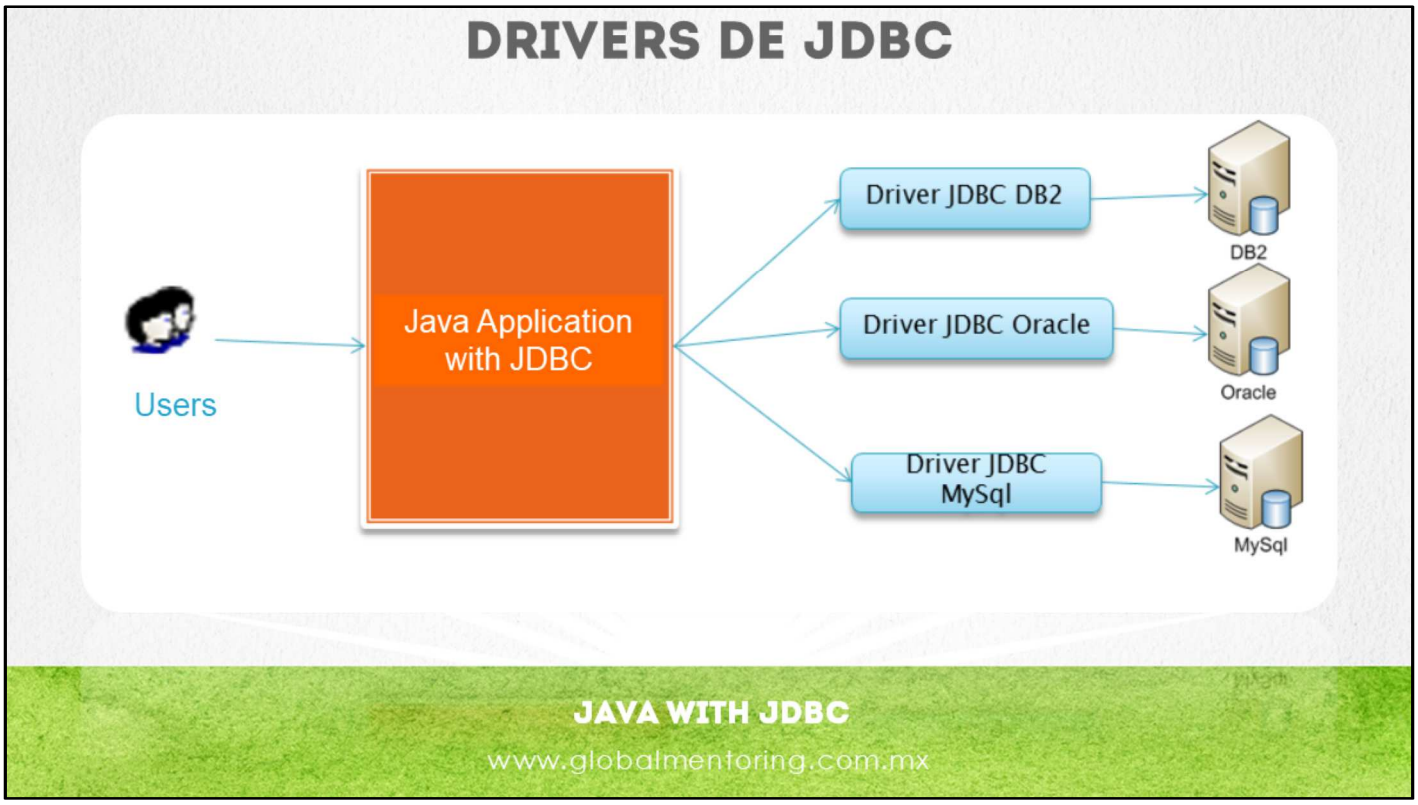
Hello, Ubaldo Acosta greets you. Welcome again. I hope you're ready to start with this lesson.

We are going to study the topic how to handle the JDBC API.

Are you ready? OK let's go!

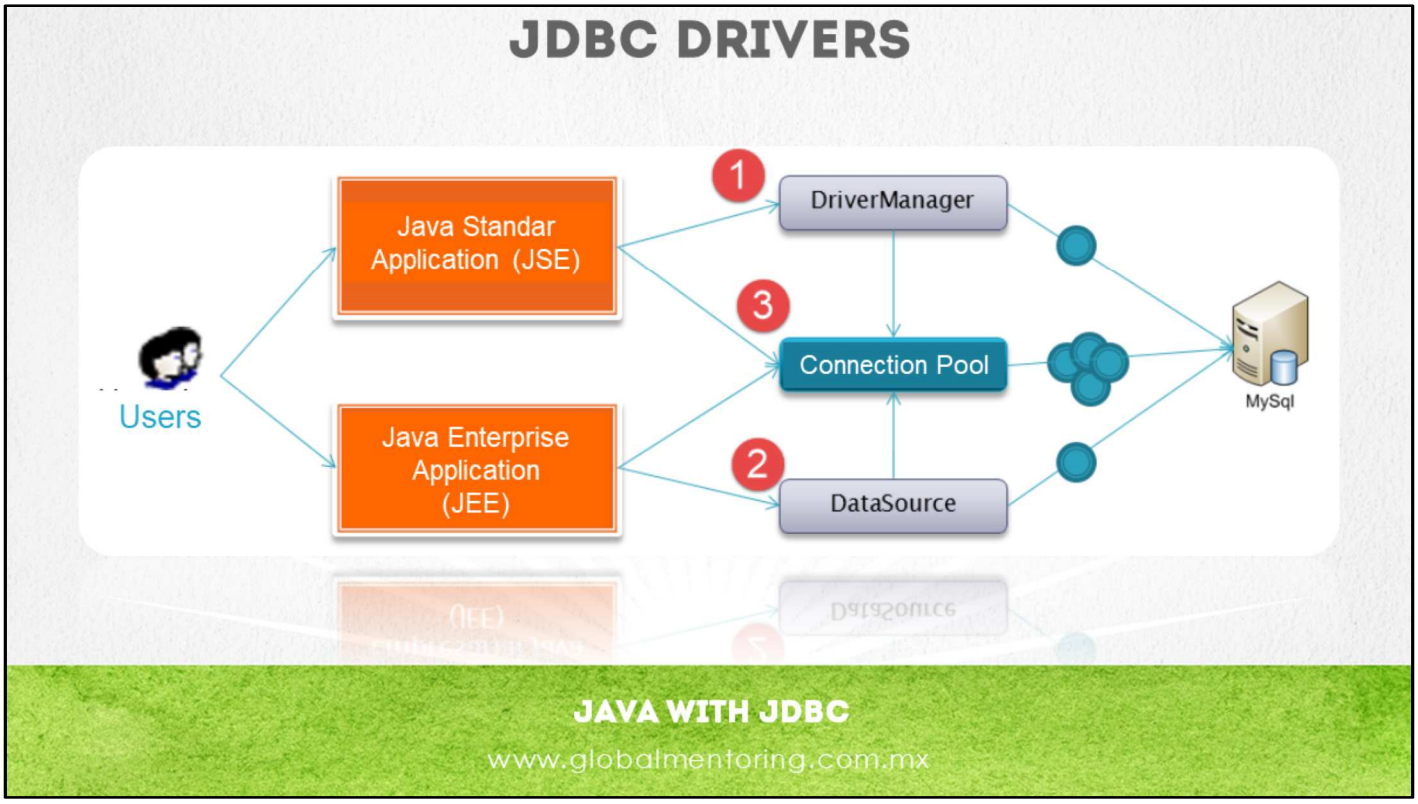


We will review below the theme of JDBC Drivers.

A driver is the implementation of the JDBC API specification defined by Java. As we have said before, each database provider implements one or several versions so that from our Java applications we can connect to your database.

The figure shows how a Java application can connect to a database for example DB2, ORACLE, MySQL among others. And we can also see how each provider creates its own driver to connect to its respective database.

There are several versions of JDBC drivers, so we must keep in mind that depending on the version of JDK is the version that we can use the JDBC driver, so normally to be able to use the latest JDBC driver from the database that select, we also have to use a recent version of the JDK.



Let's now review the topic of JDBC connections type. As we can see in the figure, there are 2 ways to connect to a database using the JDBC driver.

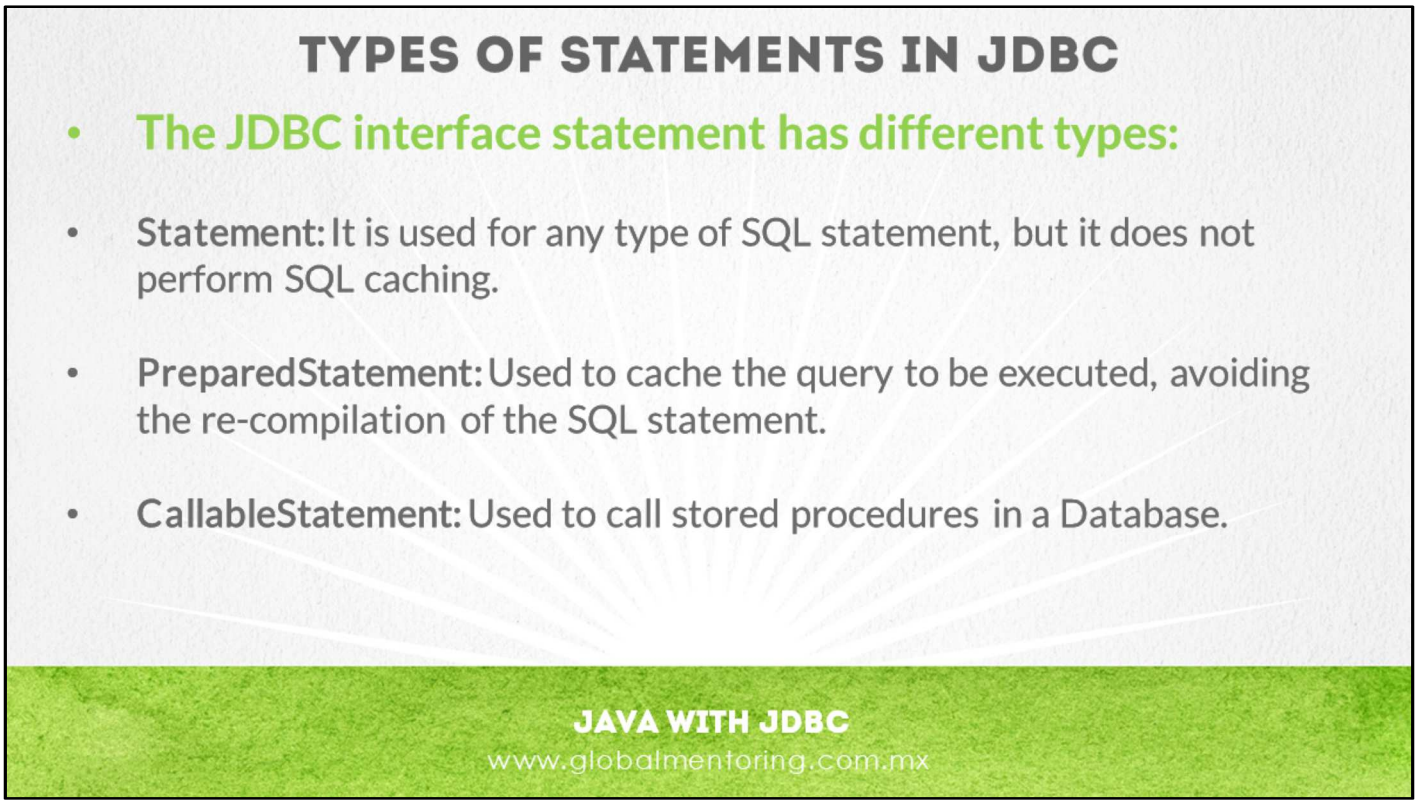
The first way that was used in the first versions of JDBC is the DriverManager option. This class is in the java.sql package and allows us to create a new connection by supporting the Class.forName method.

In the second form we can use the concept of DataSourse. This option is the most recommended, although normally we must rely on a WebServer or an application server such as Tomcat, Jboss, Glassfish, among others.

One of the advantages of using the concept of DataSource is that because the configuration is done via JNDI (Java Naming and Directory Interface) which is a Java API for the management of a resource directory, this configuration allows the Application Server we can change the values of the connection to database on the fly and without the need to modify our Java code. We will see more examples of the configuration of this type of connections in the course of JSPs and Servlets.

Finally there is a third way to connect to the database and is through a connection pool. This pool of connections is a set of objects which already have a live connection to the database, this allows us to minimize the cost and time of creation of the objects because each connection to the database is an expensive process, and therefore, if we already have several database connection objects available, we will minimize the resources and increase the performance of our application in Java.

Finally, we can see in the figure that normally a standard application makes use of what is the driver manager and a business application makes use of a DataSourse, in both cases we can combine the technologies to use a pool of connections and thus make each one more efficient of the processes according to the technology that we are using.



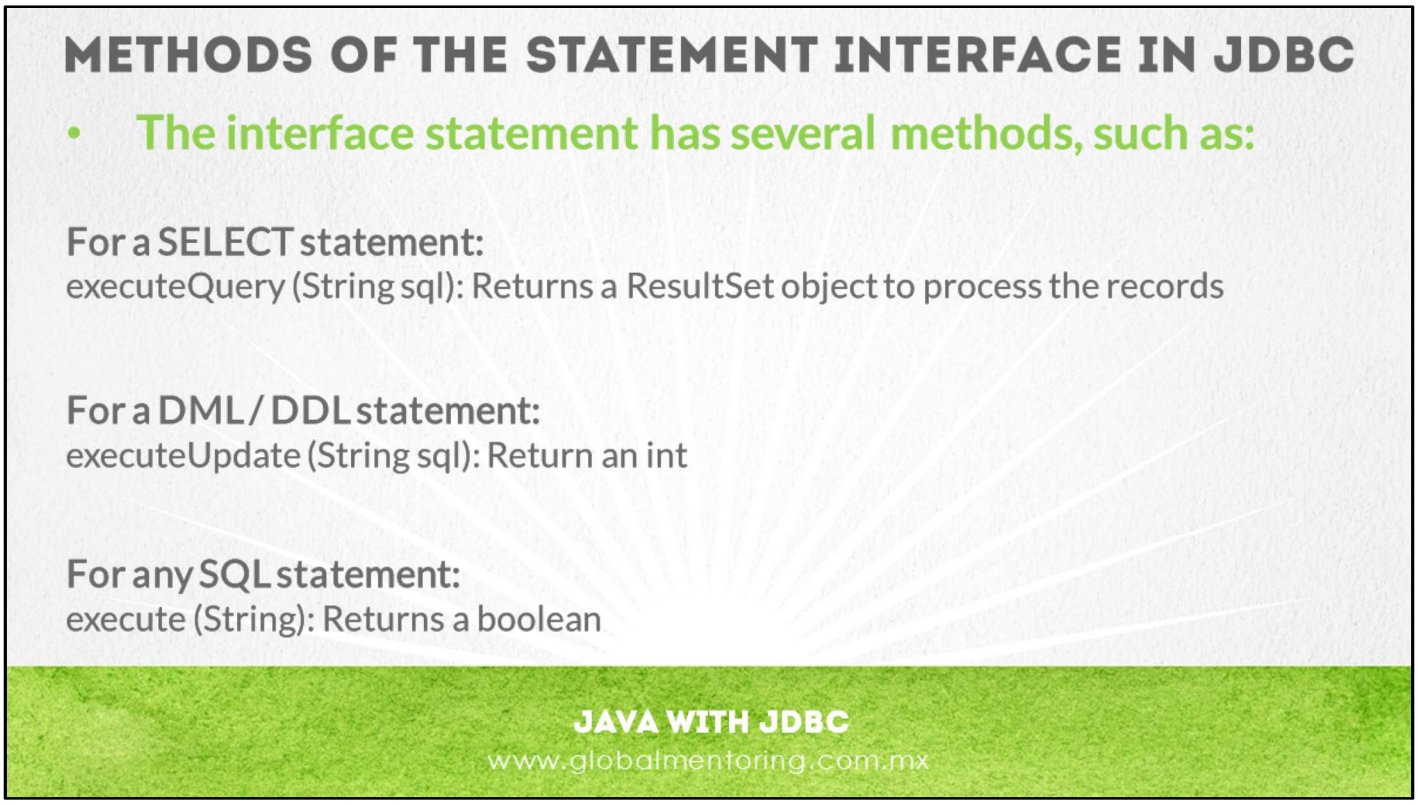
We will now review the Statement interface, which is the most basic interface to execute an SQL statement.

When executing a SQL statement repeatedly, a statement object compiles the SQL query that is executed on each occasion.

The PrepareStatement interface inherits from the Statement interface and can precompile an SQL statement, so it is a more efficient option if we need to execute a SQL statement repeatedly, in turn we can also specify parameters to execute in our query , this we will see in later exercises.

Finally, the Callable interface is used to send a stored procedure, which can return information in the form of a cursor, a specific type of data or in some other way depending on the result that needs to be processed.

Next we will create examples related to the Statement interface.

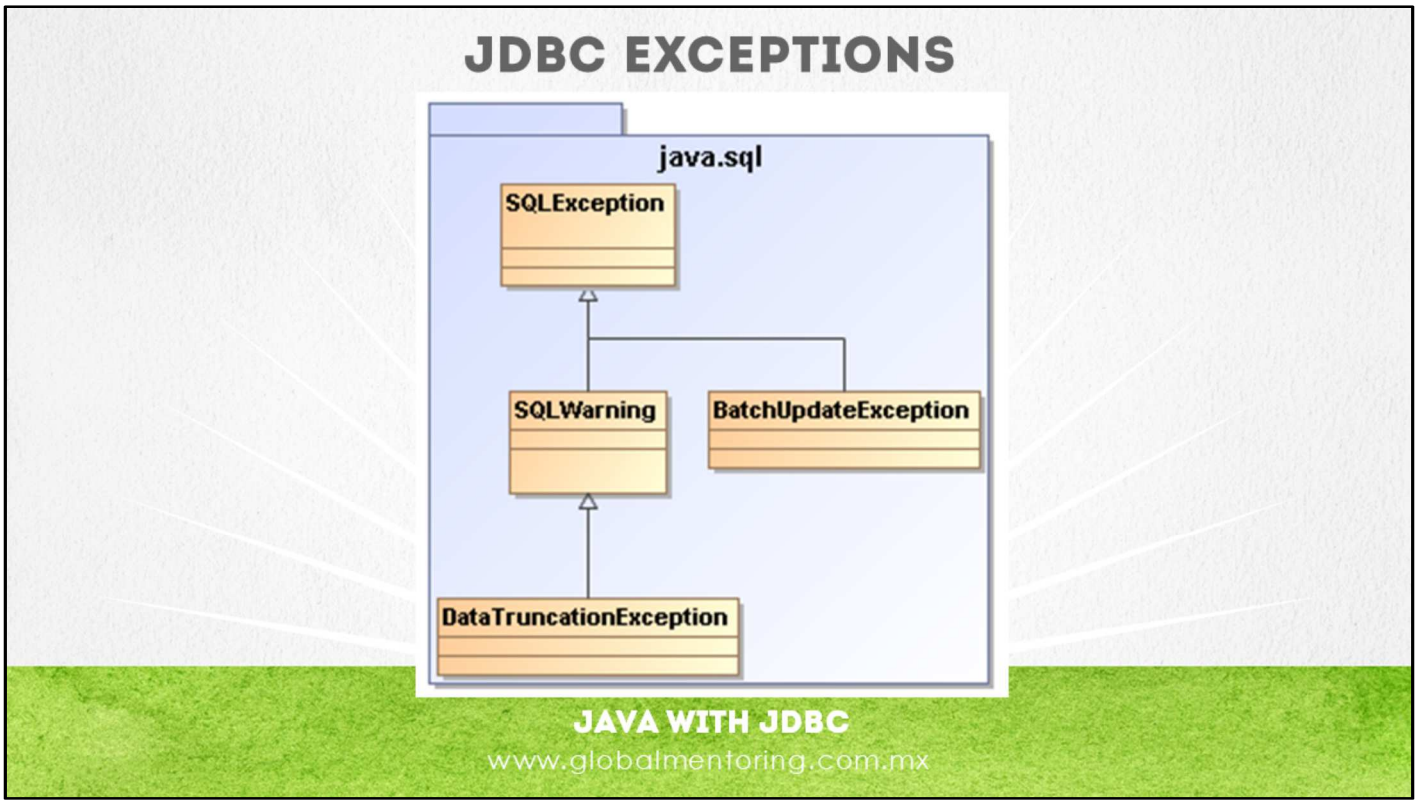


Now let's review the use of Statement in JDBC. The statement interface has several methods. There are methods to make a SELECT, to execute some instruction DML (Data Manipulation Language), DDL (Data Definition Language) or for any other SQL type statement.

The method of **executeQuery** is used to execute statements of type select and therefore the method returns a ResultSet object, which stores the result in the form of a two-dimensional matrix, that is, in lines and columns, so we can process the result of the query without any problem.

The method **executeUpdate** is used to execute DML (Data Manipulation Language) statements such as insert, update and delete statements. It will also allow us to execute statements of type DDL (Data Definition Language) as are the statements create table, truncate table, among others. The function executeUpdate returns an integer, indicating the number of records affected as a result of executing the desired query.

Finally, the **execute** method allows us to execute any type of SQL statement. This method returns a boolean value, which returns true if the result is a ResulSet, that is, if a select was executed, and returns false if any other statement was executed, such as insert, update, delete, or some other.



In the figure shown we can see some of the exception classes in the JDBC API.

The SQLException class is the most used to process exceptions when using JDBC. There are more exception classes, which are in the java.sql package.

Because the SQLException class descends from the Exception class we are forced to process that exception by a try / catch block or we must declare it as part of the signature of the method that makes use of the JDBC API.

Little by little we will be reviewing the handling of exceptions in JDBC, but in the exercises that we are going to carry out we will be managing this concept so that we can clarify it little by little.