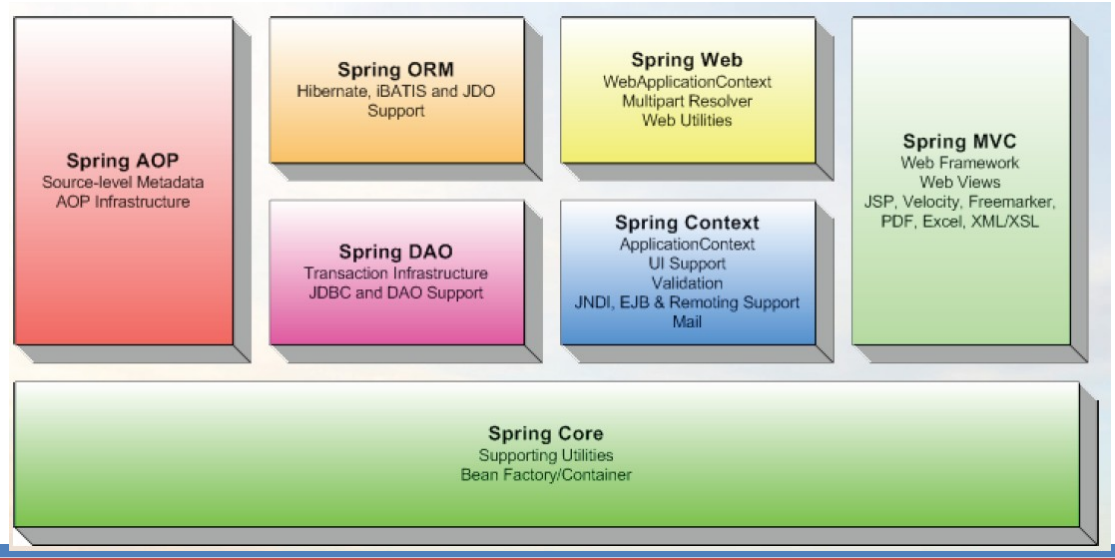
Unit 2.2
Spring JDBC template

LUCÍA SAN MIGUEL LÓPEZ

Spring framework

- Spring is an open source framework created to address the complexity of enterprise application development.
- One of the main advantages of the Spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a consistent framework for J2EE application development.



Spring JDBC template

- JDBC produces a lot of code, such as opening/closing a connection to a database, handling sql exceptions etc. It makes the code extremely difficult to read.
- Implementing JDBC in the Spring Framework takes care of working with many low-level operations (opening/closing connections, executing SQL queries, etc.).

Action	Spring	You
Define connection parameters.		X
Open the connection.	X	
Specify the SQL statement.		X
Declare parameters and provide parameter values		X
Prepare and execute the statement.	X	
Set up the loop to iterate through the results (if any).	X	
Do the work for each iteration.		X
Process any exception.	X	
Handle transactions.	X	
Close the connection, statement and resultset.	X	

Spring JDBC Maven Dependencies

- We will need the following dependency:

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.3.10</version>
</dependency>
```

- If you are using a relational database such as MySQL, add it's corresponding java driver dependencies.

Spring JDBC template

- JDBC in Spring has several approaches for interacting with the database:
  - **JdbcTemplate** class:
    - This is the base class that manages the processing of all events and database connections.
    - Executes SQL queries, iterates over the ResultSet, and retrieves the called values, updates the instructions and procedure calls, “catches” the exceptions, and translates them into the exceptions defined in the org.springframwork.dao package.

https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/jdbc.html

Spring JdbcTemplate configuration

- JDBCTemplate class uses a Datasource reference

```
JdbcTemplate jtm = new JdbcTemplate(<DataSource>);
```

- We can use the Datasource already defined in DBConnPool

```
JdbcTemplate jtm = new
JdbcTemplate(DBConnPool.getInstance().getPool());
```

Spring JdbcTemplate: Queries

1. Class *BeanPropertyRowMapper* can be used for simple classes:
   List l= jtm.query("SELECT * FROM SUPPLIERS", BeanPropertyRowMapper.newInstance(Supplier.class));

2. Use *RowMapper* anonymous inner class for nested classes:

```
List<Actor> actors = this.jdbcTemplate.query(
        "select first_name, last_name from t_actor",
        new RowMapper<Actor>() {
            public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {
                Actor actor = new Actor();
                actor.setFirstName(rs.getString("first_name"));
                actor.setLastName(rs.getString("last_name"));
                return actor;
            }
        });
```

3. Use single class RowMapper to reuse code:

```
public List<Actor> findAllActors() {
    return this.jdbcTemplate.query( "select first_name, last_name from t_actor", new ActorMapper());
}

private static final class ActorMapper implements RowMapper<Actor> {

    public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {
        Actor actor = new Actor();
        actor.setFirstName(rs.getString("first_name"));
        actor.setLastName(rs.getString("last_name"));
        return actor;
    }
}
```

Spring JdbcTemplate: Insert, update, delete

- General solution:

```
this.jdbcTemplate.update(
        "insert into t_actor (first_name, last_name) values (?, ?)",
        "Leonor", "Watling");
```

```
this.jdbcTemplate.update(
        "update t_actor set = ? where id = ?",
        "Banjo", 5276L);
```

```
this.jdbcTemplate.update(
        "delete from actor where id = ?",
        Long.valueOf(actorId));
```

Spring JdbcTemplate: Insert with autoGenerated key

```
KeyHolder keyHolder = new GeneratedKeyHolder();
JdbcTemplate jtm = new JdbcTemplate(
DBConnectionPool.getInstance().getDataSource());
jtm.update(connection -> {
PreparedStatement ps = connection
.prepareStatement("insert into asignaturas (NOMBRE,CICLO,CURSO) VALUES (?,?,?)",
Statement.RETURN_GENERATED_KEYS);
ps.setString(1, name);
ps.setString(2, ciclo);
ps.setString(3, course);
return ps;
},keyHolder);
int idReturned = (int) keyHolder.getKey());
```

Spring JdbcTemplate: Insert with SimpleJdbcInsert

- Recommended solution for insert:
  - A SimpleJdbcInsert is a multi-threaded, reusable object providing easy insert capabilities for a table.
    - All you need to provide is:
      - Name of the table
      - A Map containing the column names and the column values.

```
public int addSupplier(Supplier s) {

    SimpleJdbcInsert jdbcInsert = new SimpleJdbcInsert(
        DBConnPool.getInstance().getDataSource()).withTableName("suppliers");

    Map<String, Object> parameters = new HashMap<>();

    parameters.put("SUPP_ID", s.getSupp_id());
    parameters.put("STREET", s.getStreet());
    parameters.put("COUNTRY", s.getCountry());
     int res = jdbcInsert.execute(parameters);
    return res;
}
```

Spring JdbcTemplate: Insert with autoGenerated key (SimpleJdbcInsert)

```
public Supplier addSupplierGK(Supplier s) {

    SimpleJdbcInsert jdbcInsert = new SimpleJdbcInsert(
        DBConnPool.getInstance().getDataSource()).withTableName("suppliers")
            .usingGeneratedKeyColumns("SUPP_ID");

    Map<String, Object> parameters = new HashMap<>();

    parameters.put("STREET", s.getStreet());
    parameters.put("COUNTRY", s.getCountry());
    s.setSupp_id((int) jdbcInsert.executeAndReturnKey(parameters).longValue());
    return s;
```

Transactions

- The Spring Framework provides a consistent abstraction for transaction management:
  - Uses class DataSourceTransactionManager

```
TransactionDefinition txDef = new DefaultTransactionDefinition();
DataSourceTransactionManager transactionManager = new DataSourceTransactionManager(dataSource);
TransactionStatus txStatus = transactionManager.getTransaction(txDef);

try {
  JdbcTemplate jtm = new JdbcTemplate(
      transactionManager.getDataSource());
  res = jtm.update(DELETE_coffee_QUERY, id);
  res = jtm.update(DELETE_supplier_QUERY, id);

  transactionManager.commit(txStatus);

} catch (Exception e) {

  transactionManager.rollback(txStatus);
```