JDBC CRUD

Contents

Overview

Java Database Connectivity (JDBC) is an application programming interface (API) that allows us to connect a database with our Java application.
It uses drivers to connect with a database, allowing us to execute queries in Java.

Tutorial

To access data within the database we need to do four things; open the connection, create/prepare a statement, execute the statement, and finally close the connection.
All of this should be done in a try/catch/finally or a try-with-resources block.

Opening The Connection

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import org.apache.log4j.Logger;

public class LearningJDBC {

    public static final Logger LOGGER = Logger.getLogger(LearningJDBC.class);

    private String jdbcConnectionURL;
    private String username;
    private String password;

    public LearningJDBC(String username, String password) {
        jdbcConnectionURL = "jdbc:mysql://localhost:3306/ims";
        this.username = username;
        this.password = password;
    }

    public LearningJDBC(String jdbcConnectionURL, String username, String
password) {
        this.jdbcConnectionURL = jdbcConnectionURL;
        this.username = username;
        this.password = password;
    }

    public void readAll() {
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(jdbcConnectionURL, username,
password);
        } catch (SQLException e) {
            LOGGER.debug(e.getStackTrace());
        } finally {
            try {
                if(conn != null) {
                    conn.close();
                }
            } catch (SQLException se) {
                se.printStackTrace();
            }
        }
    }
}
```

In the above example, we have a class called LearningJDBC that has two constructors and four class level variables.
In the readAll method, we are using a try-catch block to open a connection.
To open the connection, we declare a variable of type `Connection`, and assign it the value of `DriverManager.getConnection`, passing in the URL, username, and password as parameters.
Within the try block, we would be able to then run whatever code we want to run against the database.
If the code within the try block throws an `SQLException` we will catch it and print it using the `LOGGER`, if you want to know more about logging there is a separate module on it.
Then we use a try block within the finally block to check if conn is not null, if it is not then we close the connection.

Create

```
public void create(Customer customer) {
    try(Connection conn = DriverManager.getConnection(jdbcConnectionURL,
username, password);
            Statement statement = conn.createStatement()) {

        statement.executeUpdate("INSERT INTO customers(first_name, surname)
VALUES('" +
            customer.getFirstName() + "','" + customer.getSurname() + "')");
    } catch (SQLException e) {
        LOGGER.debug(e.getStackTrace());
    }
}
```

In the above method, we are using a try-with-resources block and passing it a connection, which is being instantiated in the same way as the previous section, and a variable called `statement` of type `Statement` which we are giving the value of `conn.createStatement`.
Inside the try block, we are calling the `executeUpdate` method against the statement and passing in an SQL query to insert a new customer into the database.
Finally, if any of the code in the try block throws an `SQLException`, we will catch it and print it using the `LOGGER`.

Read

```
public Customer customerFromResultSet(ResultSet resultSet) throws SQLException {
    Long id = resultSet.getLong("id");
    String firstName = resultSet.getString("first_name");
    String surname = resultSet.getString("surname");
    return new Customer(id, firstName, surname);
}

public Customer readCustomer(Long id) {
    try(Connection conn = DriverManager.getConnection(jdbcConnectionURL,
username, password);
            Statement statement = conn.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT FROM customers
WHERE id = " + id)) {

        resultSet.next();
        return customerFromResultSet(resultSet);
    } catch (SQLException e) {
        LOGGER.debug(e.getStackTrace());
    }
    return null;
}
```

In the example above, the readCustomer method is using a try-with-resources block with a connection, and statement as in the previous example; however, this also has the variable `resultSet` of type `ResultSet` and a value of `statement.executeQuery`.

Here we have passed our SQL query into the `statement.executeQuery` method call, which runs it from within the try-with-resources block.

Then inside the try block, we are getting the next item returned from the result set (in this case it would be the first and only item), we pass that item to another method which is building a Java object from the SQL column values and returning it.

If the code in the try block throws an `SQLException`, we catch it and print it using the `LOGGER`, and if an exception is caught then we return null.

Update

```
public Customer update(Customer customer) {
    try(Connection conn = DriverManager.getConnection(jdbcConnectionURL,
username, password);
        Statement statement = conn.createStatement()) {

        statement.executeUpdate("UPDATE customer SET first_name = '" +
customer.getFirstName()
            + "', surname = '" + customer.getSurname() + "' WHERE id = " +
customer.getId());
            return readCustomer(customer.getId());
    } catch (SQLException e) {
        LOGGER.debug(e.getStackTrace());
    }
    return null;
}
```

In the above method, we are using a try-with-resources block and passing it a connection and statement, as seen previously.
Then inside the try block, we are calling `executeUpdate` against the statement variable and passing it the SQL update statement that we want it to run.
Then we are calling the `readCustomer` method from the previous section and passing it the customer ID, then returning this.
We are returning the result of this method call so that the user can verify that the update statement worked.
If a `SQLException` is thrown from within the try block, we are catching it and printing it using the `LOGGER`.
Finally, if an exception is thrown we are returning null.

Delete

```
public void delete(Long id) {
    try (Connection conn = DriverManager.getConnection(jdbcConnectionURL,
username, password);
        Statement statement = conn.createStatement()) {

        statement.executeUpdate("DELETE FROM customers WHERE id = " + id);
    } catch (SQLException e) {
        LOGGER.debug(e.getStackTrace());
    }
}
```

In the above method, we are using a try-with-resources block and passing it a connection and statement, as seen previously.
Then inside the try block, we are calling `executeUpdate` against the statement variable and passing it the SQL delete statement that we want it to run.
If a `SQLException` is thrown from within the try block then we are catching it and printing it using the `LOGGER`.

Prepared Statement

We can also use prepared statements if we want to.
The PreparedStatement interface is a subinterface of Statement and can be used to execute parameterised queries.

```
public void createPrepared(Customer customer) {
    try(Connection conn = DriverManager.getConnection(jdbcConnectionURL,
username, password);
        PreparedStatement statement = conn.prepareStatement("INSERT INTO
customer VALUES(?,?)")) {

        statement.setString(1, customer.getFirstName());
        statement.setString(2, customer.getSurname());
        statement.executeUpdate();
    } catch (SQLException e) {
        LOGGER.debug(e.getStackTrace());
    }
}
```

In the above method, we are using a try-with-resources statement and passing it a connection as we have done previously.
We are also passing it a variable called `statement` of type `PreparedStatement` which is being instantiated with `conn.prepareStatement`.
In the `prepareStatement` method call we are passing it our SQL query, however instead of hard coding the values we are setting the values as "?".
Inside the try block we are then setting each of the two values we want in the SQL statement, giving it a number, which is the place of the variable in the order (1 is the first variable and so on), and a value (in this case it is a String as we used the `setString` method, there are other set methods for different types).
If an `SQLException` is thrown from within the try block then we are catching it and printing it using the `LOGGER`.

Exercises

There are no exercises for this module.