JDBC

Setup

Run the SetupDataBase to setup the h2 db and to clean up data prior to each other test.

JDBC Core

Classes & Interfaces

Name	Interface/Class
Driver	Interface
DriverManager	Class
PreparedStatement	Interface
ResultSet	Interface
Connection	Interface

Connection

This is the proper way to get a connection:

```
try (Connection conn = DriverManager.getConnection(URL)) {}
```

Also this (with username & password)

```
try (Connection conn = DriverManager.getConnection(URL), username, p
```

This is the **wrong way**:

```
var conn = new Connection(URL, "userName", "password");  //does not
```

`Connection` is abstract; cannot be instantiated!

From the Connection

```
//get a Statement
Statement statement = connection.createStatement();

//get a PreparedStatement
PreparedStatement ps = connection.prepareStatement(sql);

//get a CallableStatement (store procedure)
CallableStatement callableStatement = connection.prepareCall(sql);
```

PreparedStatement

`PreparedStatement` is Base-1 indexing, which is, indexes **start with 1**, not 0.

```
var sql = "UPDATE world SET environment = null WHERE environment = 
ps.setString(0, "snow");  //not valid, he indexing starts with 1!
ps.setString(1, "snow"); //valid
//ps.setString("environment", "snow"); //not valid, only index
```

setNull

The `setNull()` method requires you to pass the data type.

```
PreparedStatement ps;
ps.setNull(2, Types.VARCHAR);
```

execute

I can use `execute` to run update/delete/add or select. `execute()` is defined in both `Statement` and `PreparedStatement`.

```
Statement statement = conn.createStatement()){
boolean isResultSet = statement.execute(updateSQL);
if(isResultSet){
    ResultSet resultSet = statement.getResultSet();
    //get the result from ResultSet
}else{
    int updateCount = statement.getUpdateCount();
    System.out.println(updateCount);
}
```

Closing Resources

- Closing a `Connection` also closes `PreparedStatement` (or `CallableStatement`) and `ResultSet`.
- Closing a `PreparedStatement` (or `CallableStatement`) also closes the `ResultSet`.

Procedure

Store Procedure

`registerOutParameter` is the method to call.

```
var sql = "{?= call magic_number(?) }";
try (Connection conn = DriverManager.getConnection(URL);
    var cs = conn.prepareCall(sql)) {
  cs.registerOutParameter(1, Types.INTEGER);
  cs.execute();
  //num is defined in the body of the procedure. Check SetupDataBase
  System.out.println(cs.getInt("num"));
```

```
//double_number(INOUT num INT)
cs.setInt("num", 23);
cs.registerOutParameter("num", Types.INTEGER);
boolean isResultSet = cs.execute();
if(!isResultSet){
System.out.println(cs.getInt("num"));
}
```

Mind the space between **=** and **call**

```
var sql = "{?= call magic_number(?) }";  //right
var sql = "{?=call magic_number(?) }";  //wrong, missing space
var sql = "{ call magic_number(?) }";  //equivalent to the first
```

CallableStatement

```
public interface CallableStatement extends PreparedStatement {/**/}
```

WARN: `prepareCall(sql)` returns a `CallableStatement`, **not** a ~~`PreparedStatement`~~,

```
CallableStatement cs = connection.prepareCall(sql))
```

INOUT parameter

```
CREATE PROCEDURE double_number(INOUT num INT) READS SQL DATA
```

num is used for both **IN** and **OUT** parameter.

```
cs.setInt("num", 23);
cs.registerOutParameter("num", Types.INTEGER);
cs.execute();
System.out.println(cs.getInt("num"));
```

I do not access to the `ResultSet`.

JDBCTypes

These two are (somehow) equivalent.

```
cs.setInt("num", 23);
cs.registerOutParameter("num", Types.INTEGER);
```

or

```
cs.setObject(1, 23, JDBCType.BIGINT);
cs.registerOutParameter("num", JDBCType.BIGINT);
```

Rollback

Savepoint

```
conn.setAutoCommit(false);
var moe = conn.setSavepoint();
conn.rollback(moe)
```

savepoint with autocommit true

A savepoint can only be used when autocommit mode is turned off.

```
conn.setAutoCommit(true);
//java.sql.SQLException: savepoint exception: invalid specification
Savepoint sp1 = conn.setSavepoint();
```

SavePoint

autocommit

When the auto-commit feature is enabled, invoking a rollback operation has no impact, and no exception is thrown

```
connection.setAutoCommit(true);
  try (PreparedStatement ps = connection.prepareStatement(sql,
      ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY)) {
    ps.executeUpdate();
  }
connection.rollback();  //no effect, but no exception is thrown
```

Rollback

Various

ResultSet parameters

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.CONCUR_READ_ONLY`

Remember that type always comes first