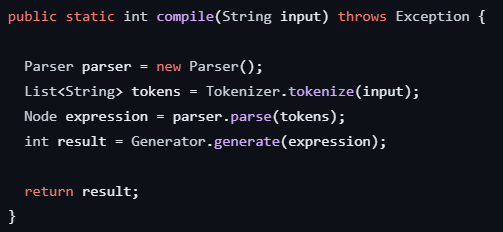
1. **Información general del proyecto**

url: [100-words-design-patterns-java/src/main/java/com/hundredwordsgof/facade at master · dstar55/100-words-design-patterns-java · GitHub](https://github.com/dstar55/100-words-design-patterns-java/tree/master/src/main/java/com/hundredwordsgof/facade)

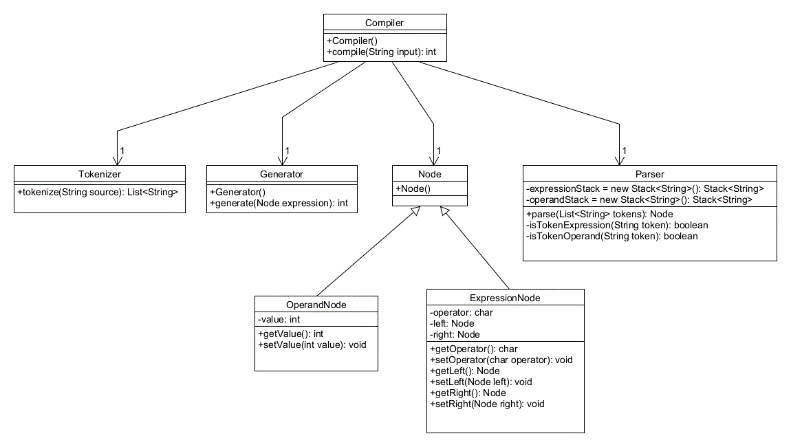
En el proyecto se aborda la creación de un compilador para un nuevo lenguaje de programación, con el propósito de permitir la ejecución de programas escritos en ese lenguaje. Entre las distintas tareas que tiene un compilador en el proyecto se incluye escaneo, convertir en tokens, análisis sintáctico, construcción del árbol de sintaxis abstracta, generación de código, entre otros. Esto lleva grandes retos de diseño y de programación como garantizar la correcta interpretación y traducción del código. Este proceso implica la implementación de subcomponentes para cada paso de la de compilación.

La estructura que utiliza el proyecto es facade (fachada).

1. **Información y estructura del fragmento del proyecto donde aparece el patrón.**

El patrón facade se aplica en la clase "Compiler.java". Esta clase es la fachada del sistema de compilación, pues es una interfaz simple y unificada para el cliente que desea compilar un código. En lugar de que el cliente interactúe directamente con los subcomponentes complejos como Tokenizer, Parser y Generator, el cliente solo debe usar el método "compile()" de la clase Compiler, que oculta la complejidad y realiza los pasos de compilación necesarios utilizando los subcomponentes.

Como podemos ver en el diagrama de clases, la clase Compiler es la encargada de interactuar y orquestar a las demás clases, reduciendo todo a un único método.

****

1. **Información general sobre el patrón: qué patrón es y para qué se usa usualmente.**

El patrón facade se utiliza para ocultar la complejidad de un sistema mediante la creación de una interfaz simplificada. Su objetivo es proporcionar una fachada que actúe como punto de entrada para el cliente, permitiéndole interactuar con el sistema de manera más sencilla sin tener que conocer los detalles internos.

1. **Información del patrón aplicado al proyecto**

En el proyecto del compilador, el patrón facade se está utilizando para crear una clase Compiler que actúa como la fachada del sistema. La clase Compiler presenta una interfaz uniforme y simplificada al cliente, ocultando la complejidad de los subcomponentes individuales involucrados en el proceso de compilación

1. **¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?**

Tiene sentido utilizar el patrón facade en este punto del proyecto del compilador porque simplifica la interacció. La clase Compiler oculta la complejidad y los detalles de implementación. Esto facilita el uso del compilador, ya que el usuario solo necesita llamar al método "compile()" de la clase Compiler en lugar de procuparse por las demás clases.

Las ventajas de utilizar el patrón Facade en este caso son:

* Simplificar interacción, pues el usuario solo necesita conocer la clase Compiler y su método "compile()".
* Ocultar la complejidad permite cambios internos en los subcomponentes sin afectar al usuario.

1. **¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?**

Una posible desventaja de utilizar el patrón facade en este punto del proyecto es que la clase Compiler es única y es limitada. Pues si el usuario necesita acceder a funcionalidades específicas de los subcomponentes, puede haber restricciones y problemas en el cómo puedo acceder. El patrón Facade está diseñado para simplificar el uso general, pero puede no ser adecuado si se requiere un mayor nivel de personalización y control.

1. **¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?**

En lugar de una única clase Compiler como fachada, se podrían tener múltiples modulos o fachadas para modelar diferentes aspectos del proceso de compilación.

Un modulo encargado de crear tokens, otro del parser, y que el usuario, sin preocuparse por el cómo están implementados, pueda usar estas nuevas fachadas de acuerdo a sus necesidades lo que permitiría una mayor flexibilidad y adaptabilidad, ya que cada fachada podría enfocarse en una funcionalidad específica, permitiendo personalización y eliminando limitaciones