Análisis patrón de diseño controlador de bases de datos

j.amayab@uniandes.edu.co

Mayo 2023

1 Información general

Link del proyecto: https://github.com/Jahyrm/SQLUC

El proyecto es un controlador de bases de datos simple, es decir una implementación de un lenguaje SQL dentro de Java cargando tablas, uniendo tablas, eliminando registros, modificándolas, actualizándolas, seleccionando y creando. Además, tiene una función de encriptar, sin embargo, esta función es un llamado aparte para cada campo dentro de la tabla.

El reto de este proyecto es realizar una acción distinta por cada instrucción, esto se desencadena en un alto acoplamiento y una baja o prácticamente nula cohesión. Además, el manejo de datos para cada instrucción es distinto por lo cual tratar cada instrucción como separado implica un procesamiento distinto de una cantidad de datos enorme. Además, tratar las funciones de encriptación y desencriptación totalmente aparte puede desencadenar un problema de desacoplamiento provocando que la clase no tenga ninguna funcionalidad o por el contrario se generen altos niveles de acoplamiento al ir con los datos de un lado a otro.

El diseño de todo el proyecto esta dado por los siguientes diagramas UML (adjunto con el archivo entregado se encuentran las imágenes originales tienen buena calidad y permiten zoom).

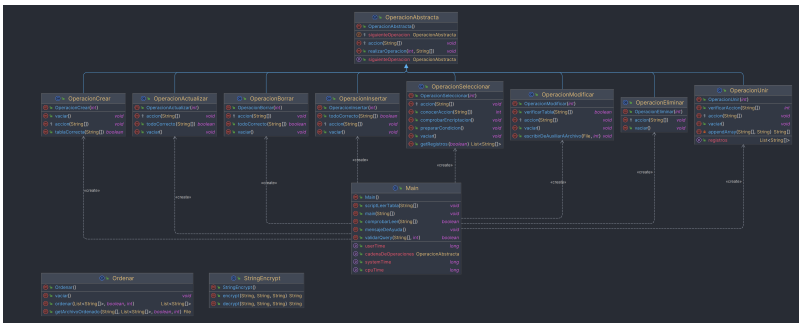

Figure 1: Diagrama de alto nivel del diseño del proyecto del controlador de bases de datos

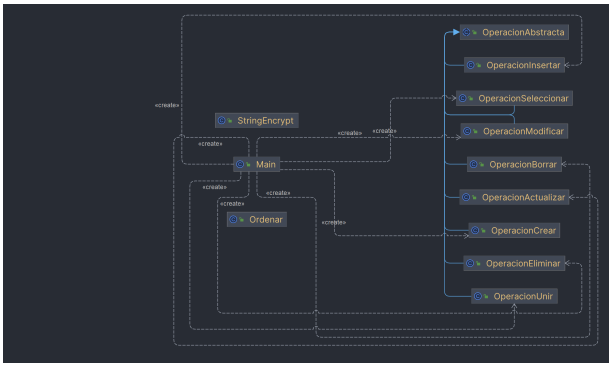

Figure 2: Diagrama de alto nivel del diseño del proyecto del controlador de bases de datos

Sin contar con los elementos de las funciones ordenar y encriptar todos los demás elementos forman parte de un patrón de diseño, en este caso el patrón de comportamiento “chain responsability”. De la figura 2 se puede notar que de la clase main, es decir de la clase que ejecuta el programa, sale una flecha hasta la clase OperacionAbstracta, sin embargo, a pesar de que el uml accede a las otras esto lo hace de forma indirecta ya que solo se comunica a través de ellas por dicha clase . El llamado a la función se hace en la siguiente línea de código:

```
DateFormat formatoHora = new SimpleDateFormat("HH:mm:ss");
String[] vectorQuery;
OperacionAbstracta cadenaDeOperaciones = getCadenaDeOperaciones();
System.out.println("SQLUC version 1.00 " + formatoFecha.format(fecha)
        + " " + formatoHora.format(fecha));
System.out.println("Ingrese .ayuda para sugerencias de uso.");
while (!query.equals("SALIR")) {
```

Figure 3: Llamado a la clase Operacionabstracta como atributo en la clase main

Además, note que la clase "OperaciónAbstracta" es una clase abstracta que no realiza concretamente el método "acción" sino que lo deja para que las clases que la extiendan implementen el método de forma concreta. La clase va buscando entre todas las clases que la extienden cual es capaz de realizar la instrucción del usuario con el método setSiguienteOperación, es decir va recorriendo clase por clase hasta que alguna ejecute la acción.

2 Patrón, aplicación ventajas y desventajas

El patrón chain responsability es un patrón de comportamiento, tal como su nombre lo indica es una cadena en la cual va pasando una instrucción. En primer lugar, se realiza una interfaz o una clase abstracta la cual recibe la instrucción y esta pasa a un receptor el cual decide si ejecuta la instrucción o la pasa al siguiente elemento (otro receptor). El objetivo de este patrón es que el emisor de la instrucción no conozca su receptor, sino que de dicha tarea se encargue la interfaz, así se reduce el acoplamiento. Su contraparte radica en incluir una instrucción para cada acción especifica lo cual maximiza el acoplamiento haciendo imposible la tarea de modificar el programa sin cambiar absolutamente todo el sistema.

Usualmente se usa en casos de manejo de solicitudes jerárquicas. Un ejemplo podría ser el procesamiento de solicitudes de aprobación en una organización, donde diferentes niveles jerárquicos pueden aprobar o rechazar la solicitud.

En el proyecto la clase Main actúa como emisor y las clases que extienden a la abstracta actúan como receptores. El objetivo es que el usuario ingrese cualquier instrucción: Unir, modificar, eliminar, etc, y dicha instrucción pase por los receptores y estos decidan si la ejecutan o no, esto tiene sentido ya que el usuario escribe un comando dentro de la consola (no es una lista de opciones) facilitando el funcionamiento y la interacción con el cliente. La ventaja es que el problema y la dificultad del diseño que es el acoplamiento se minimiza, es decir el configurar cualquier acción no afecta directamente al main o a las otras clases. Además, el código reduce su complejidad ya que no es necesario estar revisando a que clase pertenece cada instrucción, sino que el patrón se encarga de ello. Sin embargo, una de las desventajas es que no todas las instrucciones pueden ser ejecutadas, por ende, debe establecerse una ejecución general por defecto que sirva como acción si y solo si ninguna de las clases de la cadena ejecuto la opción, otra desventaja es que al ir de clase en clase aumenta el tiempo de respuesta si la acción reside en el ultimo elemento de la cadena (esto se puede complicar aún más si son demasiadas las opciones).

Otra forma que soluciona el problema de este proyecto en especifico es la creación de una única clase que agrupe múltiples métodos, como por ejemplo actualizar, eliminar, seleccionar ya que son acciones que se ejecutan sobre una única tabla. La tabla entra como parámetro y se le dan ciertas opciones al

usuario es decir ya no ejecuta un comando sino escribe un número para la opción y de acuerdo con el número se ejecuta cierto método de igual forma que antes el acoplamiento es mínimo. Sin embargo, las otras opciones se ajustarían en otras clases por separado, por ejemplo, modificar ingresaría el campo y la tabla y unir ingresarían dos tablas, pero a diferencia del patrón en estos dos últimos casos no sería optimo.