Java y JDBC

Si guardamos informacion de gente que se registra en una pagina, esos datos son muy sensibles. En donde podemos guardarlos?

En una base de datos NO relacional como PostgreSQL, SQL Server, Oracle database, MySQL.

La libreria que permite la conexion entre la aplicacion y la base de datos, se puede conocer como driver. Si tenemos que conectar nuestro programa a varias bases de datos distintas como los ejemplos de arriba, se volvería muy laborioso programar la conexion para cada una.

El JDBC es una libreria que contiene las interfaces de comunicacion y operacion con la base de datos. Si tenemos varios tipos de bases de datos, estas tienen sus drivers particulares y ademas si tienen librerias para java, sus clases implementan a la JDBC.

Entonces solo con abstraer la JDBC vamos a poder conectarnos a cualquier base de datos sin cambiar el codigo.

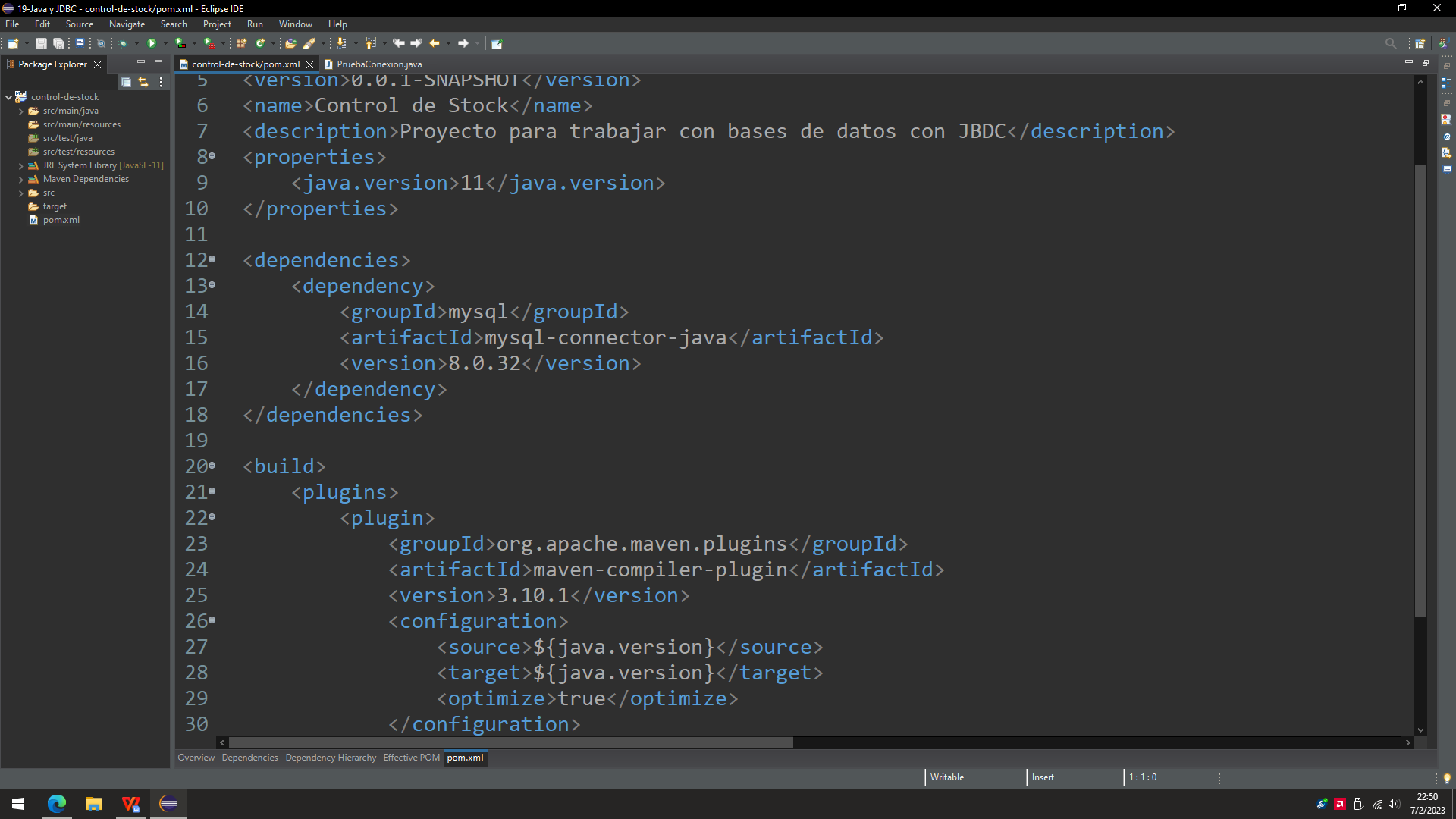
¿Cuál es la ventaja de utilizar una API como la JDBC, basada en interfaces, para realizar la comunicación entre el código y una base de datos relacional?

Transparencia a la hora de elegir la base de datos o cambiar de una para otra, con muy pocos cambios de código. Lo único que va a ser necesario hacer es cambiar la dependencia del driver de base datos y el JDBC se encarga de todo.

Ejercicio.

Despues de haber creado una base de datos llamada control\_de\_stock, voy a unirla a un proyecto Java gracias al driver JDBC.

Primero en el proyecto Maven voy a agregar unas etiquetas al pom.xml para poder conectar el proyecto al mysql.



Luego en el proyecto de java

//Prueba de conexion

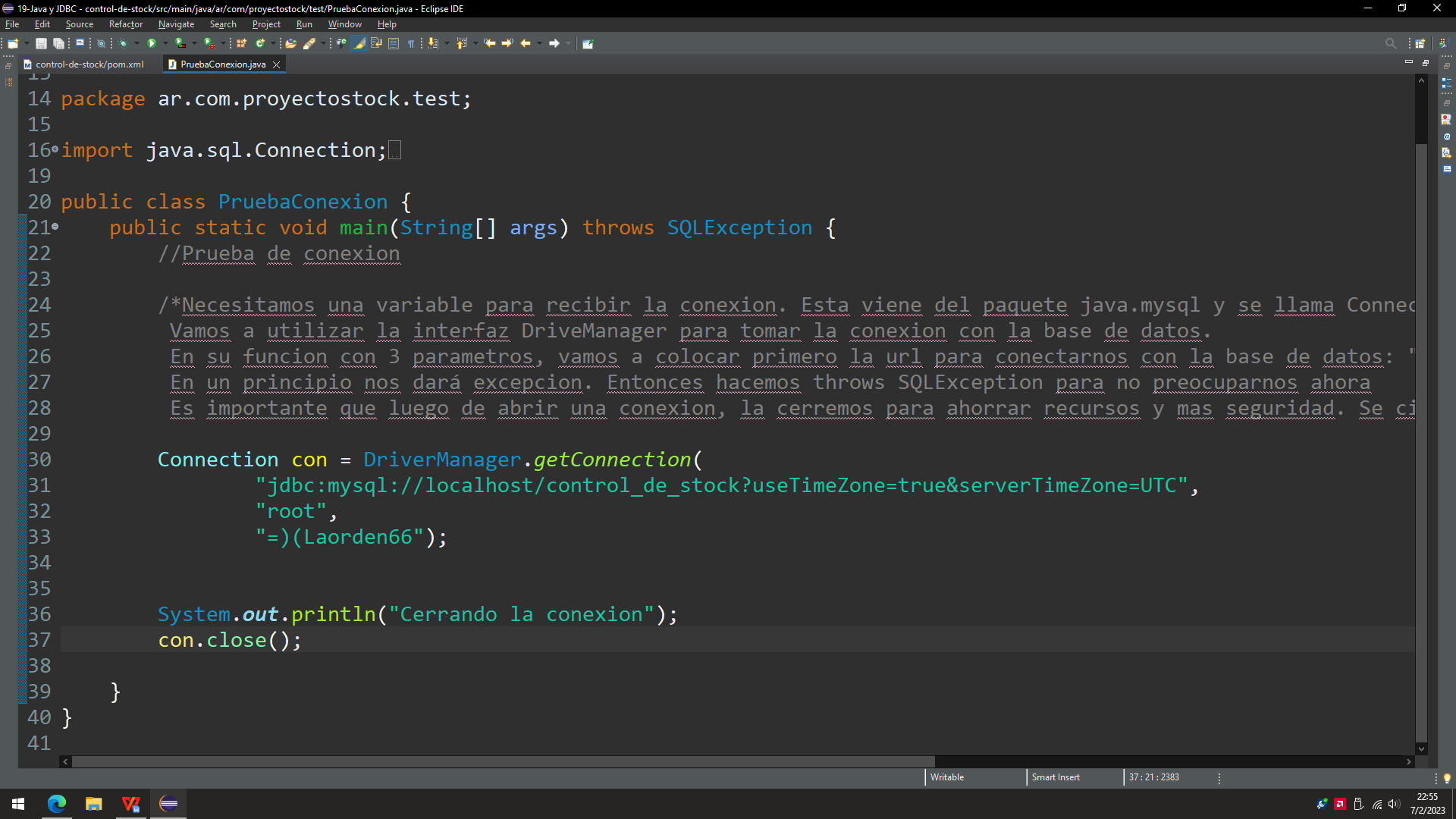
/\*Necesitamos una variable para recibir la conexion. Esta viene del paquete java.mysql y se llama Connection

Vamos a utilizar la interfaz DriveManager para tomar la conexion con la base de datos.

En su funcion con 3 parametros, vamos a colocar primero la url para conectarnos con la base de datos: "jdbc:dbutilizada://localhost/nombre\_cualquiera?propiedades".

En un principio nos dará excepcion. Entonces hacemos throws SQLException para no preocuparnos ahora

Es importante que luego de abrir una conexion, la cerremos para ahorrar recursos y mas seguridad. Se cierra con el metodo .close()\*/



Clase 2

Luego de descargar un proyecto java de prueba que nos brinda el profesor.

Este con el nombre ‘1846-Java-y-JDBC-Trabajando-con-una-Base-de-Datos-aula-2-views’ lo importamos a nuestro workbench.

Vemos que es un control\_de\_stock pero mas avanzado. Si ejecutamos su main vemos que se mostrará una interfaz grafica para administrar los productos del deposito y base de datos. Fue creada con Java Swing que es utilizada para aplicaciones desktop (hoy en dia ya no se usa mucho en el mercado).

Con el backend vamos a hacer la logica de negocio.

ControlDeStockFrame se encarga de crear y mostrar la ventana de la interfaz.

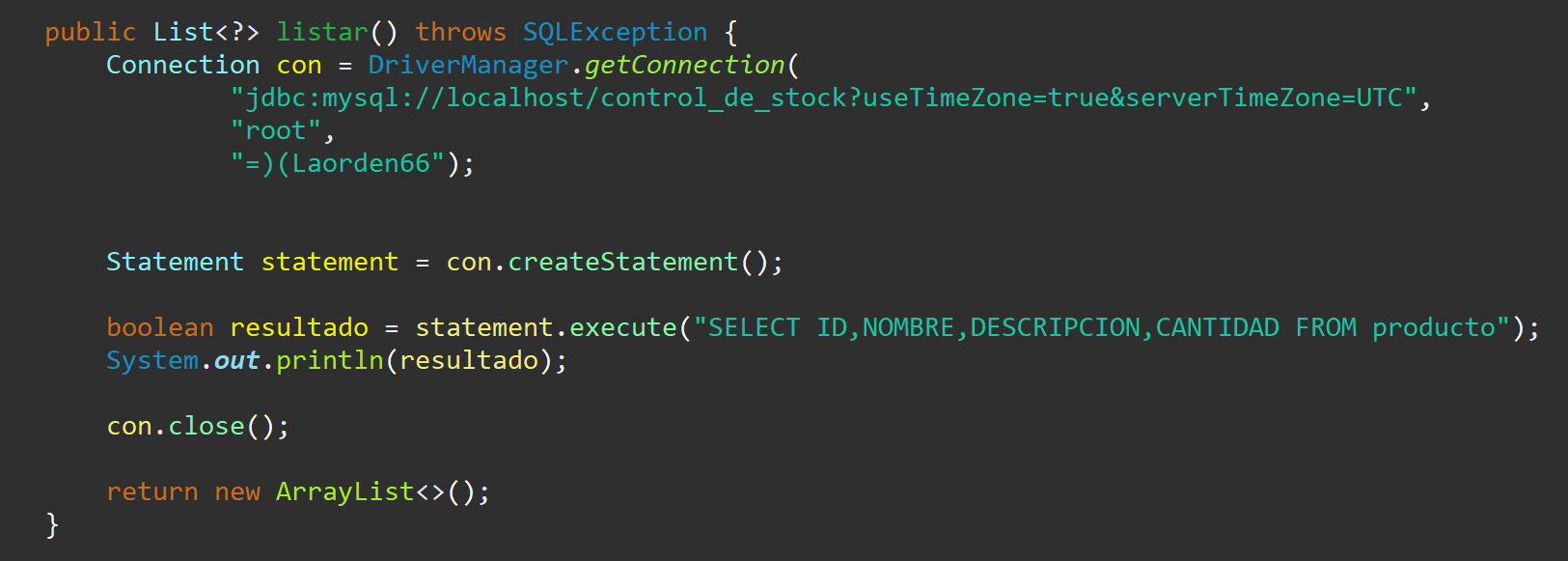
Fijemosnos dentro de cargarTabla el metodo listar. Ahi vamos a hacer lo mismo que el ejemplo de antes. Recordar cerrarla justo despues.

Los comandos de mysql son tambien llamados Statements, y queremos utilizarlos dentro de java.

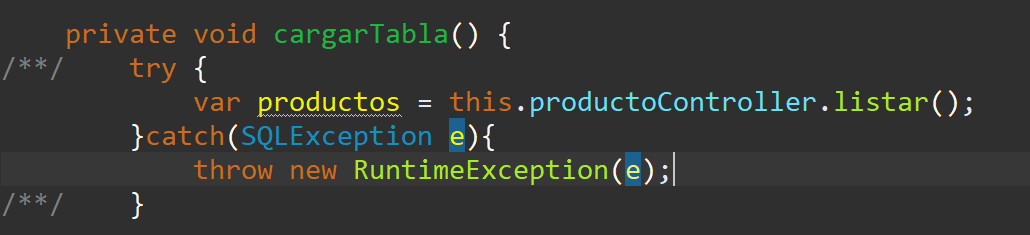
Para utilizar los statements, tenemos que aclarar que conexion vamos a utilizar (con que db nos vamos a conectar), en nuestro caso es la recientemente creada llamada ‘con’. El createStatement() nos devuelve el resultado con el tipo Statement.

Ahora podremos utilizar metodos mysql para esa conexion.

Con la nueva statement creada, podemos utilizar el metodo execute(...) para aplicarle dentro los metodos mysql. El execute nos devolvera un booleano ya que este nos indica si el resultado es un listado o no. Y lo guardamos dentro de una variable. Como nosotros con el execute queremos ejecutar un SELECT, nos dará true. En cambio si quisieramos ejecutar un INSERT,UPDATE o DELETE, nos daria false.



Antes de seguir, vamos a cubrir la excepcion con un throws SQLException y vamos a pasar la responsabilidad de manejar la excepcion a CargarTabla() y lo vamos a tratar con try/catch. Y eso evitará que trabe el programa si ocurre la excepcion.



Ahora vemos que la consola nos devolverá true por el execute, ademas de mostrarnos la interfaz

(NOTA: Le quité el boolean resultado ya que solo lo habia hecho como ejemplo ).

Para tomar el resultado del statement que recien ejecutamos, lo hacemos gracias a getResulSet(). Este nos devuelve un objeto ResultSet.

El ResultSet es un listado de resultados. Si hay una fila en ese ResultSet, podemos ir leyendo cada item. Cuando lleguemos al ultimo item, el loop se termina.

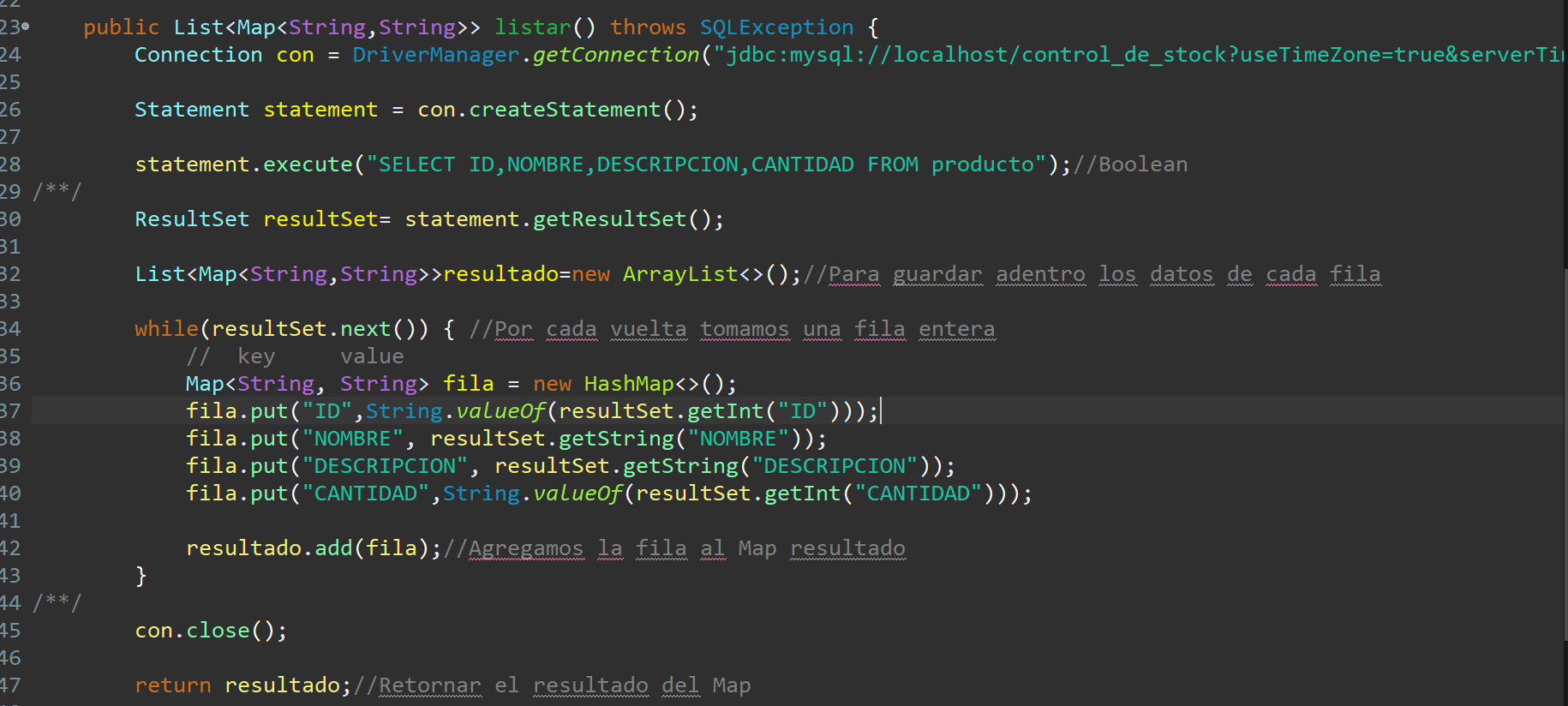
Para movernos a la siguiente fila utilizamos el metodo .next() que da falso si no tiene despues otro resultado. Este podemos utilizarlo dentro de un while. Dentro del while obtenemos los datos de la fila. Dentro creamos un Map llamado ‘fila’ para guardar dentro cada dato de las filas, lo que le metamos tiene que tener llave string y valor string. Si por ejemplo seleccionamos el ID, lo guardamos dentro de un Map gracias a un put. (Recordar que como pusimos la llave y value como tipo string, hay que transformar el int del getInt a tipo String. Para eso, con el String.valueOf tomamos el String del Int)

Y luego tambien lo hacemos con los demas de la fila.

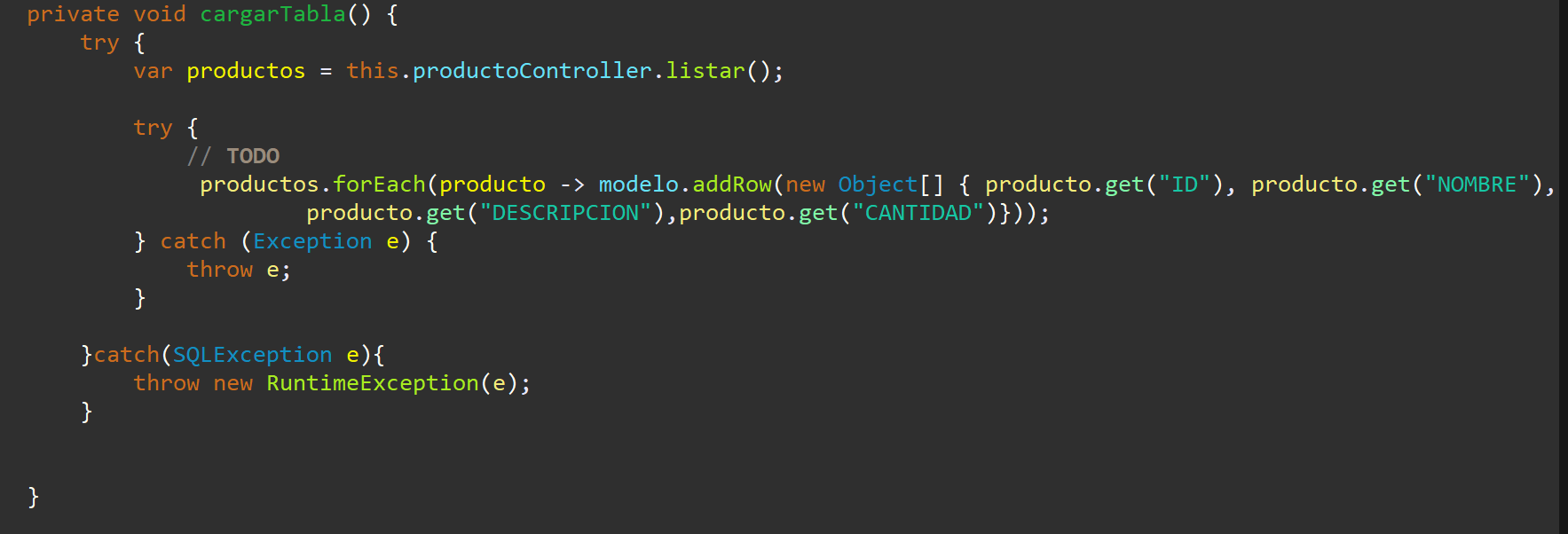
Luego de haber terminado de meter cada valor al Map ‘fila’, este es ingresado al Map ‘resultado’. Y hará lo mismo con cada fila hasta que no haya mas.

Al terminar, nos retornará lo que tenga el Map ‘resultado’

(NOTA: Al List listar() le ponemos como parametro que va a guardar Map<String, String>, refiriendonos a ‘resultado’ que a su vez guardará varias ‘fila’)



En controlDeStockFrame.java, donde está cargarTabla(), voy a acomodar el try/catch para que intente mostrarlo en la tabla.



Si ejecutamos, vemos que se muestra correctamente los elementos que tenemos en la base de datos hasta ahora.

Creando ConnectionFactory

En el caso de que cambiemos de base de datos o el nombre de la base de datos, deberiamos acordarnos de cambiar el nombre en varias partes del codigo para que no haya errores (en losDriverManager por ejemplo).

Para mejorar eso, podemos crear una clase que solamente se va a encargar de la creacion de conexiones.

Cuando encapsulamos la creacion de un objeto en especifico y centralizandola en un solo punto, se le llama Factory Method (Es un patron de diseño). En nuestro caso, queremos centralizar la creacion de conexiones en un punto, podemos llamarlo ConnectionFactory para seguir el standar. Y siempre que la invoquemos, es porque queremos crear una conexion.

¿Cuál es la ventaja de utilizar una ConnectionFactory en nuestra aplicación?

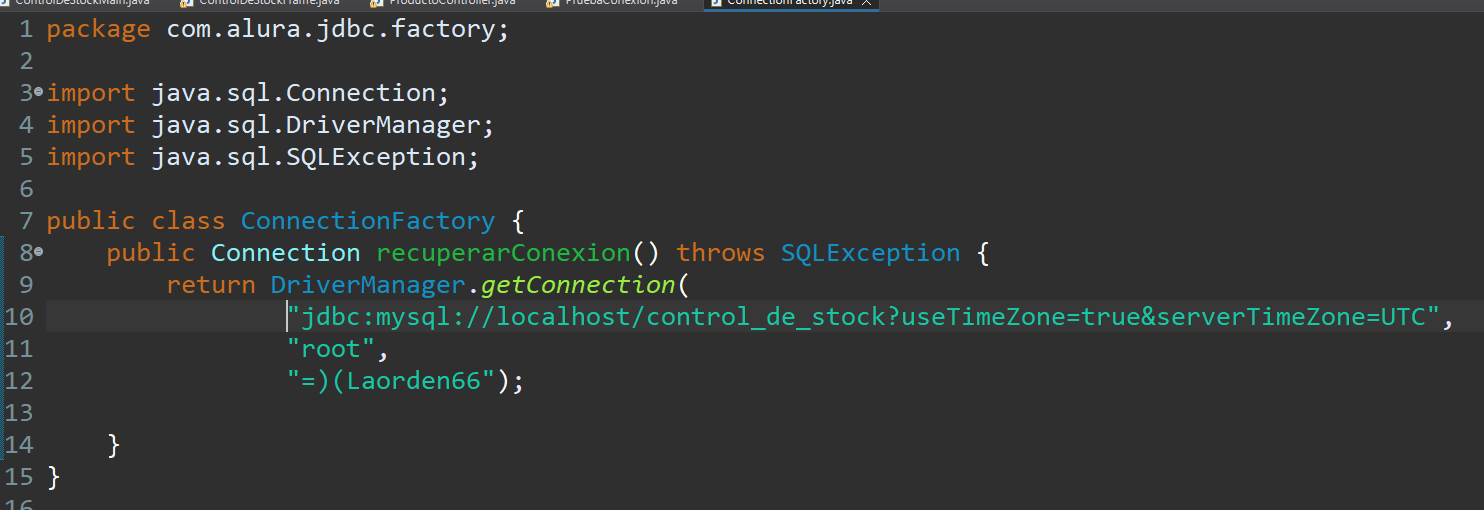
Proveer una forma más sencilla de crear un objeto.

Los objetos son creados sin exponer la lógica o las configuraciones de creación al cliente. Además, es posible referirnos al objeto recién creado utilizando una interfaz (una abstracción), desacoplando la implementación.

Vamos a crear el ConnectionFactory dentro de un paquete que podemos llamar com.alura.jdbc.factory.

Vamos a sacar el DriverManager.getConnection de ProductoController.java y lo colocamos dentro de un metodo recuperarConexion() dentro de ConnectionFactory. Asi cuando creamos una nueva conexion, podemos llamar al metodo recuperarConexion.

new ConnectionFactory().recuperarConexion()

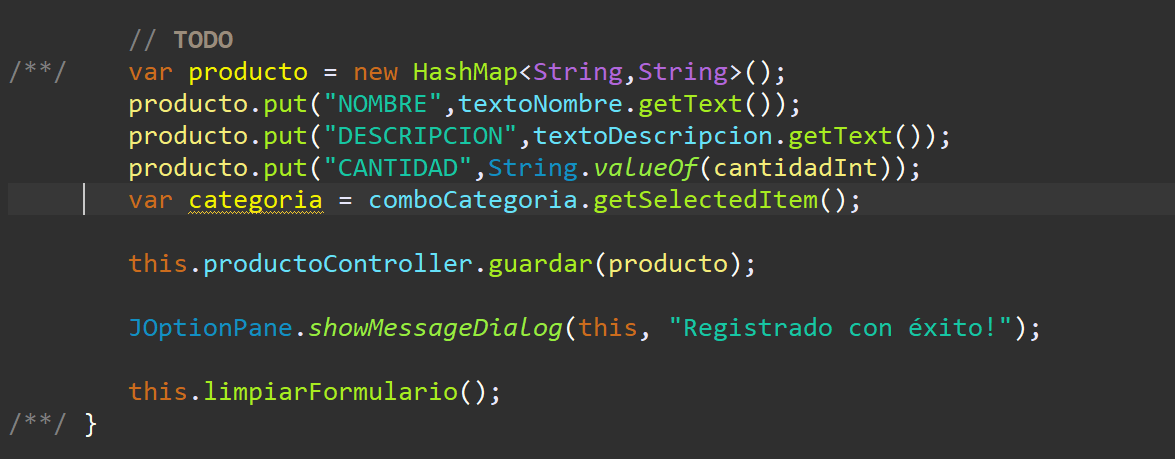


Ahora queremos poder ingresar nuevos productos a la base de datos y guardarlos (gracias al boton guardar) desde nuestra aplicacion.

Dentro de ControlDeStockFrame, vamos a configurarAccionesDelFormulario(). Ahi tenemos varias acciones para los botones como guardar, limpiar, eliminar, modificar.

Entramos al metodo guardar() dentro del BotonGuardar y vemos que ahi transformamos la informacion del formulario a un objeto para despues enviarla por parametro en el metodo guardar(..) que vamos a sobreescribir en ProductoController.

Todo lo que esta dentro de ‘producto’ lo mandamos dentro del parametro guardar(...).



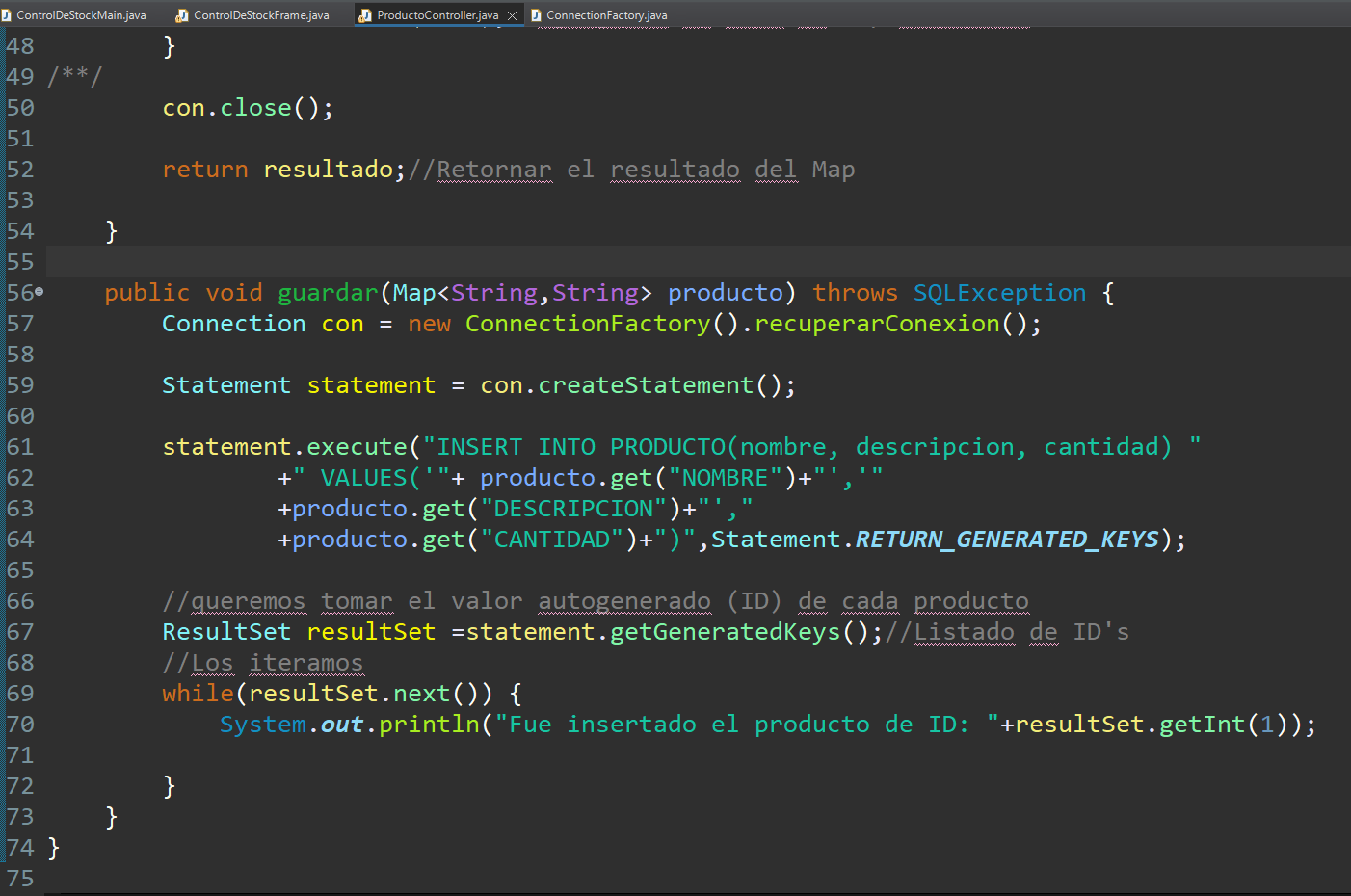
Entonces en la sobreescritura del metodo guardar() en PrpductoController.java vamos a cambiar el Object del parametro por un Map, ya que se esta enviando eso por parametro como vimos recien en la imagen.

Nosotros con esta funcion queremos hacer una conexion con la base de datos, entonces la creamos como hicimos hace un rato. Pero en este caso queremos hacer un statement para agregar nuevos productos a la base de datos. Este statement tendrá que hacer un INSERT INTO. (Recordar usar comillas simples para utilizar String dentro del String de mysql).

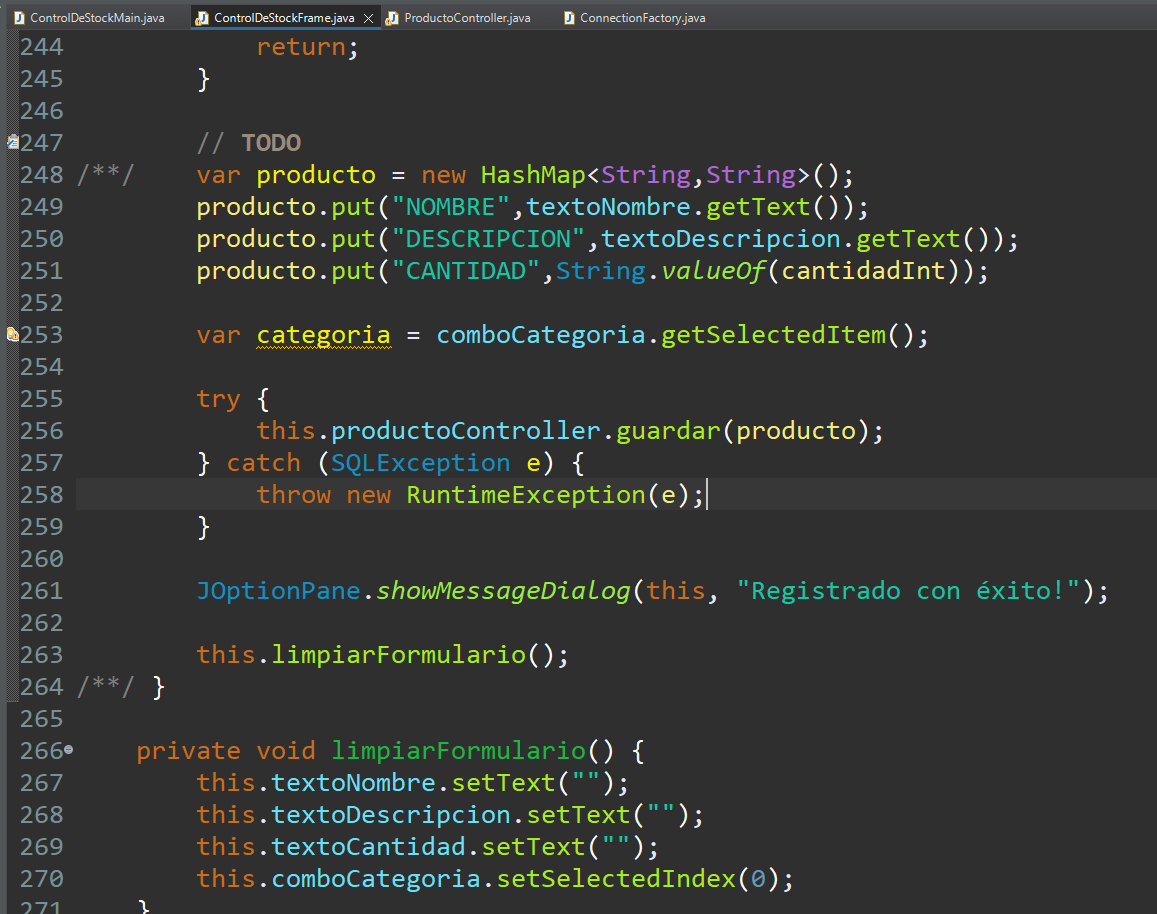
En este punto, no necesitamos que execute() nos diga true o false por si devuelve una tabla o no. Ya sabemos que con INSERT nos dará false. Entonces para que la usamos?

Si le agregamos un segundo parametro en concreto, podrá devolvernos algun valor de la tabla. Statement.RETURN\_GENERATED\_KEYS vamos a utilizarla para tomar el ID.

Gracias a un ResultSet que adentro de este ejecutamos el getGeneratedKeys() para tomar el valor generado dentro. Con un while como hicimos hace un rato, vamos a declarar que queremos utilizar el ID.



Luego hacemos try/catch en donde se implementa guardar para evitar que se trabe el codigo.



Ahora vamos a hacer lo mismo que hicimos con INSERT INTO pero ahora con DELETE.

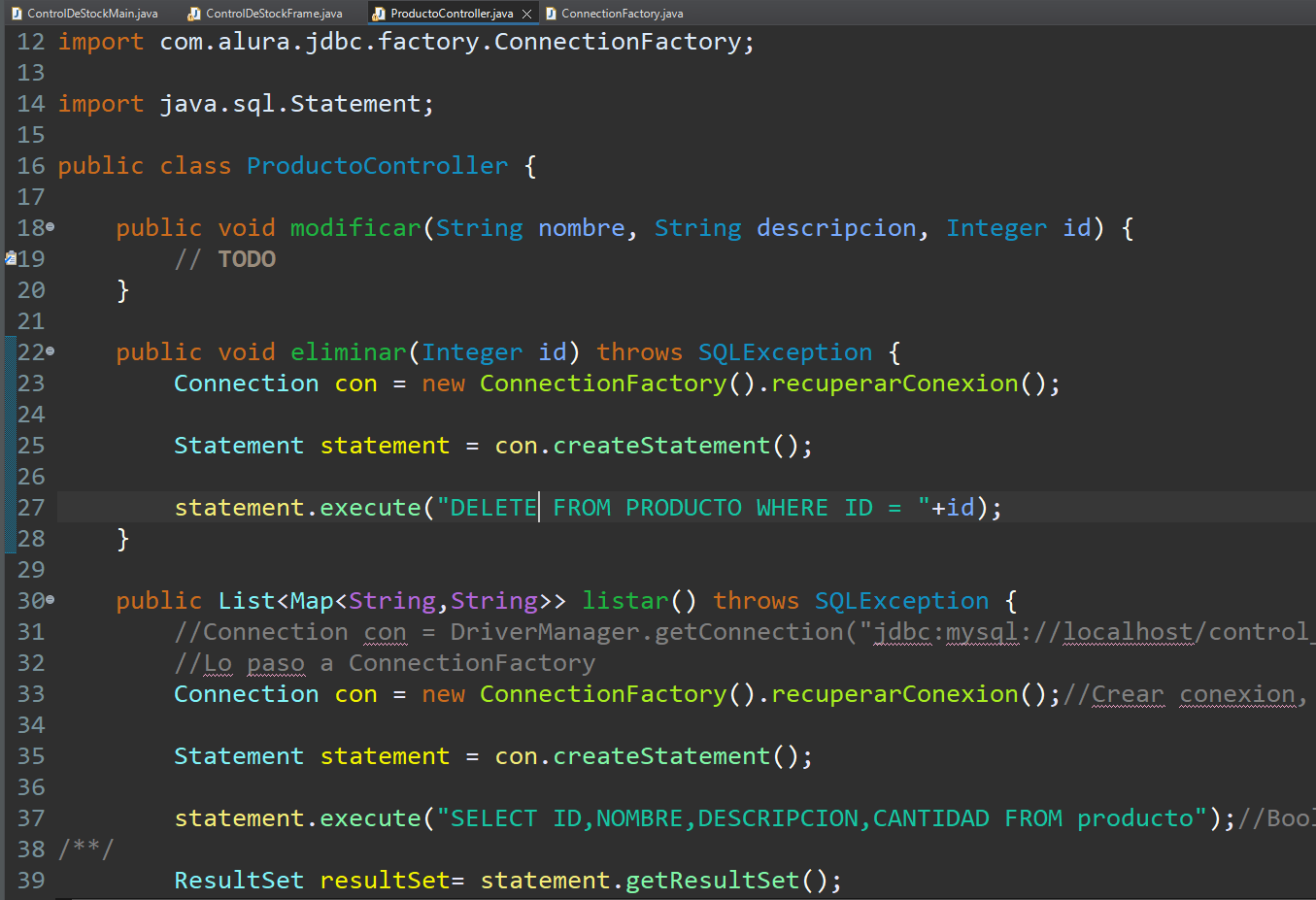
Dentro de configurarAccionesDelFormulario buscamos al boton eliminar en lugar del boton para guardar. Entramos a su metodo eliminar().

En ProductoController.java. En la sobreescritura del meotdo eliminar(),como hicimos con el metodo listar(), puedo iniciar una conexion, hacer el statement, hacer throws a la excepcion y tratarla en ControlDeStockFrame.java en donde se implementa el metodo eliminar(id).

Tratando el Error:



Hasta ahora en la sobreescritura de eliminar() tenemos esto y funcionando:



Nos quedaría pendiente hacer lo mismo con el boton modificar e incluso podriamos mostrar alguna ventana luego de presionar el boton para que nos diga que modificamos tal valor con exito.

Clase 3

Evitando SQL Injection protegiendonos con PreparedStatement

Que pasaría si queremos agregar por nuestra interfaz un nuevo producto, pero que sin querer al escribir el nombre escribimos algo que puede romper el codigo y poner en peligro la base de datos?

Por suerte el JDBC tiene una opcion para validar las informaciones de la query y evitar que eso ocurra.

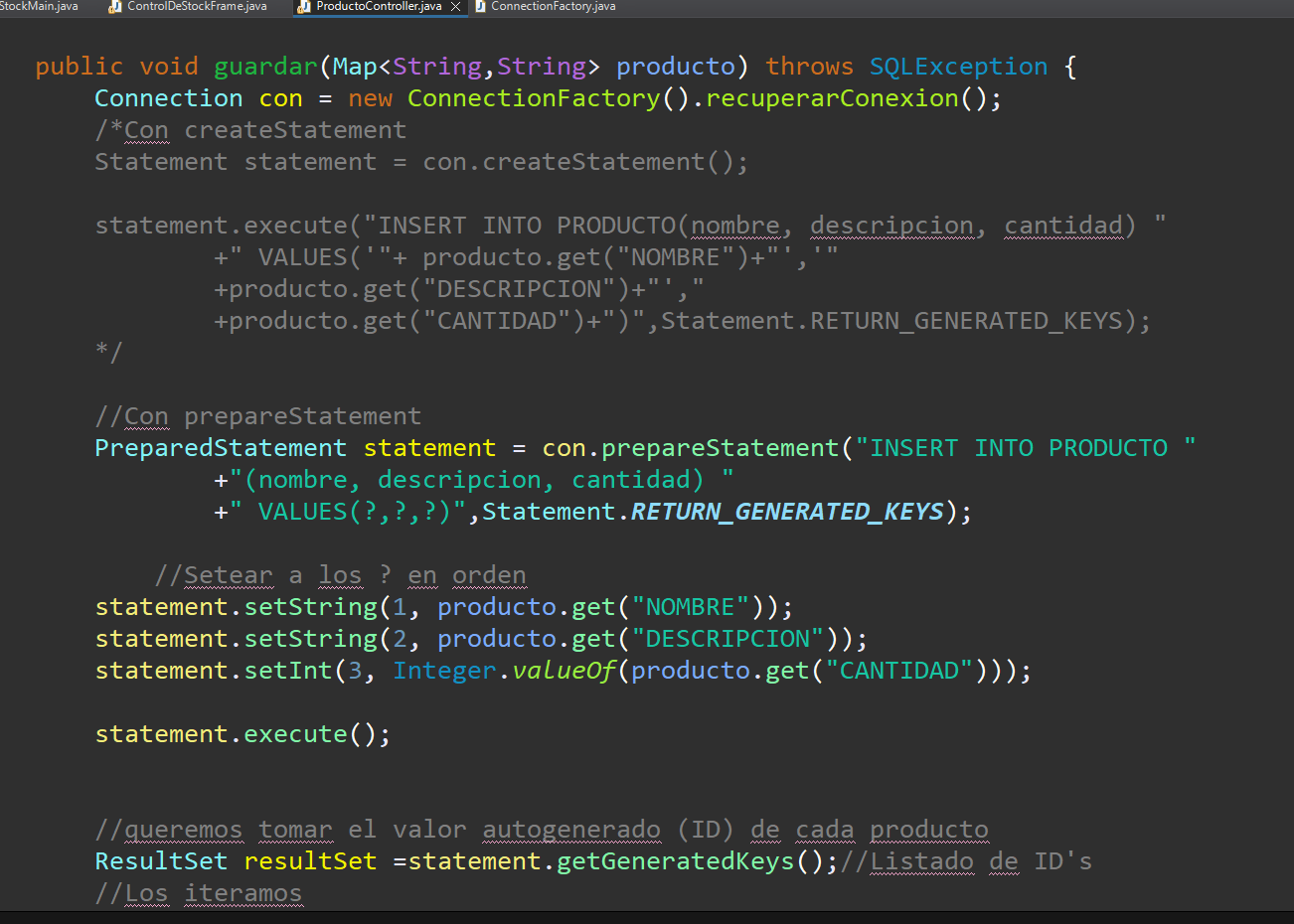
Ahora en lugar de crear un statement como hicimos antes, vamos a “preparar” un statement. Cuando hacemos eso, pasamos la responsabilidad de administrar los parametros del comando SQl para el JDBC.

Entonces en guardar(), tenemos que cambiar el createStatement() por prepareStatement() que retornará un objeto PreparedStatement. Y lo que sería el String con los metodos mysql del execute(), lo colocamos dentro del parametro del createStatement. Las VALUES tienen que ser signos de pregunta ‘?’.

El RETURN\_GENERATED\_KEYS tambien lo ponemos como segundo parametro del prepareStatement(). El execute() queda vacío.

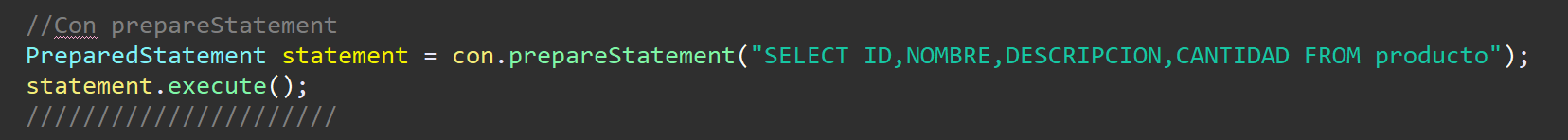
Para setear los valores que serían los ‘?’ en orden, utilizamos el metodo setString(orden, llave).Por ejemplo statement.setString(1, producto.get(“NOMBRE”));

Todo esto que hicimos hasta ahora, fue para que lo que asignemos en los inputs (caracteres especiales y metodos mysql), sea tomado como parte texto. El prepareStatement() se encarga de normalizar la String y que eso sea un texto comun y corriente. Con esto evitamos riesgos de ataques llamados SQL injections. Ademas, mejoramos la legibilidad del codigo.

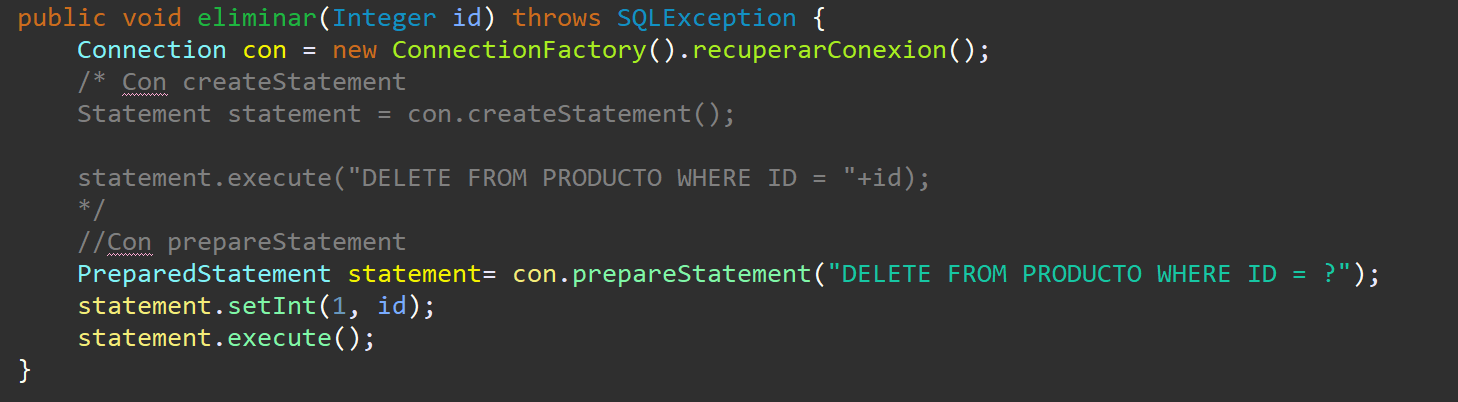


Tambien deberia hacer lo mismo para los otros metodos.

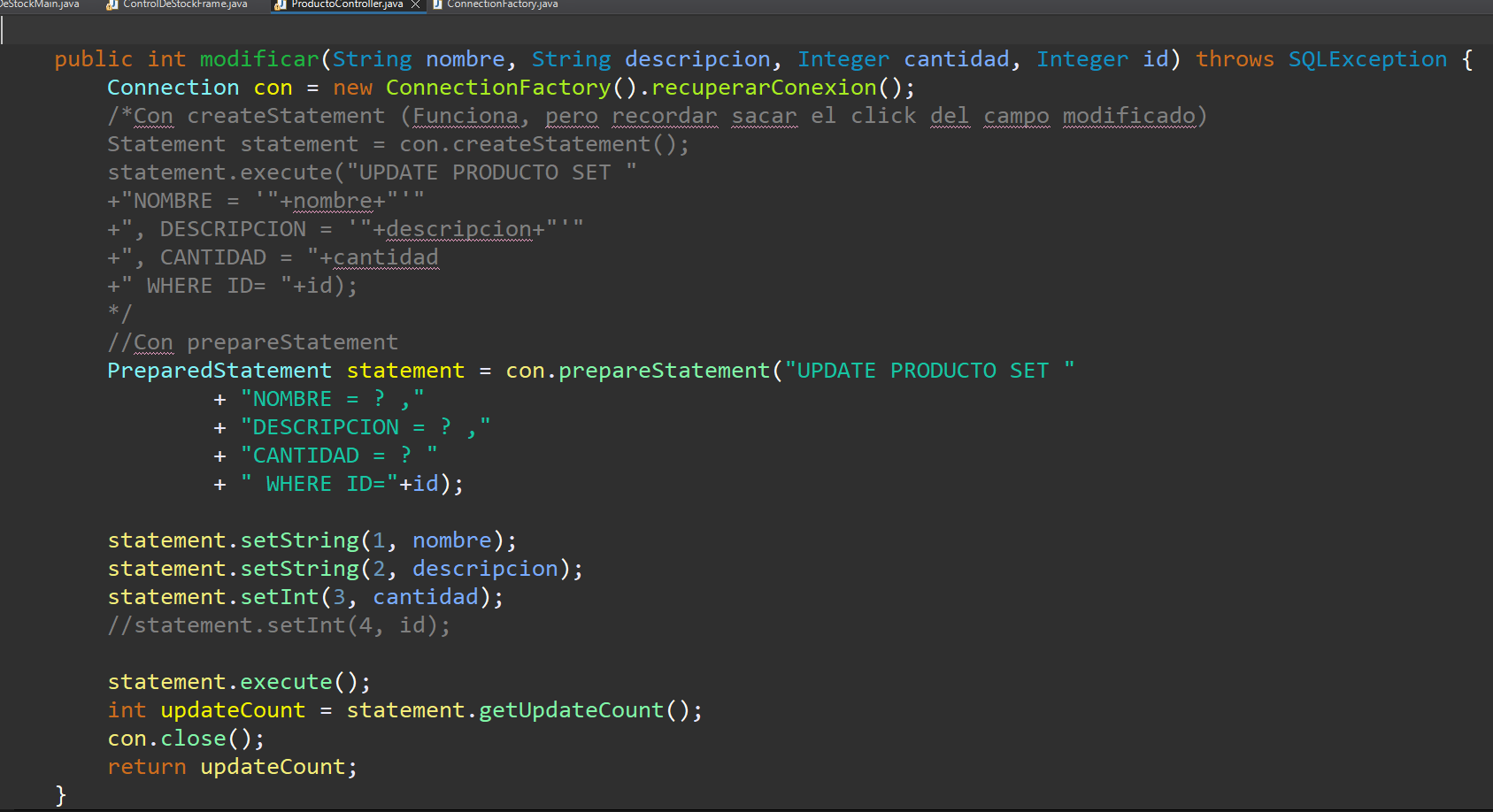
Con listar():



Con eliminar():



Por ultimo. Con modificar():



NOTA:

Cual es el riesgo de usar Statement en lugar de PreparedStatement?

El Statement no mantiene una version de la query compilada en la base de datos.

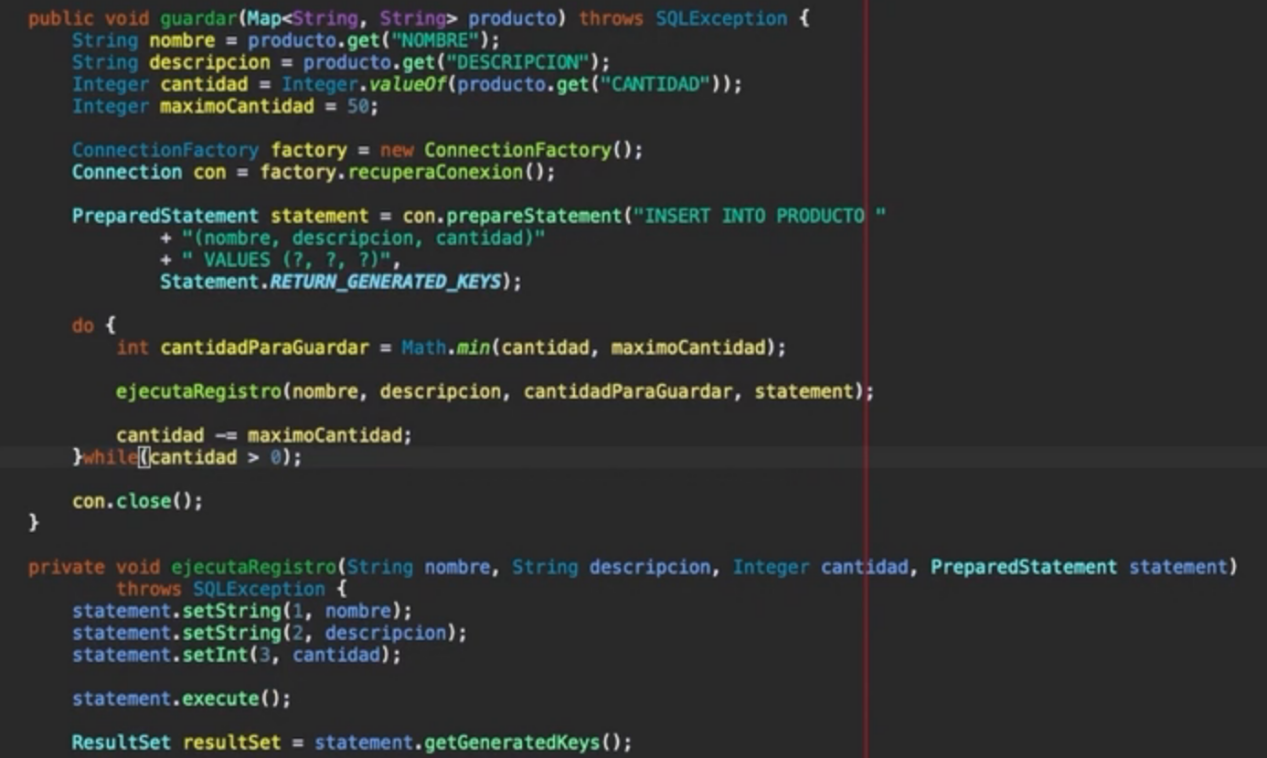
En cambio el PreparedStatement SI la mantiene y de forma parametrizada. Así el usuario puede ejecutar la misma consulta diversas veces con parametros distintos.

Como vimos en el MySQL, nuestro programa tiene la limitacion de solo definir hasta 50 cantidades por registro.

Queremos entonces que si ingresamos un numero mayor a 50, que automaticamente guarde en un registro los 50 y en otro el numero que sobre. Así no se pierda la cantidad puesta de mas.

Ademas, vamos a acomodar la accion de modificar los registros dentro de un metodo que llamaremos ejecutaRegistro() que utilizaremos dentro del boton guardar.

El do while es fundamental para dividir los registros segun el maximo de cantidad permitida. Permite dividir en varios registros cuando se supera el 50.



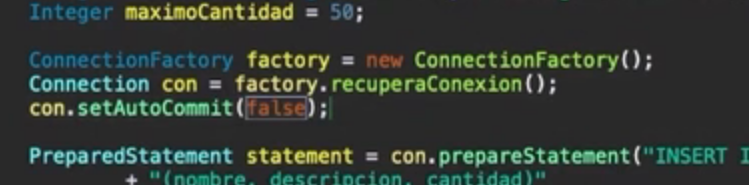
Como vimos, la peticion se divide en varios registros si la cantidad se divide por 50.

Pero que pasa si ocurre un error en el medio? Seguro algunas cantidades se perderían.

Ya que lo que se esta haciendo, son varias transacciones si supera las 50 cantidades. En este punto se hace una transaccion por cada registro. Si alguna de esas transacciones falla por alguna razon, no se terminarían de pasar todas las cantidades, haciendo que se pierdan. Y no queremos eso. Queremos que se complete todo exitosamente, y que si no se completa, que deshaga la transaccion a como estaba antes, asi el receptor ni el emisor se vean afectados.

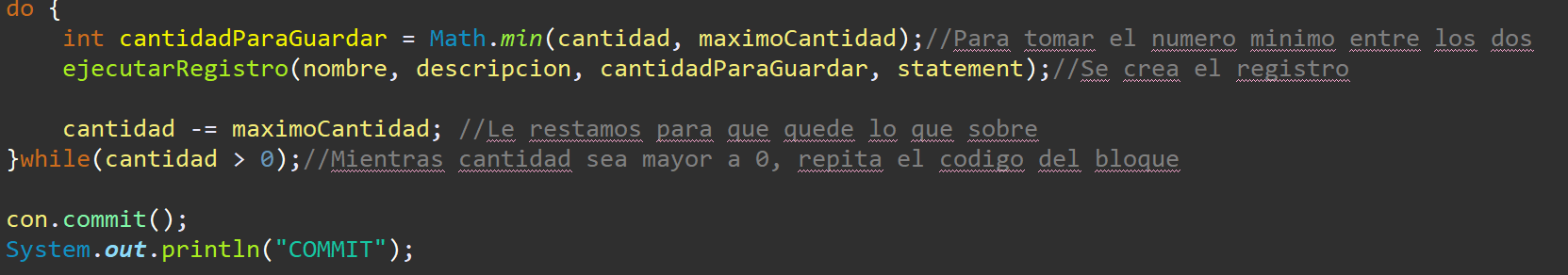
Para evitar el error recien explicado, podemos quitar la responsabilidad del JDBC y hacerla nosotros. Así intentamos que se haga una sola transaccion entera, es decir, lograr que se guarde el 100% de la peticion o nada. Así no se puedan perder alguna cantidad.

Vamos a hacer un setAutoCommit(false) para que la conexion no tega el control de la transaccion, sino que nosotros tengamos el control. Así nosotros decidimos cuando el commit será realizado.



Ahora el problema es que al quitarle el control al JDBC para completar la transaccion, los nuevos productos no se guardan en el listado, no se registran.

Entonces manualmente vamos a declarar que se haga el commit luego de que termine el loop while. Es decir, que la transaccion termine solo cuando finalice de enviar los registros necesarios segun la cantidad de unidades.

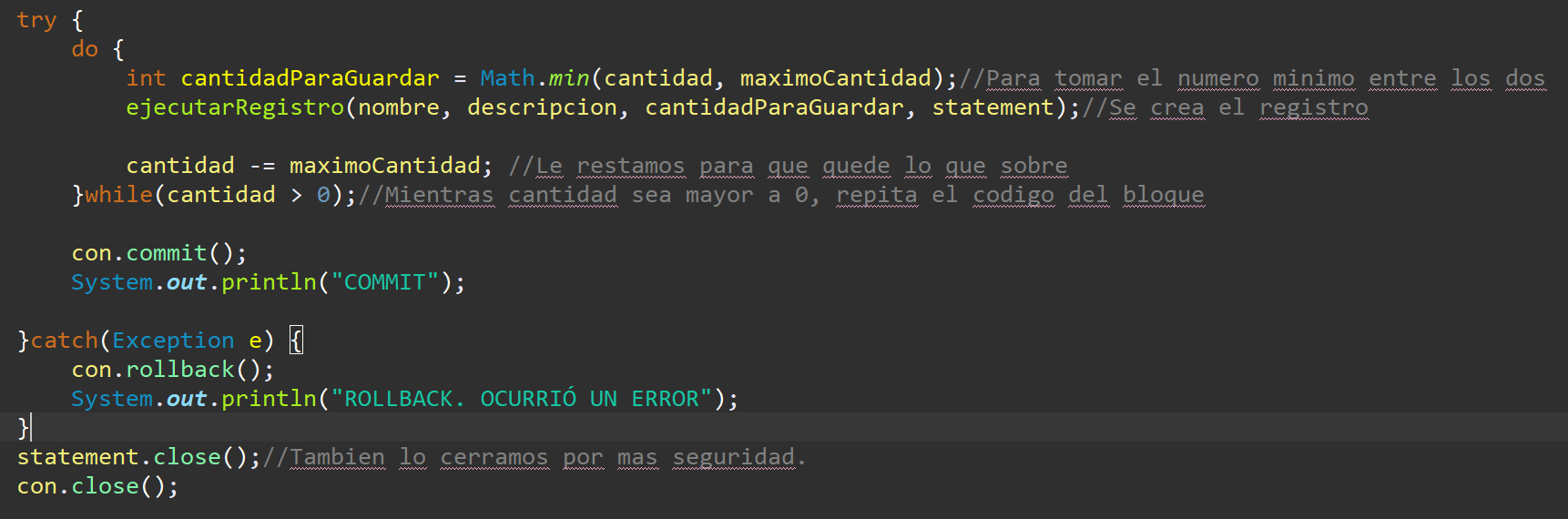


NOTA: La diferencia con antes, es que de manera automatica, la JDBC terminaba cada transaccion cuando terminaba de crear un registro (es decir, por cada 50u). Nosotros queriamos que termine cada transaccion cuando se terminen de mandar todos los registros necesarios del mismo producto.

De todas formas, para garantizar de que en el medio de la gran transaccion que hicimos no se pierdan cantidades por otros errores que puedan ocurrir, vamos a hacer rollback. Que cancelará y deshará la transaccion que se estaba produciendo, haciendo que quede como antes de que esta ocurra.

Encerramos el do while en un try y en el catch tomamos la exception y le hacemos un rollback() a la conexion. El rollback() como dice el nombre, deshace todos los cambios hechos en la transacion. Solo se puede usar cuando el auto commit esta false.

Entonces. Si sale todo bien, tenemos un commit. Sino, tenemos un rollback de la transaccion. Y asi no se corre el peligro de que la informacion transladada en la transaccion no se pierda.



NOTA: Recordar. Al decidir que usted quiera manejar las transacciones de la aplicacion en lugar de JDBC, ademas de setear el auto-commit en false, hay que explicitar el commit y el rollback de la conexion. Así si la transaccion es exitosa, se realiza el commit, sino el rollback para que no haya problemas.

Hasta aquí aprendimos que una transaccion involucra un conjunto de operaciones. Como en nuestro caso, para guardar una cierta cantidad de registros para un producto. Tambien vimos que la conexion que abrimos, la tenemos que cerrar luego de utilizarla. Tambien habría que cerrar los preparedStatement y los resultSet. Con eso agilizamos el rendimiento al no tener cosas abiertas sin necesidad.

Utilizando el try-with-resources.

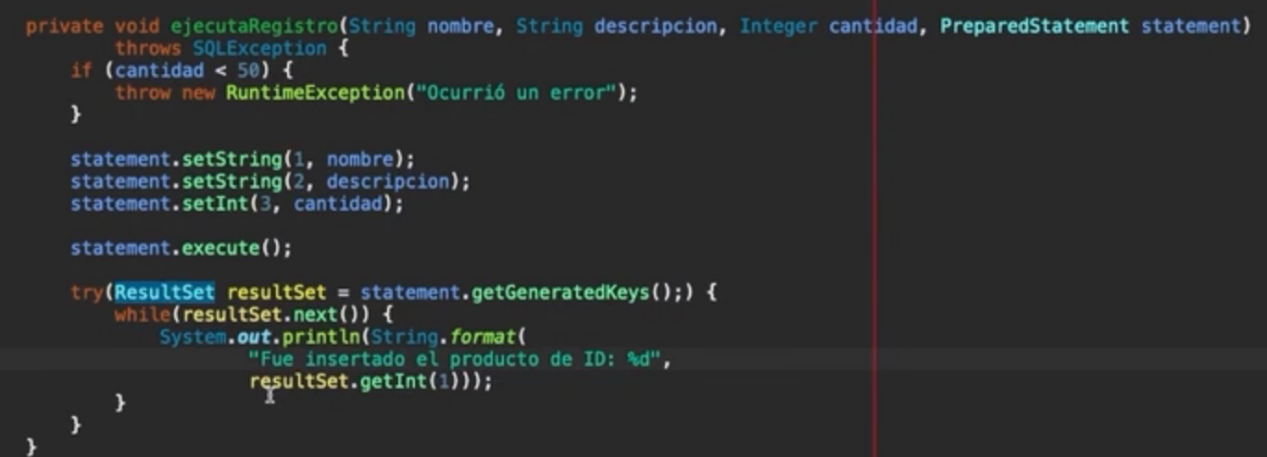
El Try With Resources nos permite declarar recursos que van a ser utilizados en un bloque de try catch con la certeza de que estos recursos van a ser cerrados o finalizados automáticamente después de la ejecución del bloque. Entonces si utilizamos el TWR no va a ser necesario ejecutar los close().

Entonces si nos olvidamos de cerrar las conexiones o si ocurren errores inesperados antes de que se cierren, el try-with-resources nos salvará de un apuro.

Ya que si permanecen todas las cosas abiertas, generará un problema de rendimiento en la aplicacion.

Un requisito para utilizarlo, es implementar la interfaz AutoCloseable.

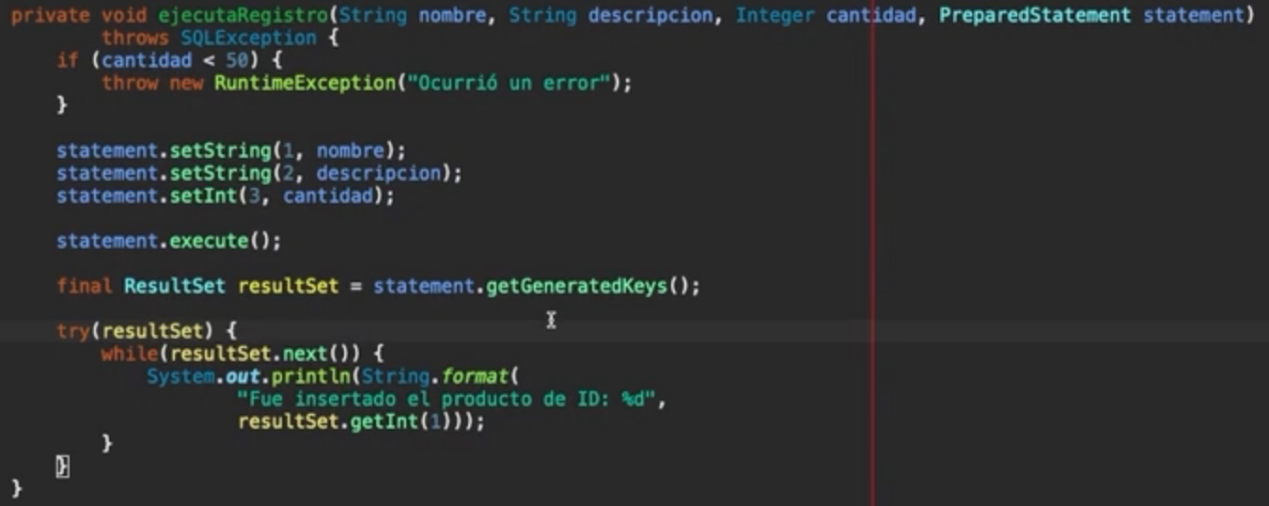
Con la version antigua de utilizar try-with-resources.



Podemos ver que a diferencia de antes, ya no es necesario el metodo close porque la propia JVM se encarga de cerrarla.

NOTA: Tomemos en cuenta que la clase o interfaz que se esta utilizando en el TWR es ResultSet e implementa la interfaz AutoCloseable. Por eso eclipse no se queja.

Con la version nueva de utilizar try-with-resources.

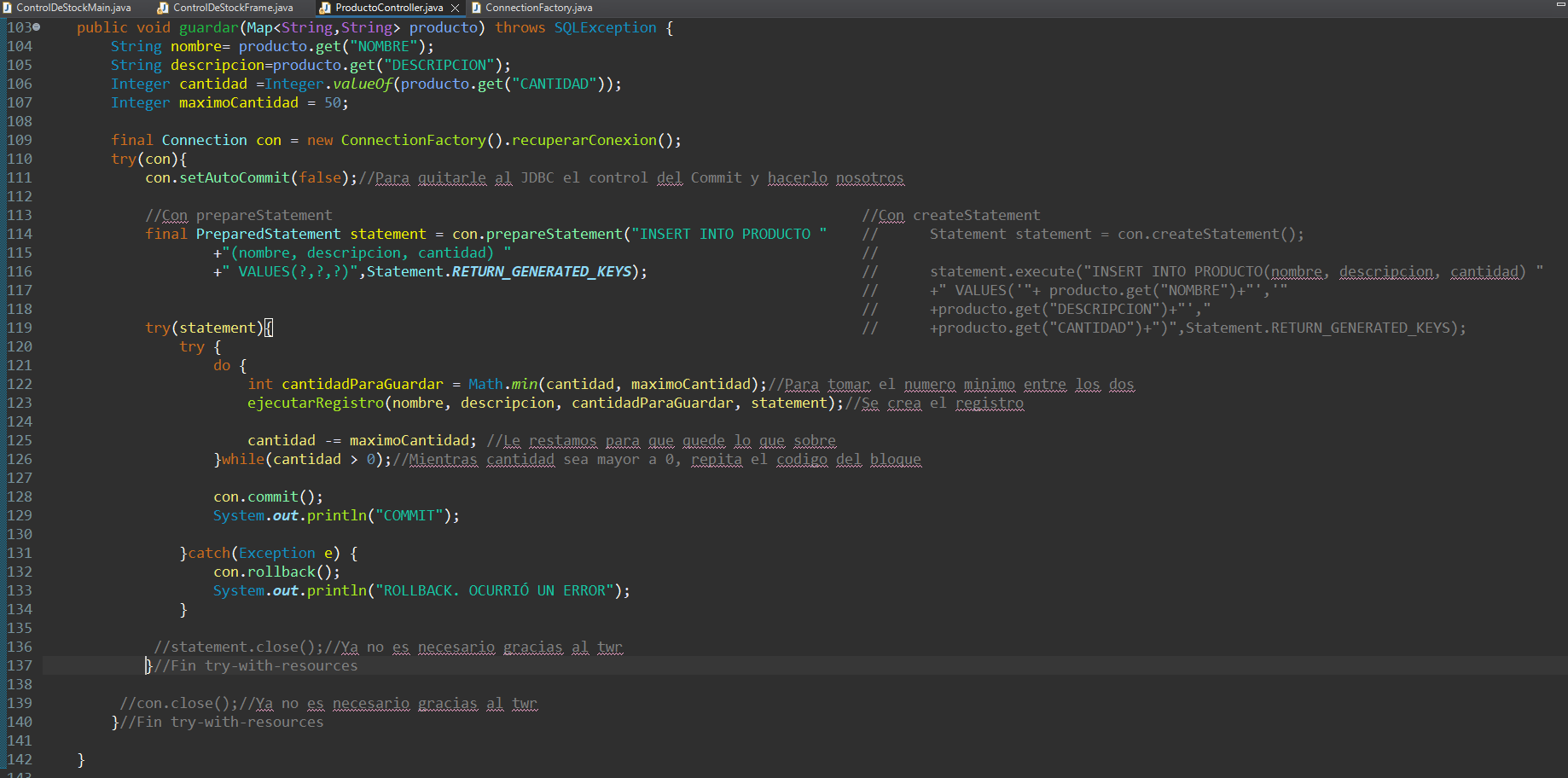


A diferencia de la version vieja, vemos que lo del parametro del try esta arriba en el ‘final’. Que luego si es pasado por parametro en el try. Esto hace que el codigo sea mas legible.

Basicamente el try envuelve toda la transaccion que queramos hacer. Y dentro del parametro del try va lo que queremos abrir. Al terminar el bloque de ese try, automaticamente se cierra habiendo error o no.

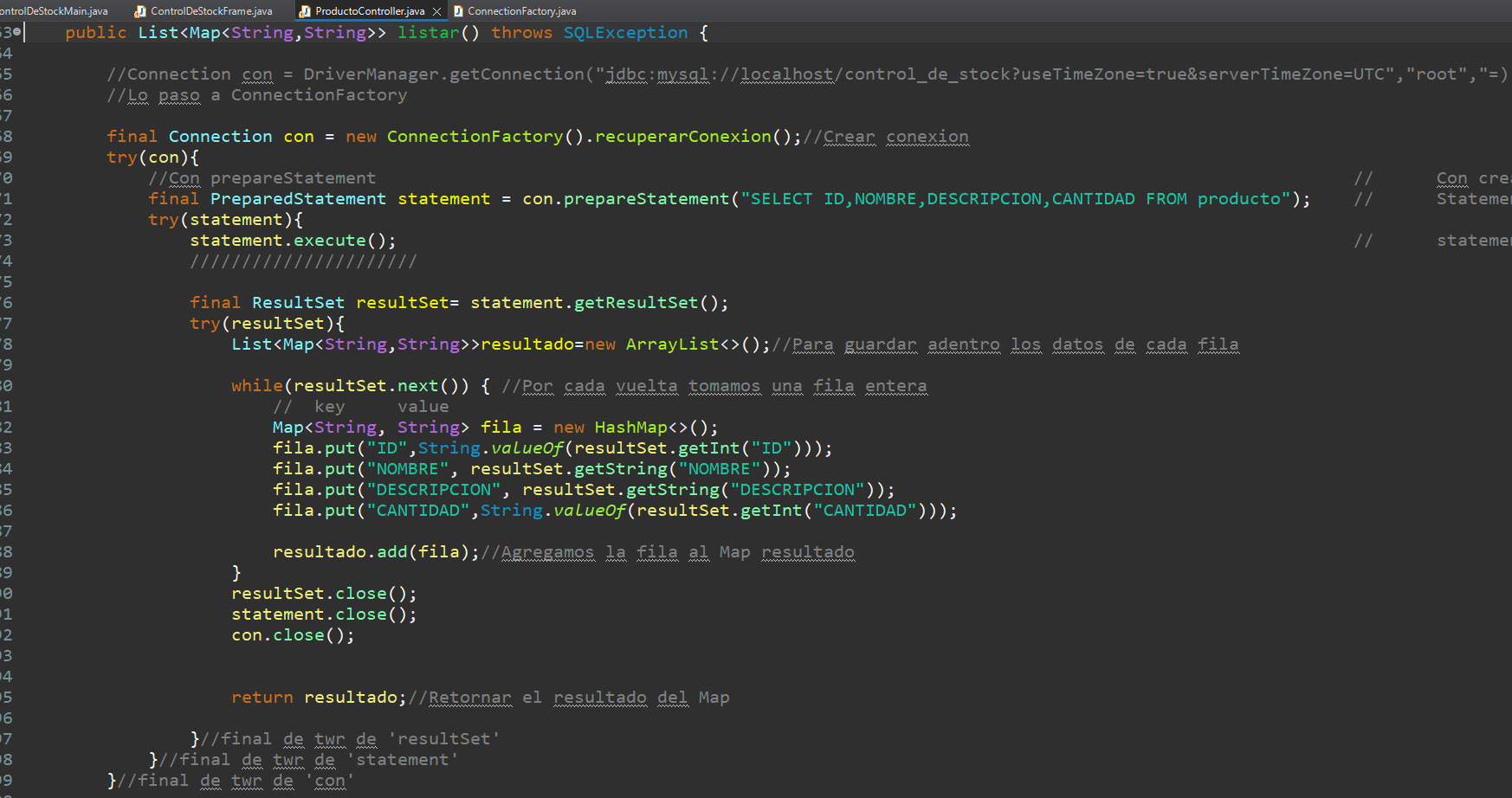
Esto del try-with-resources lo podemos aplicar en cada lugar de nuestro codigo en donde podriamos usar close. Recien lo hicimos con ejecutarRegistro().

Ahora con la conexion y statement del guardar():

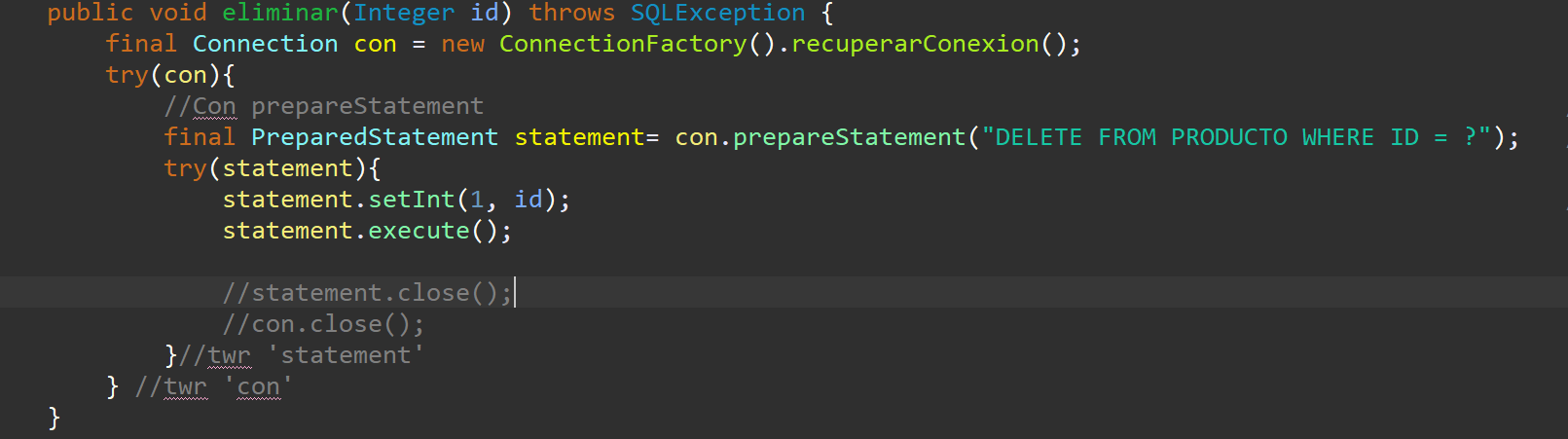


NOTA: Detalle opcional. Podriamos borrar el try simple que hay dentro del try(statement), ya que podemos usar el mismo try(statement) para luego hacer el catch del rollback.

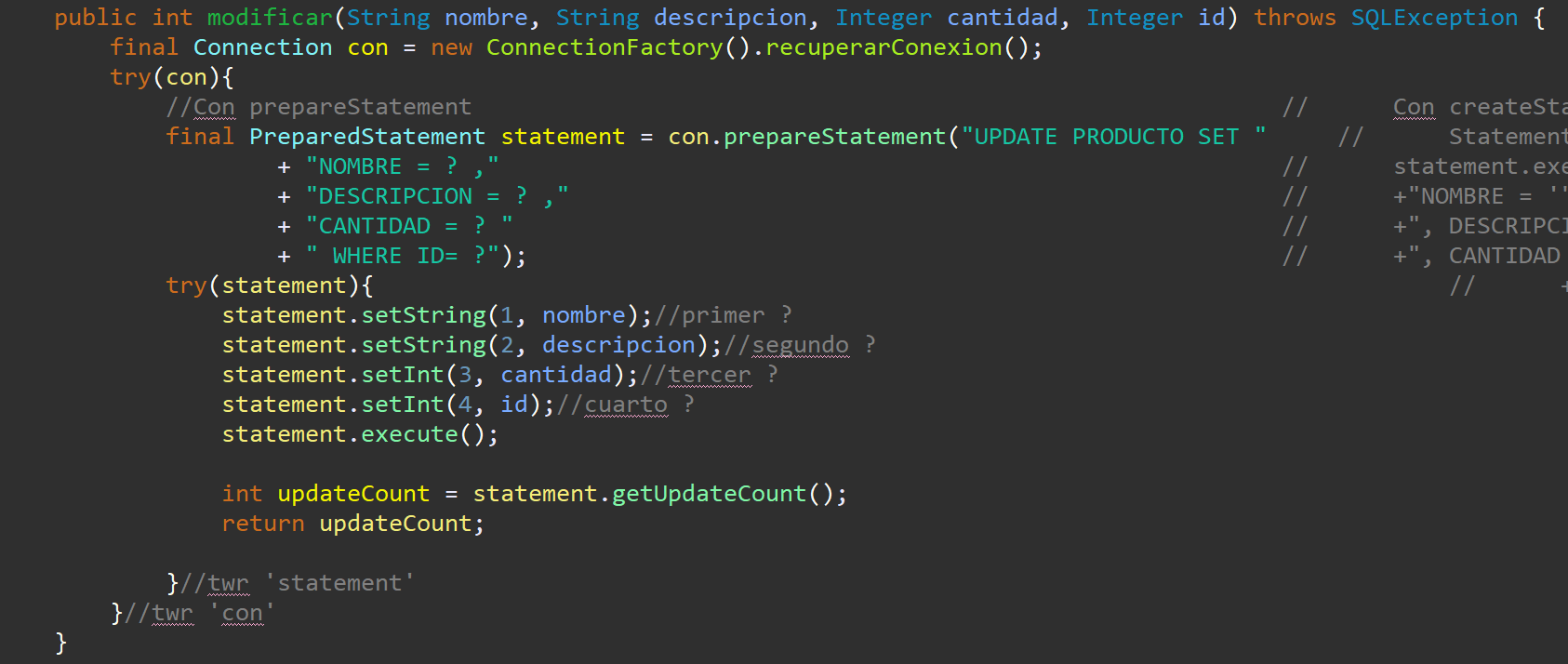
Ahora con listar():



Ahora con eliminar():



Ahora con modificar():



Listo. Ya no nos tenemos que preocupar tanto por cerrar las cosas. El try-with-resources lo hará por nosotros ya que en el mismo abrimos recursos como PreparedStatement, Connection y ResultSet. Esto es porque esos recursos que abrimos implementan la interfaz AutoCloseable. Recordar que ese es requisito fundamental (todo lo que queremos abrir en el twr tiene que implementar AutoCloseable para cerrarlo automaticamente).

CLASE 4

Recapitulando. Nosotros tenemos un programa en Java que queremos conectar con una base de datos en MySQL. Pero no es manera directa, sino que hay un Driver MySQL que será el interlocutor entre los dos.

Pero si nosotros tenemos que cambiar a otra base de datos distinta, como SQL Server, tenemos que utilizar su driver tambien. Y así con cada uno.

Entonces para facilitar el trabajo y poder conectarnos a cualquier tipo de base de datos sin importar el driver, utilizamos la interfaz JDBC que hace ese trabajo por nosotros.

Tambien, hemos centralizado la creacion de la conexion a la base de datos gracias al ConnectionFactory y así hacer el codigo mas simple.

Si nosotros abrimos y cerramos una conexion por accion, que pasaría si en una pagina varios usuarios abren la misma conexion? Se hace en cola? Claro que no. Nuestra app debe estar disponible para que varias personas la puedan utilizar al mismo tiempo en paralelo. Pero esto tiene un limite.

Una buena alternativa es disponer una cantidad minima y maxima de conexiones abiertas. Esta implementacion se va a conectar con el JDBC para abrir la cantidad establecida de conexiones.

Pool de conexiones. Se va a conectar con el JDBC para mantener una cantidad minima y maxima de conexiones abiertas. Así no se hace una cola muy grande y que tampoco se ahogue la aplicacion.

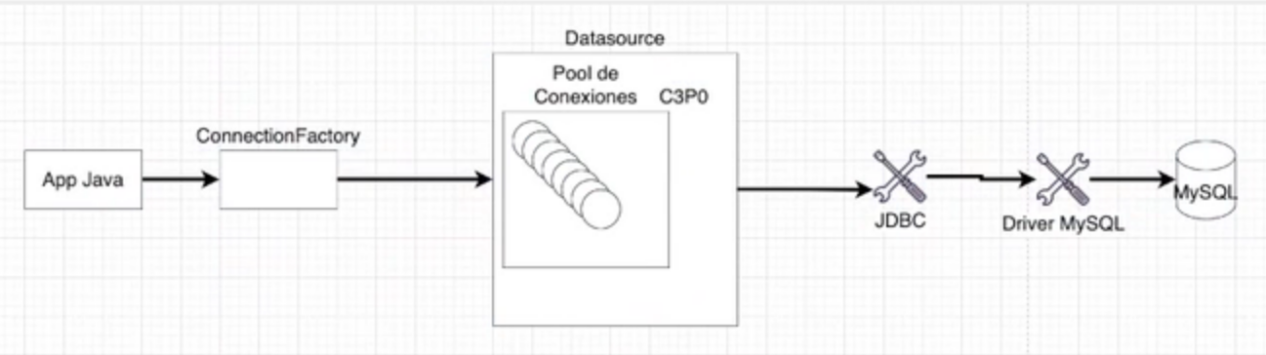
Al igual que los drivers de bases de datos, las Pool tambien tienen funciones para manejarlo, como por ejemplo C3P0.

Para utilizarlo vamos a aprovechar la JDBC para usar una interfaz llamada dataSource. Esta va a abstraer la implementacion del pool de conexiones para nosotros.

Con esto, la ConnectionFactory que tenemos no va a tener mas la responsabilidad de crear una conexion. Lo que va a hacer ahora es solicitar una conexion abierta para el dataSource. Es decir, la ConnenctionFactory en lugar de decir “JBDC, dame una conexion”, va a decir “dataSource, dame una conexion que tienen ahí en tu pool de conexiones y yo sigo acá con mi trabajo”.

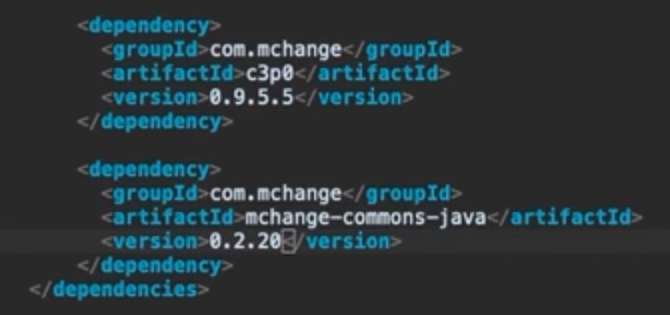
El dataSources va a ser el que maneja si dar una conexion.

Esquema:



Implementar pool de conexiones para que nuestra aplicacion tenga control de las mismas.

Primero vamos a agregar al pom.xml el driver C3P0 y tambien el commons-java.



C3P0 nos va a dar la posibilidad de configurar un pool de conexiones.

El commons-java nos va a dar la posibilidad de agregar mas detalles del dataSource por el log de la consola de la aplicacion.

Ahora vamos a modificar connectionFactory para la configuracion del pool de conexiones.

La idea es que la ConnectionFactory utilice las conexiones desde el pool, en lugar de crearlas por ella misma como estabamos haciendo antes.

Vamos a crear un constructor donde se instancia ComboPooledDataSource() que es la clase del c3p0. En esta instancia del pool vamos a configurar la URL,usuario y contraseña de la base de datos.

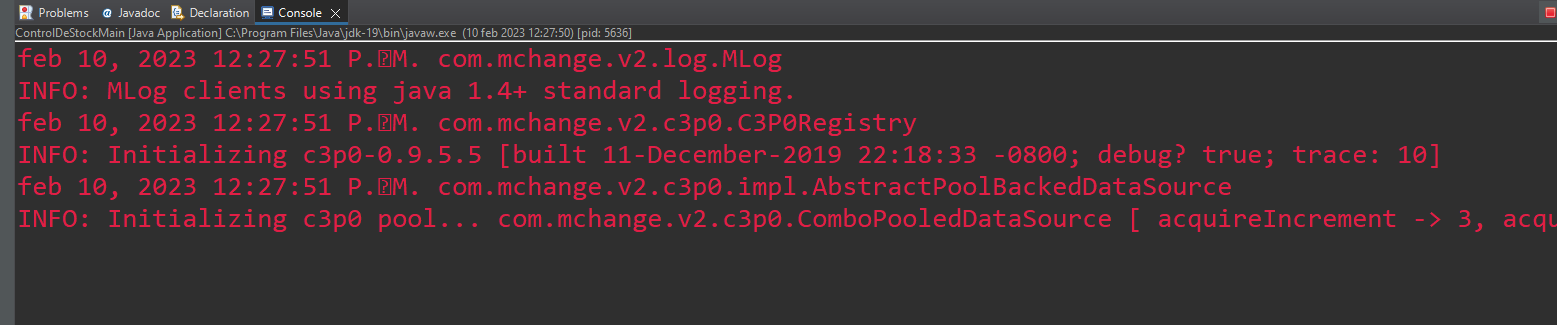
Todo el pooledDataSource lo guardamos en una variable para poder utilizarlo mas legiblemente en las distintas funciones donde queramos aplicarlo, como en recuperarConexion().

En recuperarConexion(), en lugar de crear la conexion con el JDBC, vamos a para pedirle al pool de dataSource.



Entonces en recuperarConexion, en lugar de crear una conexion con el DriverManager, estamos tomando la conexion desde el pool para realizar las operaciones en la base de datos.

Si ejecutamos. Gracias a las dependencias que agregamos, podemos ver la informacion de la instanciacion del pool de conexiones.

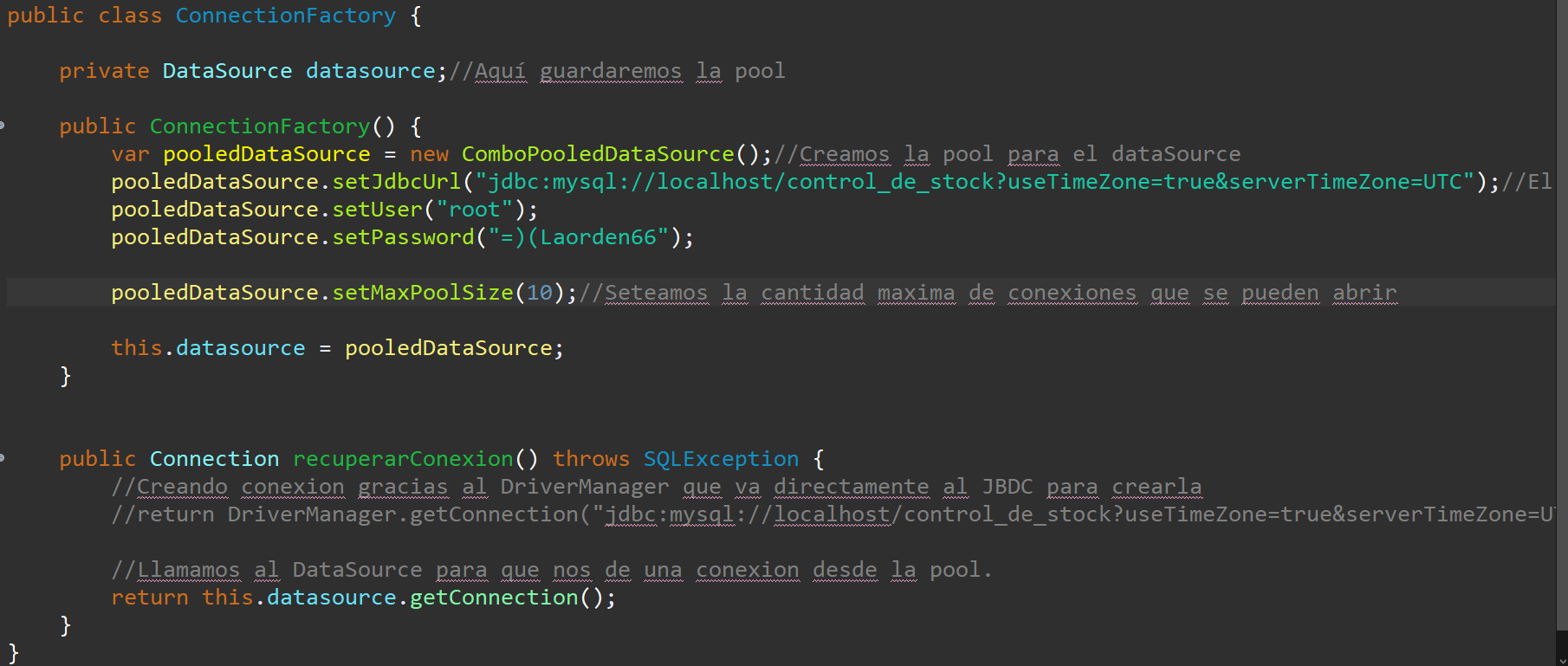


NOTA: En un escenario donde multiples clientes pueden acceder a una misma aplicacion simultaneamente Cual sería el mejor abordaje?

Reutilizar un conjunto de conexiones de tamaño fijo o dinamico. Esta es la estrategia de mantener un pool de conexiones. Vamos a abrir una cantidad especifica de conexiones y reutilizarlas.

En este punto no lo determinamos todavia. En las siguientes explicaciones veremos el tema de determinar esa cantidad de conexiones.

Determinar cantidad maxima de conexiones que el pool puede mantener. Lo haremos con el metodo setMaxPoolSize(...). El numero que ingresamos dentro, va a ser la cantidad maxima de conexiones que podemos tener abiertas.



NOTA: En el codigo que hicimos, cada vez que ejecutamos el recuperaConexion(), se abre una conexion gracias al datasource que nos da una de la pool. Entonces si el recuperaConexion() se abre mas veces que el numero que hay en el setMaxPoolSize(), el programa esperará a que una conexion quede disponible para usarla.

Si llegamos al tope de conexiones, el proximo pedido tendrá que esperar un momento hasta que haya una conexion libre.

NOTA: Para ver la lista de conexiones abiertas, podemos usar el comando de MySQL ‘SHOW PROCESSLIST’.

Para cerrarla podemos usar ‘KILL [Id\_de\_la\_conexion]’.

Solo puedes cerrar conexiones que tú mismo has abierto o que tienes permisos para administrar. Además, es importante tener en cuenta que cerrar una conexión puede interrumpir una operación en curso y puede tener efectos secundarios indeseados en la aplicación que está usando la conexión. Por lo tanto, debes usar esta funcionalidad con cuidado y solo en casos necesarios.

CLASE 5

En nuestro codigo, a nuestros productos les estamos asignandoles asignandoles variables sueltas recibidas por parametro. Mejor vamos a crear una forma de representar el producto directamente en el proyecto en Java. Es decir, vamos a crear un modelo.

((((((((((((((((((DESPUES DEL VIDEO 5.3)))))))))))))))))

Desafío: Operaciones de modificación y exclusión en el ProductoDAO

Utilizando todo lo que aprendimos en la refactorización de la lógica de acceso a la capa de datos del ProductoController para la clase ProductoDAO es el momento de poner en práctica lo que aprendimos para refactorizar las operaciones de modificación y exclusión de productos siguiendo las buenas prácticas que aprendimos.

Empecemos por el método de eliminar de la clase ProductoController. Lo que vamos hacer primero es crear una llamada a productoDao.eliminar(id);. Este método aún no existe, entonces vamos a crearlo en el ProductoDAO y mover la lógica de exclusión del producto para allá. Vamos a dejarla más sencilla también, removiendo pedazos de código que ya no son más necesarios, como la de tomar una conexión.

Recordemos que no debemos más estar lanzando la SQLException para las demás capas. Los errores que una clase puede lanzar deben ser tratados en ella de forma que podamos lanzar, como máximo, una excepción del tipo unchecked.

Por último vamos a ajustar el código referente a la exclusión de un producto en la clase ControlDeStockFrame y listo. Podemos hacer lo mismo con el método de modificación.

Así va a quedar el resultado:

// ControlDeStockFrame**private** **void** **modificar**() {

**if** (tieneFilaElegida()) {

JOptionPane.showMessageDialog(this, "Por favor, elije un item");

**return**;

}

Optional.ofNullable(modelo.getValueAt(tabla.getSelectedRow(), tabla.getSelectedColumn()))

.ifPresentOrElse(fila -> {

**Integer** id = Integer.valueOf(modelo.getValueAt(tabla.getSelectedRow(), 0).toString());

**String** nombre = (String) modelo.getValueAt(tabla.getSelectedRow(), 1);

**String** descripcion = (String) modelo.getValueAt(tabla.getSelectedRow(), 2);

**Integer** cantidad = Integer.valueOf(modelo.getValueAt(tabla.getSelectedRow(), 3).toString());

**var** filasModificadas = this.productoController.modificar(nombre, descripcion, cantidad, id);

JOptionPane.showMessageDialog(this, String.format("%d item modificado con éxito!", filasModificadas));

}, () -> JOptionPane.showMessageDialog(this, "Por favor, elije un item"));

}

**private** **void** **eliminar**() {

**if** (tieneFilaElegida()) {

JOptionPane.showMessageDialog(this, "Por favor, elije un item");

**return**;

}

Optional.ofNullable(modelo.getValueAt(tabla.getSelectedRow(), tabla.getSelectedColumn()))

.ifPresentOrElse(fila -> {

**Integer** id = Integer.valueOf(modelo.getValueAt(tabla.getSelectedRow(), 0).toString());

**var** filasModificadas = this.productoController.eliminar(id);

modelo.removeRow(tabla.getSelectedRow());

JOptionPane.showMessageDialog(this,

String.format("%d item eliminado con éxito!", filasModificadas));

}, () -> JOptionPane.showMessageDialog(this, "Por favor, elije un item"));

}

// ProductoController**public** **int** **modificar**(String nombre, String descripcion, Integer cantidad, Integer id) {

**return** productoDao.modificar(nombre, descripcion, cantidad, id);

}

**public** **int** **eliminar**(Integer id) {

**return** productoDao.eliminar(id);

}

// ProductoDAO**public** **int** **eliminar**(Integer id) {

**try** {

**final** **PreparedStatement** statement = con.prepareStatement("DELETE FROM PRODUCTO WHERE ID = ?");

**try** (statement) {

statement.setInt(1, id);

statement.execute();

**int** updateCount = statement.getUpdateCount();

**return** updateCount;

}

} **catch** (SQLException e) {

**throw** **new** **RuntimeException**(e);

}

}

**public** **int** **modificar**(String nombre, String descripcion, Integer cantidad, Integer id) {

**try** {

**final** **PreparedStatement** statement = con.prepareStatement(

"UPDATE PRODUCTO SET "

+ " NOMBRE = ?, "

+ " DESCRIPCION = ?,"

+ " CANTIDAD = ?"

+ " WHERE ID = ?");

**try** (statement) {

statement.setString(1, nombre);

statement.setString(2, descripcion);

statement.setInt(3, cantidad);

statement.setInt(4, id);

statement.execute();

**int** updateCount = statement.getUpdateCount();

**return** updateCount;

}

} **catch** (SQLException e) {

**throw** **new** **RuntimeException**(e);

}

}

((((((((((((((((((((((DESPUES DEL VIDEO 6.4))))))))))))))))))))))))))))))))

¿Cuál es el problema de la aplicación tener queries N + 1?

Porque son utilizadas múltiples queries, aumentando la cantidad de acceso a la base de datos y, por consecuencia, empeorando la performance de la aplicación y del propio sistema de base de datos.

Cuando las consultas son sencillas no hay problemas. Pero cuanto más complejidad van teniendo nuestras consultas hay la necesidad de buscar más informaciones de múltiples tablas, aumentando el acceso exponencialmente. Eso impacta gravemente la performance de la aplicación y del sistema de base de datos.

((((((((((((((((((((((((((((DESPUES DEL VIDEO 6.6)))))))))))))))))))))))))))))))

José creó una relación entre dos tablas utilizando la clave foránea. Pero ahora tiene una duda de como hacer para buscar las informaciones relacionadas de las dos tablas en su aplicación. ¿Qué podemos decir a José que haga para relacionar las dos tablas en una sola búsqueda?

José debe utilizar el INNER JOIN.

Con el INNER JOIN José va a lograr buscar las informaciones que están relacionadas entre las dos tablas.