Amalia Carbonell

202122079

Taller 5 - Patrones

Patrón a investigar : Strategy

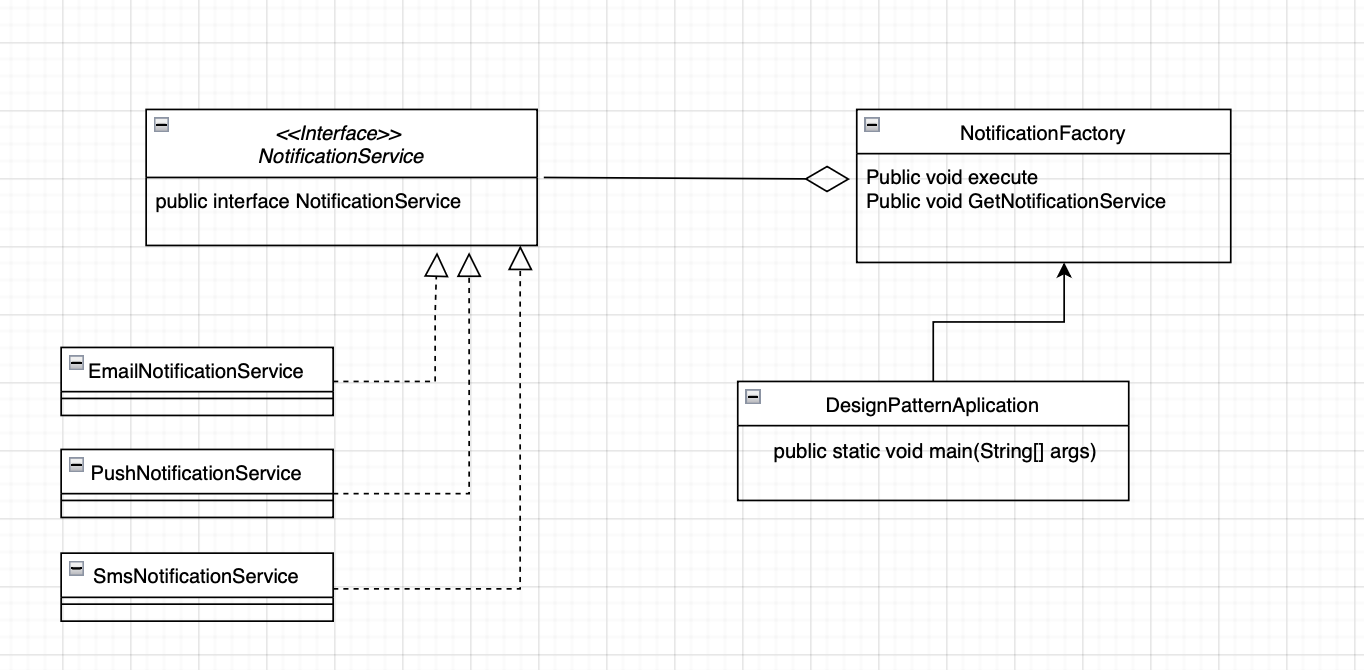
Proyecto : <https://github.com/thegeekyasian/strategy-pattern-in-spring-boot.git>

**Información General del Proyecto:**

Este proyecto el cual por medio del patrón de diseño Strategy desarrolla un Java Spring Boot. Este es una herramienta de código abierto que facilita el uso de frameworks basados en java para crear microservicios y/o aplicaciones web. Esta aplicación tiene varias funcionalidades como dar la notificación de email, los servicios de notificaciones, el tipo de notificaciones, push notificaciones y sms notificaciones.

**Información y estructura del fragmento del proyecto:**

En el repositorio GIT, dentro de la carpeta src/main después dentro de la carpeta the geeky asian podemos encontrar la carpeta llamada design Patterns y por último Strategies. En este lugar podemos ver las estrategias creadas por el proyecto para realizar las funcionalidades mencionadas anteriormente. Estas notificaciones o algoritmos dentro de la carpeta de estrategias van a ser llamadas por la clase Notification Service la cual es una interfaz “a cargo” de las demás notificaciones. Esta clase notification service está conectada con la clase Notification Factory la cual crea una notificación según las especificaciones mandadas. En un diagrama UML bastante simple estas clases se verían de la siguiente manera.

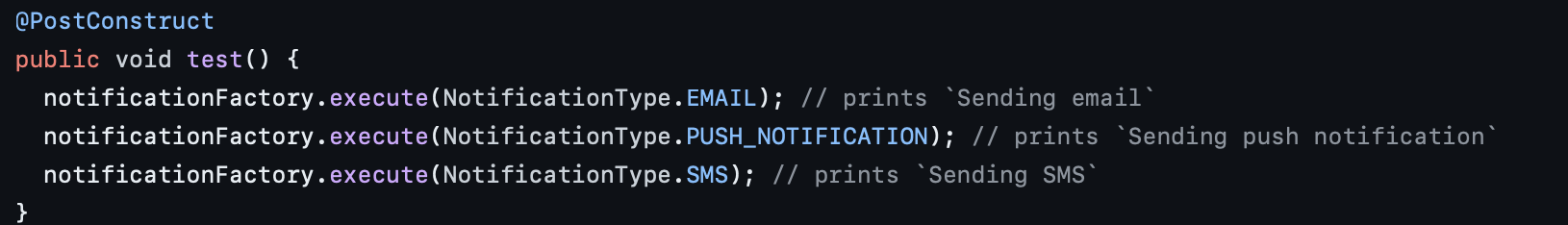


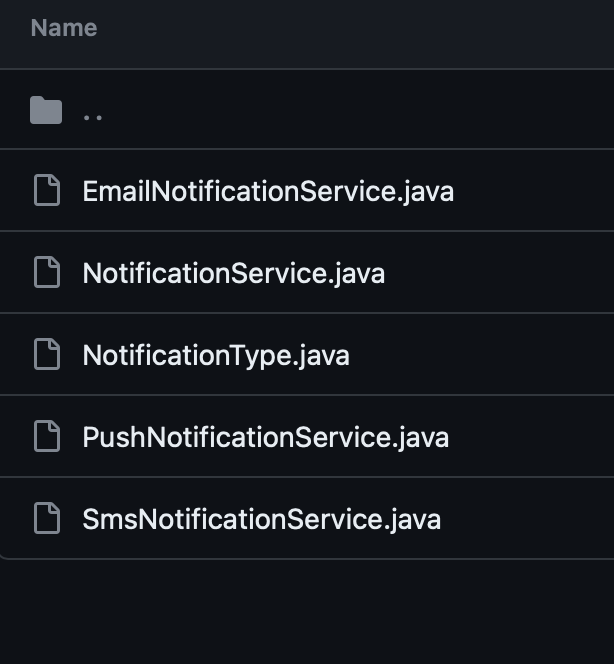
**Información General del Patron:**

El patrón escogido es el patrón Strategy. Este patrón permite definir una familia de algoritmos, crear una clase para cada uno de ellos y encapsularlas y hacer que sus objetos sean intercambiables. Este patrón es usualmente usado para permitir la selección o el uso dinámico de los algoritmos a la hora de la ejecución. Se usa cuando a un mismo proceso se le puede aplicar varios algoritmos diferentes.

**Información del patrón aplicado al proyecto:**

En el proyecto, el patrón permite crear una familia de algoritmos que definen las acciones que debe realizar principalmente la notificación del Spring Boot. Cuando se va a correr el Notification executer se debe definir qué algoritmo de notificación se va a utilizar.

****

En esta imagen podemos ver como, aunque sea una prueba unitaria, se está ejecutando la creación de una notificación y cada una tiene un algoritmo diferente. La línea de código es la misma pues el algoritmo que se aplica debe funcionar siempre con la misma cantidad de parámetros para que se pueda usar de manera intercambiable. Las palabras EMAIL, PUSH NOTIFICATION y SMS hacen referencia a los algoritmos específicos que pueden ser utilizados.En la siguiente imagen podemos ver las opciones de algoritmos o en este caso tipo de notificación que se puede utilizar. NotificationService es una interfaz que recopila todos los servicios ofrecidos por el sistema de esta manera cuando la clase notification factory cree una nueva notificación no le importara la forma en que cada una se está desarrollando pero sí que cuando se busque hacer una notificación específica ese tipo específico funcione.

**¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto?**  Tiene mucho sentido haber utilizado este patrón en este punto del proyecto pues si en una misma clase se tuvieran los diferentes algoritmos/ tipos de notificación, cualquier cambio en alguna notificación afecta a toda la clase. Esa clase sería muy compleja de entender, mantener y las posibilidades de error en el código por culpa del acoplamiento serían muy altas.

**¿Qué ventajas tiene?**

Una de las ventajas de utilizar el algoritmo Strategy es aislar el comportamiento de cada una de las notificaciones y se puede intercambiar el algoritmo o notificación seleccionada durante la ejecución. Adicionalmente se implementa el principio de Open/Closed pues se pueden introducir nuevos tipos de notificaciones sin modificar el contexto.

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

Una de las desventajas de utilizar este patrón es que el cliente debe conocer muy bien y ser capaz de diferenciar los tipos de notificaciones para poder usar la aplicación de manera correcta. Adicionalmente, si la aplicación sigue en una etapa inicial puede ser más complicado implementar el patrón con clases nuevas e interfaces.

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

Unas de las opciones que yo considero podrían ayudar a solucionar los mismo problemas podrían ser : implementar sub clases separadas para cada algoritmo o una estructura de control condicional. Estas dos opciones requieren más código y un conocimiento alto de las posibles conexiones que este puede realizar mediante un proceso. Estas opciones no serían tan efectivas como lo es el patrón Strategy pues son menos flexibles a los cambios y dificultan el cambio de algoritmo sin modificación del código fuente.

Fuentes:

<https://refactoring.guru/design-patterns/strategy>

<https://github.com/thegeekyasian/strategy-pattern-in-spring-boot.git>