cual es de clase *Document* y hereda de *Element* (ver *fig 1.*).

**A picture containing text, diagram, plan, parallel

Description automatically generated**

**Figura 1. Representación de un nodo en el árbol y la clase principal y la relación entre *Parser* y *TreeBuilder*.**

**Retos:** Entre los retos de diseño que enfrenta el proyecto se encuentra la variabilidad que presenta un documento HTML en los datos que contiene, por ende, se debe de pensar en la manera ideal de almacenar la información del documento, donde mantenga la jerarquía y el orden. Otro reto es manejar documentos de HTML dinámicos generados con JavaScript, lo que implica la posibilidad de que un elemento cambie de estado después de su revisión. Consecuentemente, el diseño debe de tener en cuenta la posibilidad de que los elementos ya leídos cambien de estado.

**Fragmento de interés:** El fragmento de interés donde se encuentra el patrón de Estrategia es la clase *Parser* y su asociación con el *TreeBuilder* (ver *fig 2.*). Este fragmento se deriva de la estructura del patrón Estrategia descrito en el libro guía (ver *fig 3.*).

A diagram of a tree builder

Description automatically generated with low confidence

**Figura 2. Fragmento de interés: implementación del patrón de diseño Estrategia en jsoup.**

A diagram of a strategy

Description automatically generated with low confidence

**Figura 3.** Estructura del patrón Estrategia [1].

**Aplicación del patrón:** Como se puede evidenciar, el patrón es aplicado para intercambiar los algoritmos de construcción del árbol que representa el documento HTML. La clase *HtmlTreeBuilder* crea un DOM (Document Object Model) apartir de los Tokens del documento, mientras que la clase *XmlTreeBuilder* se emplea cuando se quiere analizar (parse) un documento XML o un documento HTML sin aplicar las reglas DOM HTML para ignorar estructuras específicas de HTML.

Estos algoritmos se emplean dentro del contexto del *Parser*, la clase encargada de llevar a cabo la lectura del documento. Los participantes del patrón son la Estrategia, que se declara como una interfaz común para los algoritmos contenidos, las Estrategias Concretas, que implementan un algoritmo utilizando la interfaz de Estrategia, y el Contexto, que se referencia al objeto Estrategia y le permite acceder a sus datos para conocer qué algoritmo utilizar dado el caso. De este modo, el Contexto utiliza la interfaz de Estrategia para llamar a uno de los algoritmos definidos en las Estrategias Concretas.

**Implementación:**

|  |  |
| --- | --- |
| **Ventajas** | **Desventajas** |
| * Permite a la clase *Parser* realizar su trabajo sin importarle el tipo de documento que está analizando, o sin importarle qué tipo específico de información debe de recolectar. * El patrón Estrategia permite flexibilidad y extensibilidad, al poder intercambiar algoritmos según lo requiera el usuario, * En el caso de que se desee que jsoup analice otros tipos de documentos, solo se debe de encapsular un nuevo algoritmo, sin modificar el código del *Parser* (Contexto). * Simplificación del código al evitar declaraciones de condicionales o casos de selección para diferentes tipos de documento (HTML o XML) o de lectura (con o sin reglas DOM HTML). | * Los documentos o tipos de lectura se limitan a tener una estructura de árbol para que se pueda emplear la Estrategia. * Se agrega un capa más de delegación de instrucciones, lo cual puede afectar el tiempo de ejecución. |

**Alternativas de solución:** Otra forma de solucionar los problemas que resuelve el patrón es tener dos tipos de *Parser* diferentes, pero esto implicaría una mayor extensión de código. Otra manera sería solo tener un *Parser* que conozca el tipo de lectura que eligió el usuario, y dentro de la clase revisar por medio de condicionales el tipo de lectura para ejecutar diferentes métodos dependiendo del caso; pero, una vez más, esto implica mayor extensión de código con la gran cantidad de condicionales que se deberían de incluir.

**Referencias:**

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides, y G. Booch, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1.a ed. Addison-Wesley Professional, 1994.

[2] J. Hedley, «jsoup». [En línea]. Disponible en: https://github.com/jhy/jsoup