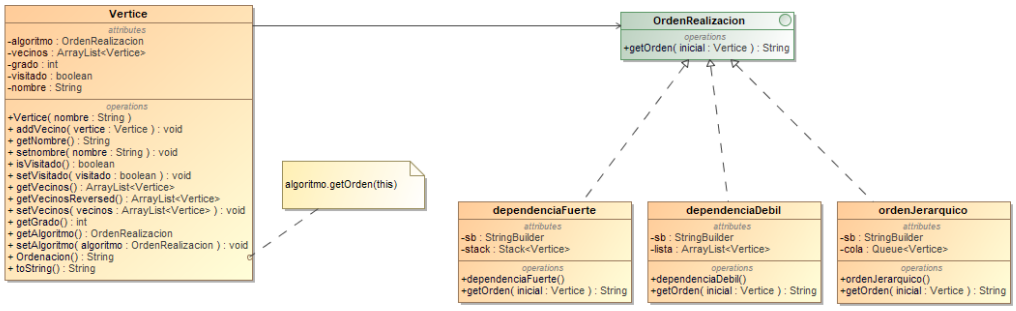
EJERCICIO NÚMERO 2

En este trabajo a realizar, nos piden la creación de un software para gestionar proyectos en una empresa, los cuales, tienen dependencias entre si. Se pide realizar la implementación de 3 algoritmos para recorrer el orden de tareas teniendo en cuenta, a su vez, el orden alfabético para el orden de las mismas. Tras reunir la información necesaria para solucionar este problema comenzamos a trabajar.

Lo primero en lo que pensamos es en que la lista de tareas con sus dependencias puede variar en cualquier momento así como el orden que queremos aplicar sobre estas por lo que crearemos un software flexible para añadir fácilmente nuevos algoritmos. Sin pensarlo dos veces, se nos ocurrió utilizar el **Patrón Estrategia** debido a la flexibilidad que aporta para añadir nuevos tipos de algoritmos de busqueda.

Nuestra implementación con el **Patrón Estrategia** será de la siguiente forma:

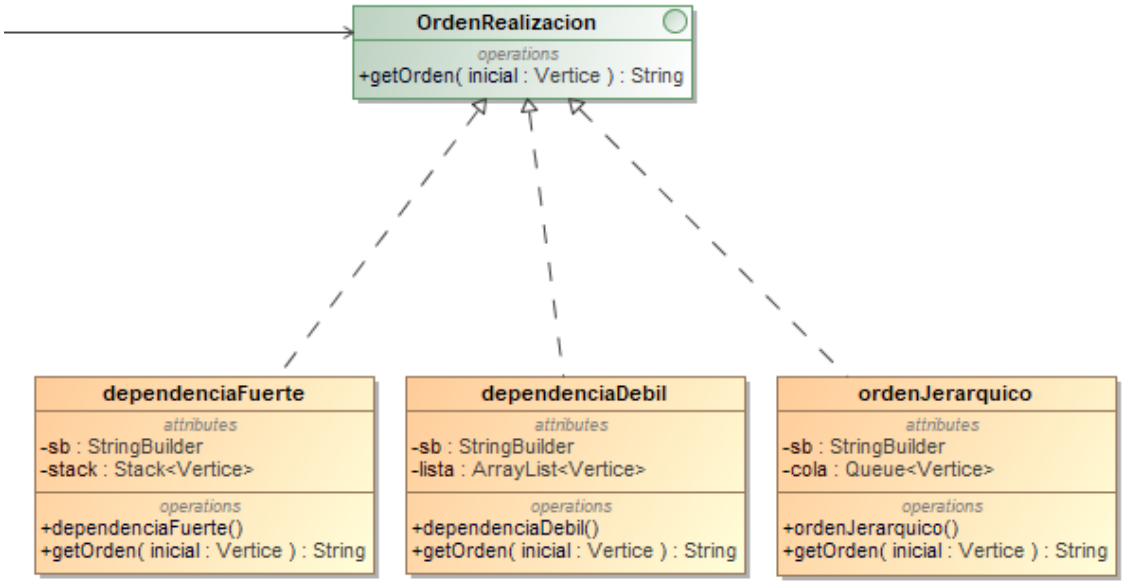


En donde Vertice jugaría el rol de **Contexto**.
OrdenRealizacion jugaría el rol de **Estrategia**.
Y dependenciaDebil, dependenciaFuerte y ordenJerarquico jugaran el rol de **Estrategia Concreta**.

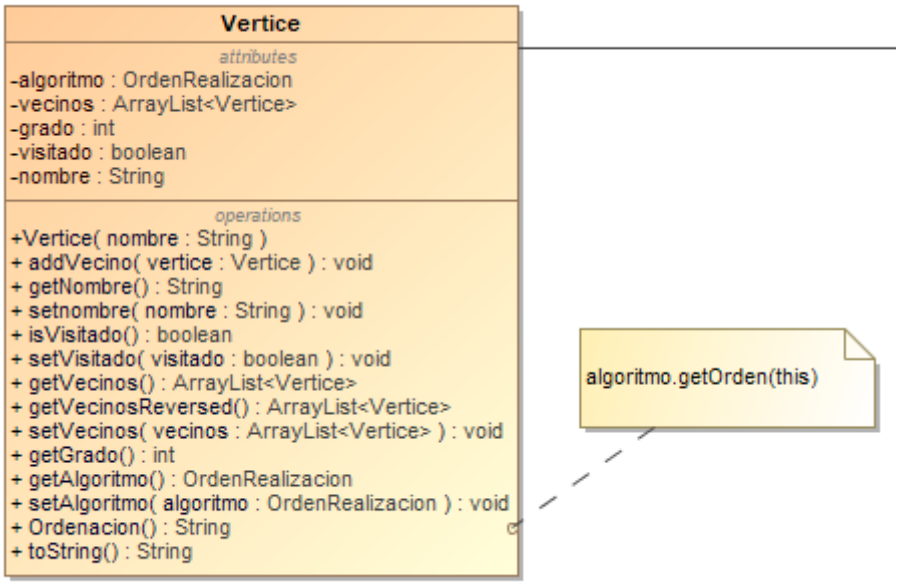
Como hemos mencionado antes, el patrón que escogimos fue este ya que nos permite tener una gran flexibilidad a la hora de aumentar los algoritmos de ordenación.

Entre las **ventajas** de este patrón destacan, además de la ya mencionada, que evita tener que utilizar múltiples sentencias condicionales en el contexto para elegir el algoritmo de ordenación adecuado, y que además permite representar de forma sencilla familias de algoritmos factorizando sus partes comunes en una misma clase padre.

Además, como se puede apreciar mas claramente en esta imagen, en caso de querer añadir nuevos algoritmos de búsqueda, simplemente tendríamos que crear un nueva clase concreta con el algoritmo en cuestión y hacer su diseño correctamente, ajustándose a la interfaz usada.



Antes de pasar a los principios de diseño, cabe destacar que, como podemos ver:



Vertice encapsula una instancia de OrdenRealizacion y además delega la ordenación en el objeto algoritmo pasandose a si mismo como parámetro.

PRINCIPIOS

Entrando ya con los principios de diseño empleados en el ejercicio, el principio mas obvio es el **Abierto-Cerrado** ya que en nuestro código pueden ser añadidos en cualquier momento nuevos algoritmos simplemente creando una nueva clase que implemente la interfaz OrdenRealizacion, consiguiendo que no afecte a las demás clases que ya residen en el código, (nunca debería de suceder con un uso adecuado). Además, al tratarse de interfaces las clases se ven obligadas a implementar sus métodos, a diferencia de las abstractas.