Explicación DAO

María Turpín Buendía

El primer paso para implementar el DAO (Data Access Object) es tener claro que debemos realizar una serie de clases que nos permitan reutilizar los mismos métodos para la ejecución de sentencias SQL, separando para ello el método que me permite escribir en la BBDD y el que me permite leer de ella.

Por el principio de responsabilidad única también tendremos clases encargadas de realizar la conexión con la BBDD leyendo de las variables de entorno de nuestro sistema. A su vez, es necesario declarar una serie de interfaces que sirven de contrato para cada una de nuestras clases, cumpliendo así con el principio de inversión de dependencia (D de SOLID). Lo correcto sería segregarlas también.

IConfiguration

Se encarga de establecer que la clase Configuration implemente los métodos getUser, getUrl y getPassword que leerán de las variables de entorno para encapsular y establecer la conexión.

Configuration

Esta clase se trata de un Singleton, permite ser instanciada únicamente por ella misma y una única vez. Como se ha mencionado, registra la configuración para conectar la BBDD, esto es, implementa los métodos de IConfiguration.

Statement

Se trata de una interfaz funcional con un método "run" que toma como parámetros un PreparedStatement de SQL y un genérico llamado entity, que bien podría ser una pizza, un ingrediente, un comentario o un usuario y lanza una excepción del tipo SQLException. Gracias a esta interfaz podremos posteriormente insertar los datos de la entidad genérica en el PreparedStatement para completar la query SQL.

Resultset

Se trata de una interfaz funcional que obtiene como parámetros un ResultSet y una entidad genérica. Se encargará de insertar los datos del ResultSet a la entidad en cuestión a través de una función lambda que la implementará en el momento de llamada al método.

IRunnables

Se trata de una interfaz con dos métodos declarados: uno para obtener las querys (getSQL) y el otro para ejecutar un PreparedStatement. Este último método lanzaría una excepción SQLException.

Runnables

Esta clase implementa la interfaz anterior (IRunnables) y tiene 3 atributos marcados como final y private del tipo String SQL, un genérico llamado entity y un Statement<T> (interfaz funcional). Se encarga de implementar los métodos getSQL, que devuelve el valor de su atributo String SQL, y el método run al que se le pasa un PreparedStatement como argumento y se encarga de llamar al método run de la interfaz funcional Statement, a la cual le pasa un Statement<T> y el genérico guardado en la propia clase. Esta clase nos sirve para tener un objeto que guarde los 3 atributos

que nos interesan de cara a la implementación de los métodos en EntityManager, que se explicarán a continuación.

IEntityManager

Interfaz encargada de definir los parámetros que se le pasarán a los métodos de EntityManager y los tipos que devuelve. En ella podemos ver 4 métodos: save, addStatement, addRangeStatement y select. Ya se puede discernir en esta clase lo que se comenta al principio de este documento: que debe diferenciarse la entrada de datos a la base de datos de la lectura con los métodos save y select.

EntityManager

La clase EntityManager implementa la interfaz IEntityManager y es un ejemplo perfecto de fluent interface, ya que sus métodos retornan this a excepción de save y select que serán los métodos que se invocarán en última instancia.

Esta clase consta de 2 atributos: uno de tipo IConfiguration, ya que se encargará de realizar la conexión con la base de datos en los métodos save y select y una lista de IRunnables (cumpliendo así con los principios SOLID). A continuación, se explicarán sus métodos:

- **buildConnection**: Se encarga de obtener las variables de configuración de la conexión con la BBDD a través del objeto Configuration.
- **addStatement**: se encarga de crear un objeto Runnable pasándole los parámetros de sql, entidad y Statement<T>, posteriormente los añade a su lista de runnables. En la llamada a este método se realizará la implementación de la interfaz Statement<T> al pasarle una función lambda en la que preparamos el statement con los datos de interés.
- **addRangeStatement**: su funcionalidad es análoga a la de addStatement, pero está pensada para varios runnables.
- **save**: Este método realiza la conexión con la base de datos y ejecuta todos los runanables de la lista de runnables uilizando para ello transacciones y empleando try catch para no tener que usarlo en ninguna de las implementaciones de los objetos del dao. Esta es la principal ventaja de esta metodología de trabajo. Solamente encontramos try catch en esta parte de código, ahorrando así la ejecución de muchos test innecesarios. Este método está pensado para la escritura en la base de datos.
- **select**: Al igual que en save, aquí realizamos la conexión con la base de datos y realizamos las transacciones al commitear a la misma, así como el rollback y el cierre de conexión. No obstante, en esta ocasión el método devuelve un genérico que solamente podrá ser de tipo Entity a través de una sentencia SQL de tipo lectura (select) para sacar el resultSet y poder asignarlo como parámetros de la entidad en cuestión. Tanto en select como en save en el bloque finally se realiza la limpieza de la variable List<IRunnables> de la clase.

Finalmente tendremos una IDAO e IngredientDAO, la primera es una interfaz con los métodos CRUD que deberán implementar cualquiera de los objetosDAO de los que se componga nuestro dominio. La clase ingredientDAO se encarga de implementarla y hace uso de EntityManager para realizar las inserciones y lecturas de la base de datos. Como podemos ver en la misma, no encontramos ningún try catch a lo largo de la implementación de los métodos y se emplea fluent interface, lo cual facilita la lectura y mejora la calidad del código.