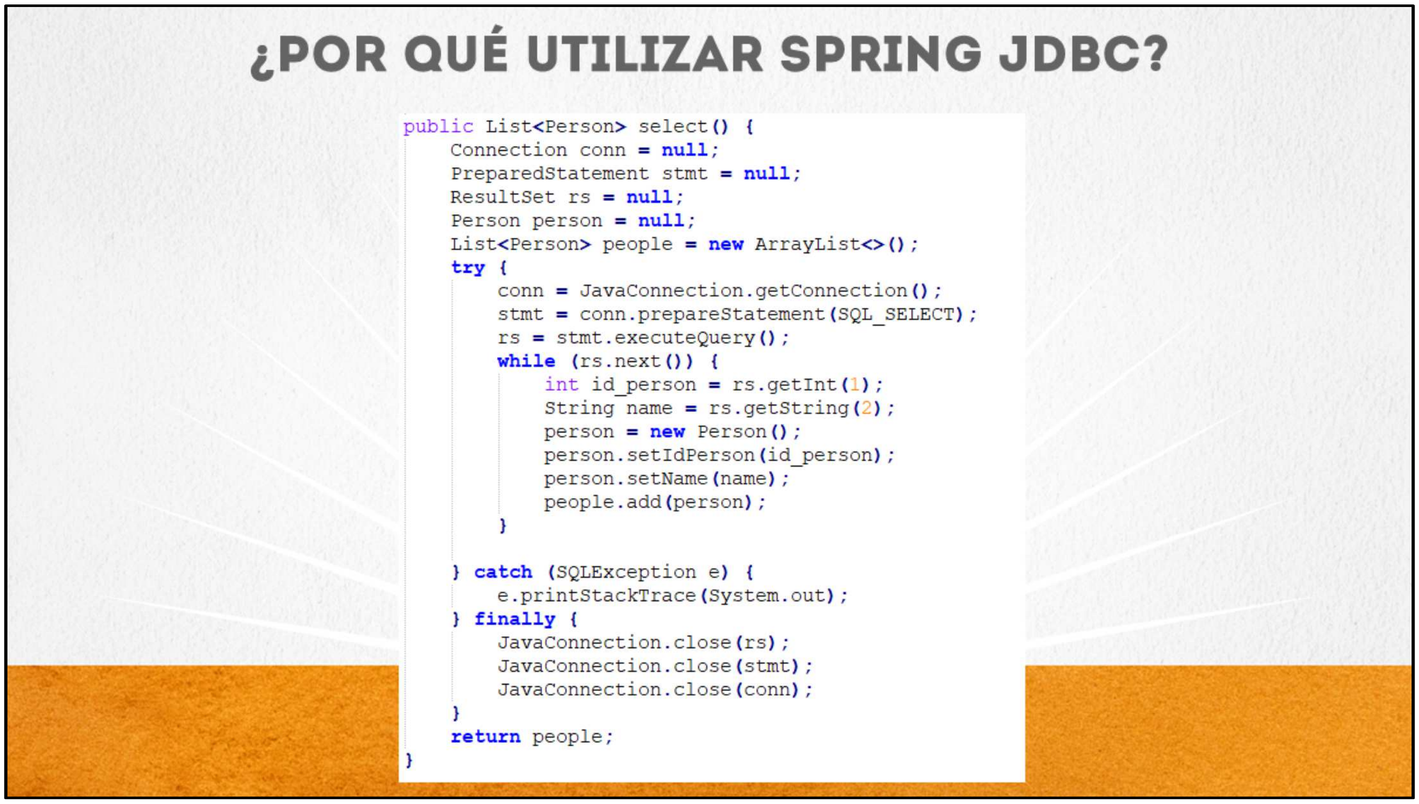
SPRING FRAMEWORK COURSE
SPRING JDBC
By the expert: Eng. Ubaldo Acosta
Eng. Ubaldo Acosta
Global Mentoring
JAVA
ONLINE COURSES
UNIVERSITY
SPRING FRAMEWORK COURSE
www.globalmentoring.com.mx

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

We are going to study the Spring JDBC topic.

Are you ready? OK let's go!



¿POR QUÉ UTILIZAR SPRING JDBC?

```java
public List<Person> select() {
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;
    Person person = null;
    List<Person> people = new ArrayList<>();
    try {
        conn = JavaConnection.getConnection();
        stmt = conn.prepareStatement(SQL_SELECT);
        rs = stmt.executeQuery();
        while (rs.next()) {
            int id_person = rs.getInt(1);
            String name = rs.getString(2);
            person = new Person();
            person.setIdPerson(id_person);
            person.setName(name);
            people.add(person);
        }

    } catch (SQLException e) {
        e.printStackTrace(System.out);
    } finally {
        JavaConnection.close(rs);
        JavaConnection.close(stmt);
        JavaConnection.close(conn);
    }
    return people;
}
```

One of the objectives of Spring is the simplification of tasks when creating Java applications. Most of the business systems persist their information in Databases, for this reason in this lesson we will focus on simplifying the use of JDBC using Spring.

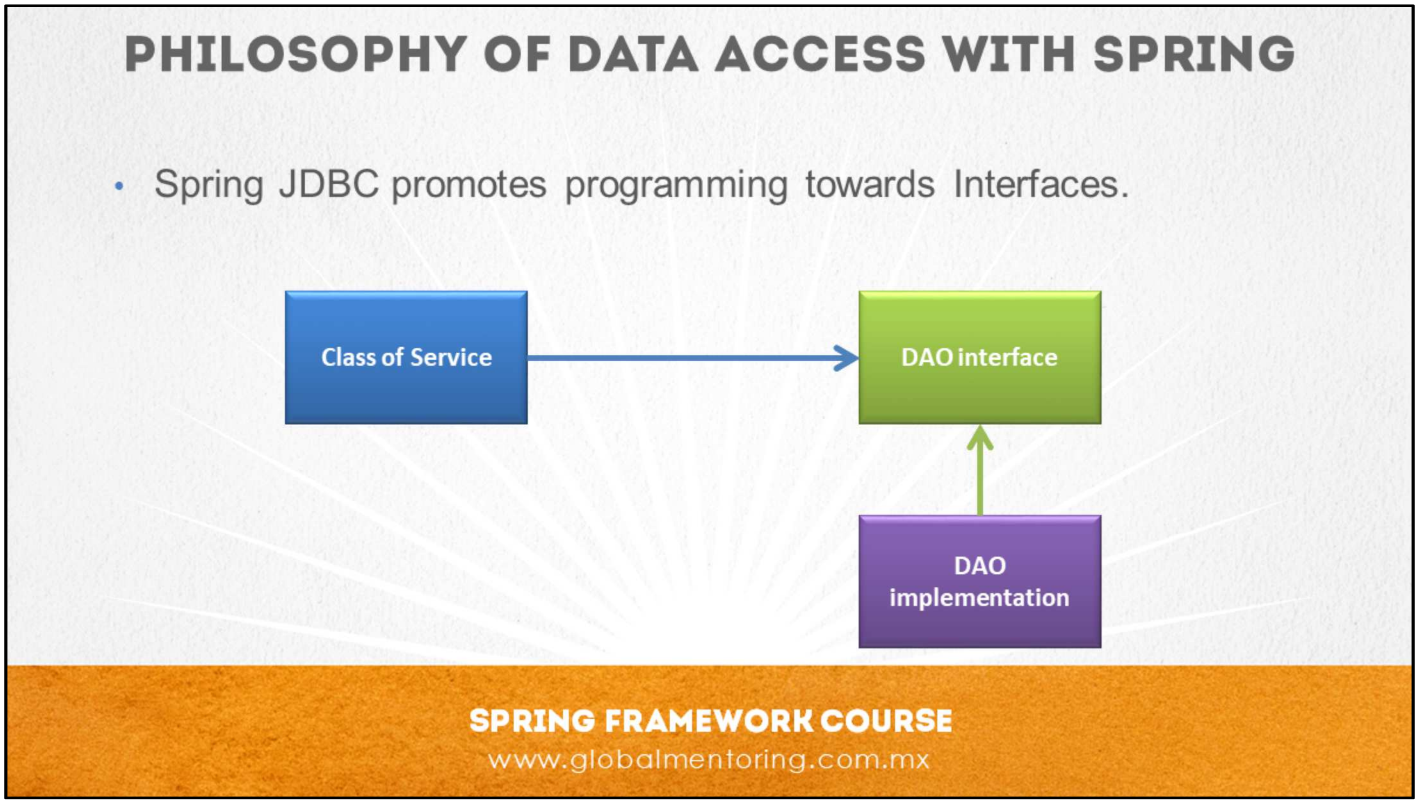
As we can see in the figure, and if you already have some experience using JDBC, you will have noticed that it is difficult to consult the database, however basic it may be, and many of the steps to follow are repetitive and unnecessary from the point of our Java method.

Therefore, Spring, in search of this simplification, added the Spring JDBC package, which applies the Template concept to simplify processes such as Open and Close Database connections, better handling of Exceptions, among many other features.

In addition to simplifying the management of JDBC, Spring offers transparent integration with persistence frameworks, such as: Hibernate, JPA, JDO, iBatis, among several others. Removing complexity in integration and simplifying the integration between layers of our Java Web and Enterprise architectures.

However, it should be mentioned that Spring JDBC is not an ORM framework (Object Relational Mapping) anymore, but its intention is only to simplify the use of the JDBC API in our Java applications.

In this lesson we will study how Spring simplifies several of the tasks to create our data layer using JDBC.



PHILOSOPHY OF DATA ACCESS WITH SPRING

- Spring JDBC promotes programming towards Interfaces.

Spring JDBC promotes the use of Interfaces in order to create systems with low coupling and high cohesion.

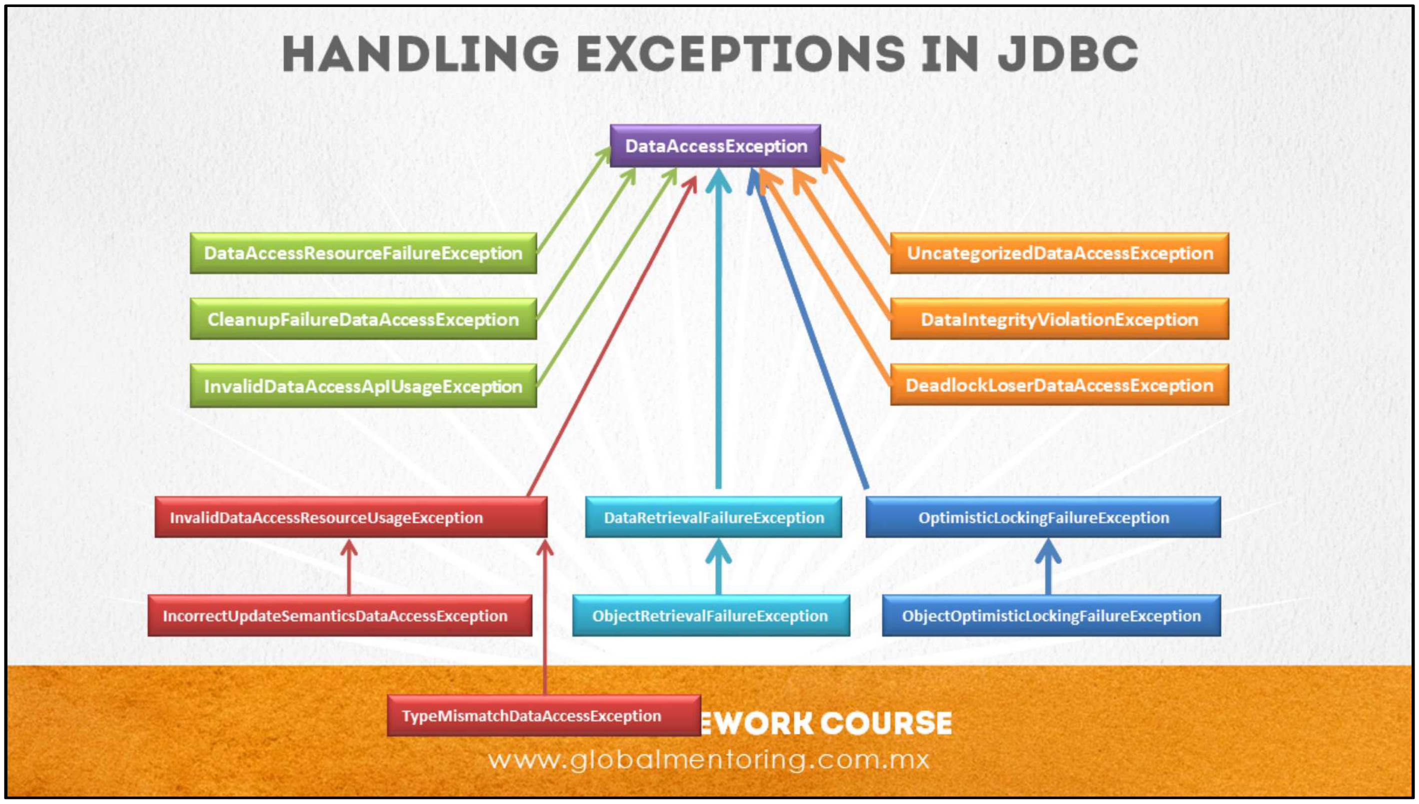
In the figure we can see the use of the DAO (Data Access Object) design pattern, which is responsible for establishing communication with the database.

However, the functionality of this class must be exposed through an interface. Therefore, the class of Service which needs to communicate with the database, does so through the DAO interface and not through any implementation.

This configuration allows to make unit tests more easily, since at any time we can change the DAO class implementation, even without really needing to connect with a database, but only simulate the information that should be obtained.

And most importantly, the service classes do NOT need to know what technology is being used to connect to the database, that is, we could use pure JDBC, JPA, Hibernate, iBatis, among other technologies, and the service class so agnostic uses these interfaces without needing to know what technology is used to read and write to the database.

With this configuration we will create all our data layer, with this we will allow to change technology with a lesser impact if this is necessary and also our architecture is made much more flexible to future changes.


HANDLING EXCEPTIONS IN JDBC
DataAccessException
DataAccessResourceFailureException
CleanupFailureDataAccessException
InvalidDataAccessApiUsageException
UncategorizedDataAccessException
DataIntegrityViolationException
DeadlockLoserDataAccessException
InvalidDataAccessResourceUsageException
DataRetrievalFailureException
OptimisticLockingFailureException
IncorrectUpdateSemanticsDataAccessException
ObjectRetrievalFailureException
ObjectOptimisticLockingFailureException
TypeMismatchDataAccessException
EWORK COURSE
www.globalmentoring.com.mx

If you have ever written JDBC code, you will know that the exception handling of type SQLException is too much code for the little or no business functionality that you add.

When handling exceptions of type SQLException there is very little that we can do to correct this type of errors, besides that we do not have details of what exactly happened.

Some of the most common problems of type SQLException are:

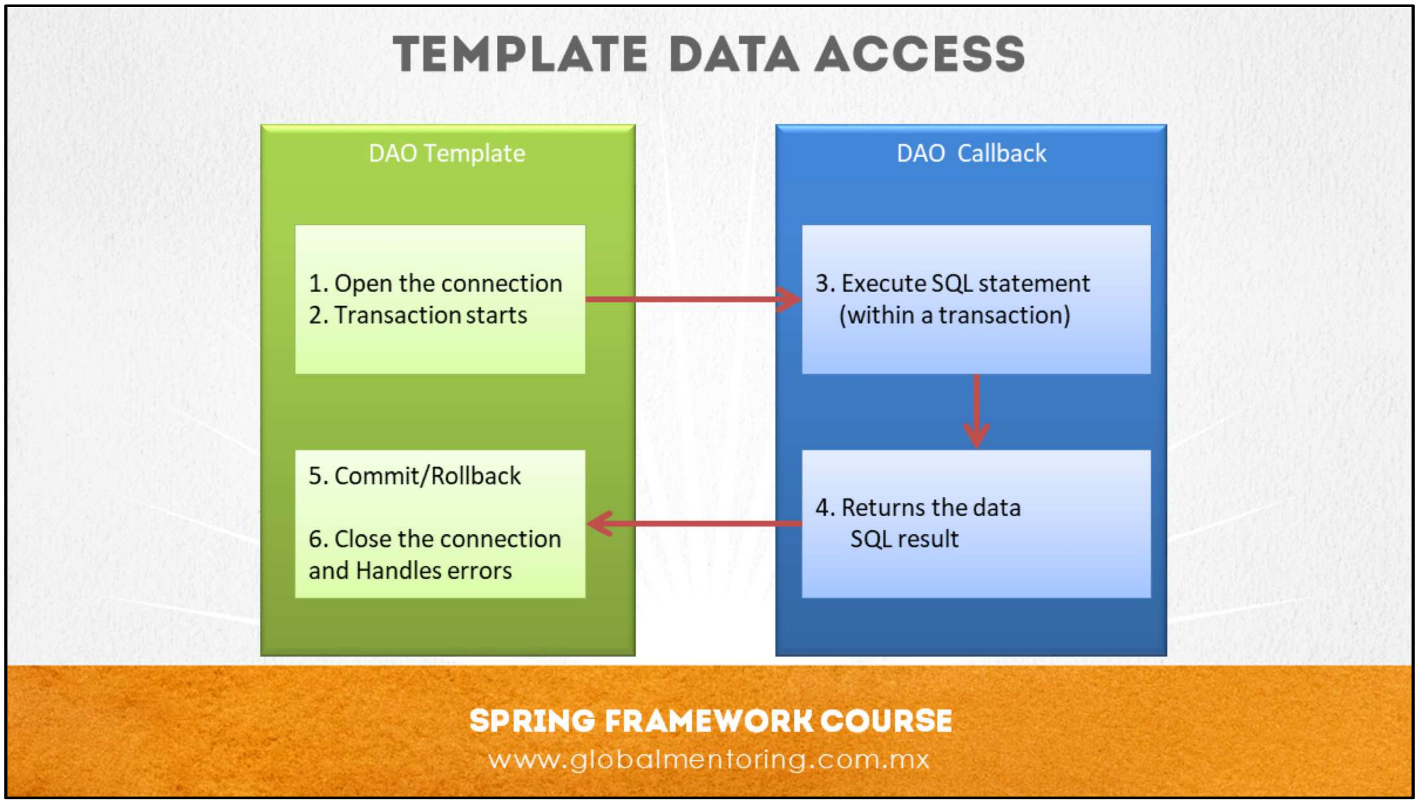
- The application could not connect to the database
- The SQL query has syntax errors and can not be executed
- The referenced table or column does NOT exist
- An attempt to insert or update violates database restrictions

Which leads us to the conclusion that many of the SQLException type exceptions are fatal and we can not do anything to recover from these exceptions.

With this in mind, Spring has converted all SQL exceptions into Runtime Exception type exceptions, that is, we are not required to add try / catch blocks to our code, and optionally we can process an exception only in cases where we can recover the flow of the application.

In the figure we can see the hierarchy of SQL classes that Spring handles, starting with the most generic, known as DataAccessException, and from this exception we have a set of exceptions that give us much more detail than the exceptions of the JDBC API.

With this class hierarchy we can add try / catch code that is useful for our business and also add cases where we can recover from this type of exception.


TEMPLATE DATA ACCESS
DAO Template
1. Open the connection
2. Transaction starts
5. Commit/Rollback
6. Close the connection and Handles errors
DAO Callback
3. Execute SQL statement (within a transaction)
4. Returns the data SQL result
SPRING FRAMEWORK COURSE
www.globalmentoring.com.mx

The Template design pattern is used with the objective of taking care of tedious or repetitive tasks, and leaving what is really important to us.

With this vision, Spring created the concept of Template Data Access. As we can see in the figure, the DAO Template is responsible for the most common tasks when we work with the JDBC API.

The tasks that this realizes are:

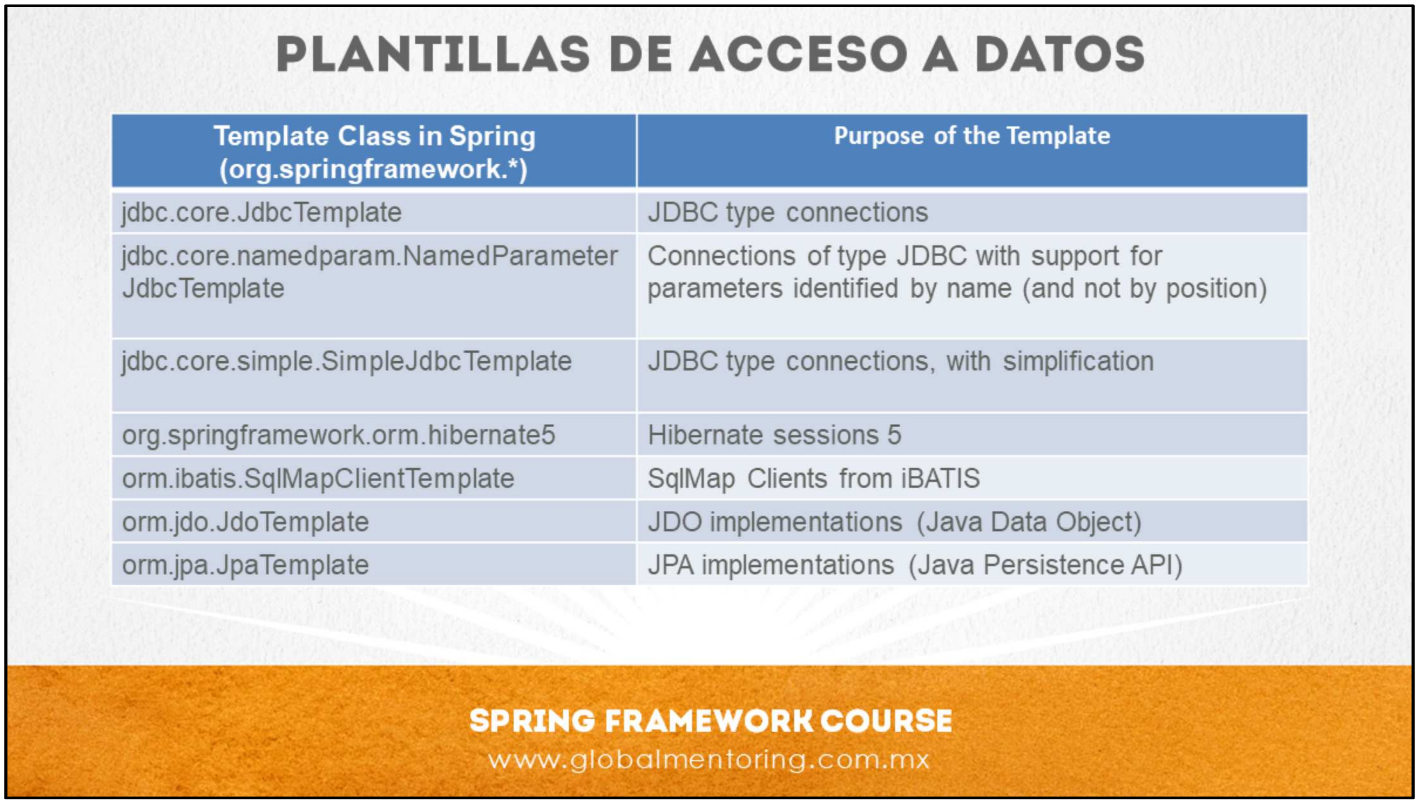
- Open the connection to the database (open connection)
- Start a transaction (beggin transaction)
- Terminate the transaction (committing or rollback)
- Close the database connection
- Convert and Handle SQLException type exceptions

On the other hand, the only work that we should be responsible for is:

- Prepare our SQL statement to be executed
- Recover the data and return it

To accomplish this task, Spring uses the concept of Callback methods, which are functions that are executed indirectly from the DAO template, and allow us to separate the repetitive tasks (managed by the template) and the relevant tasks for the data layer. .

As we can see, this is a very convenient way to execute queries to a database, because on the one hand Spring supports us with the most tedious and repetitive code and frees us from the task of adding this code by ourselves, and so we only take care of adding lines of code that really add value to our data layer.



PLANTILLAS DE ACCESO A DATOS

Template Class in Spring (org.springframework.*)	Purpose of the Template
jdbc.core.JdbcTemplate	JDBC type connections
jdbc.core.namedparam.NamedParameter JdbcTemplate	Connections of type JDBC with support for parameters identified by name (and not by position)
jdbc.core.simple.SimpleJdbcTemplate	JDBC type connections, with simplification
org.springframework.orm.hibernate5	Hibernate sessions 5
orm.ibatis.SqlMapClientTemplate	SqlMap Clients from iBATIS
orm.jdo.JdoTemplate	JDO implementations (Java Data Object)
orm.jpa.JpaTemplate	JPA implementations (Java Persistence API)

SPRING FRAMEWORK COURSE
www.globalmentoring.com.mx

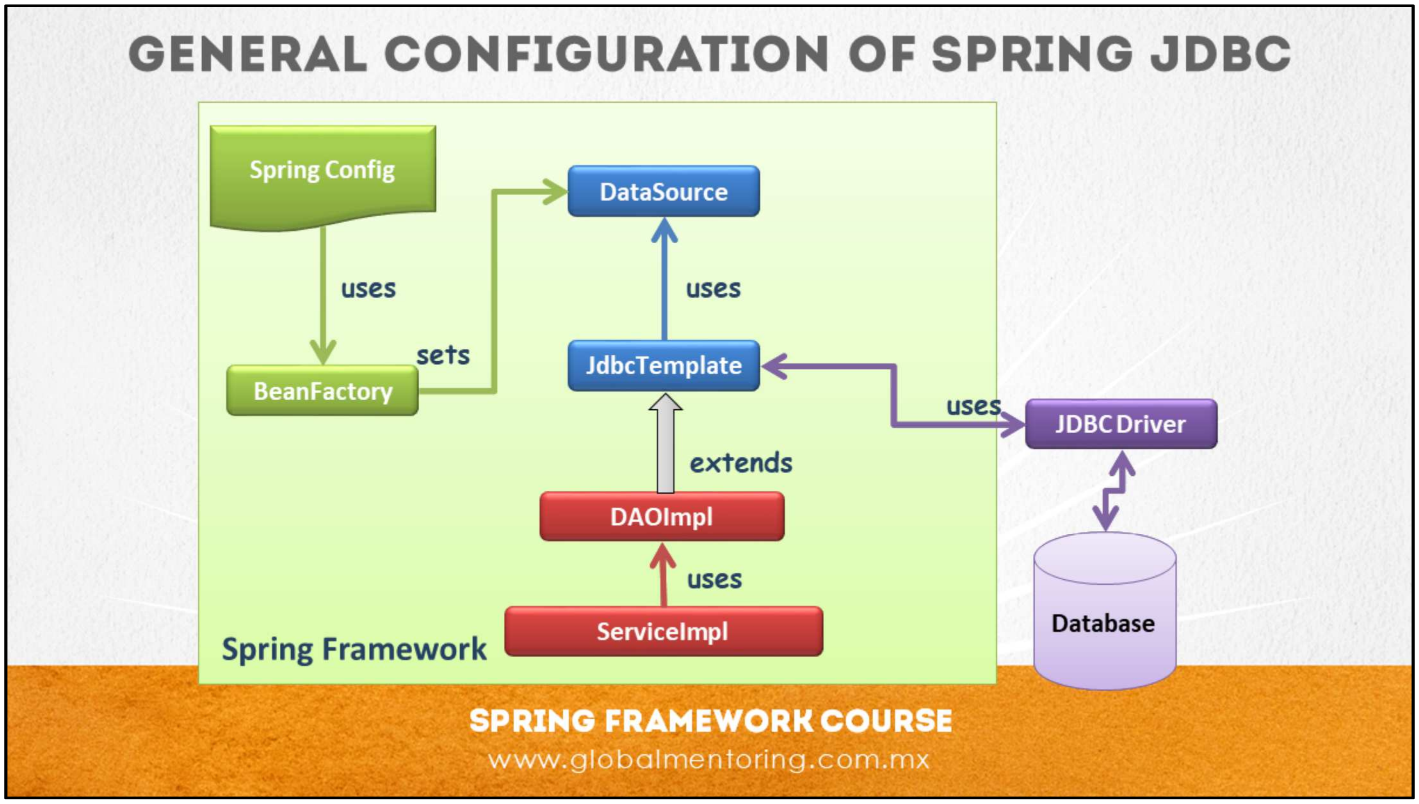
Spring brings several templates by default depending on the data access technology that will be used, for example you can see templates for technologies such as:

- Jdbc
- Hibernate
- iBatis
- JDO
- JPA
- Among other

The selection of one template or another will depend on the selected data layer technology, for example if we are using ORM (Object Relational Mapping) frameworks such as Hibernate, it is very easy to rely on the template of this technology to simplify the integration between the technologies of Spring and Hibernate.

Spring by default does not add ORM more to its list of projects, however Spring JDBC aims to simplify data access when we use the JDBC API directly, or when we are going to use an ORM technology, its templates also make it much easier to use of these data persistence frameworks.

In later lessons we will study in detail the integration between frameworks such as Spring and Hibernate.



In the figure we can see the basic architecture of Spring JDBC.

The container of Spring is in charge of the configuration of the beans and the injection of dependencies.

Subsequently, a service class needs to recover information from the database, for which it relies on the respective DAO class.

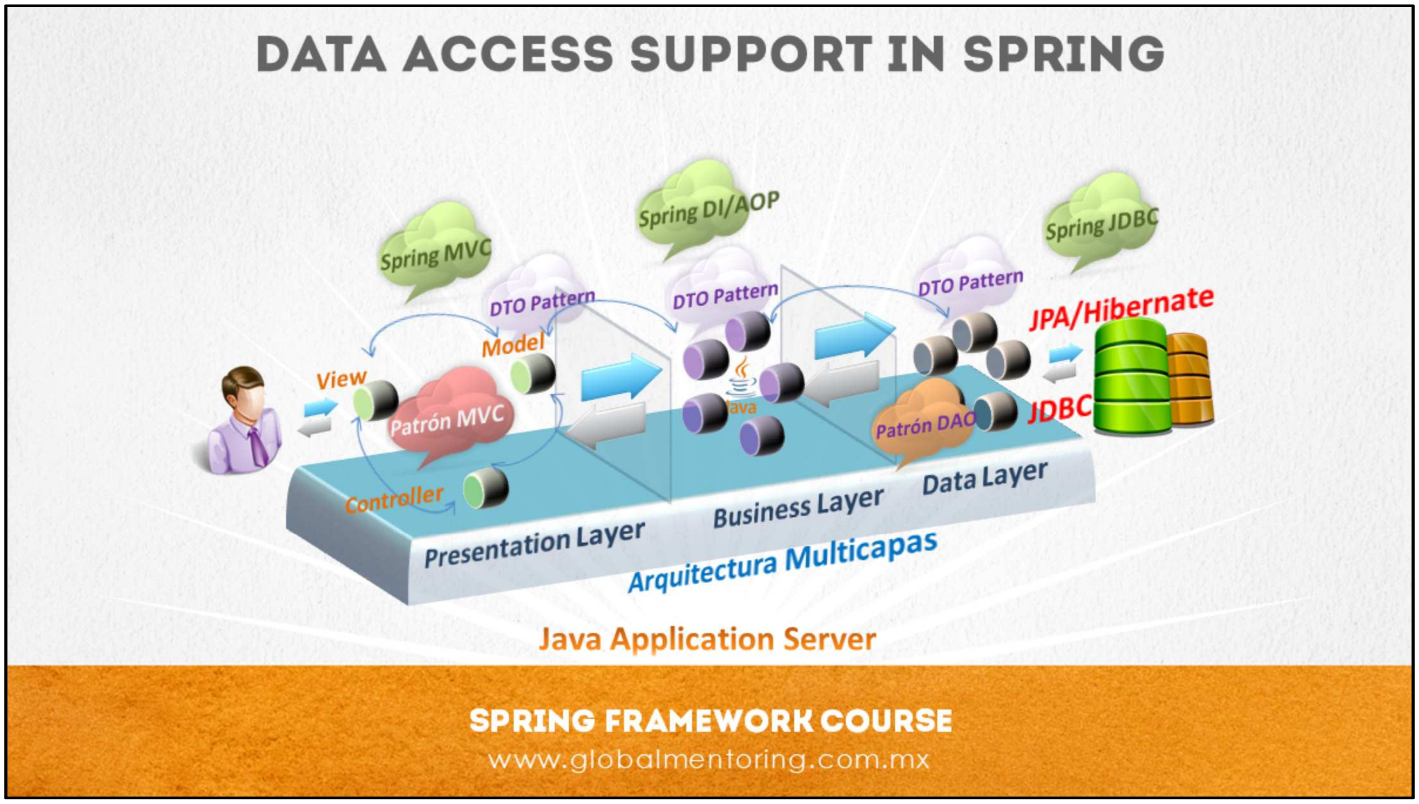
However, this is where Spring JDBC begins its participation. The DAO class extends from a template (Java class) called JdbcTemplate.

Extending this template may be somewhat intrusive to our code, however, Spring offers this solution to abstract the repetitive tasks when using the JDBC API, such as handling exceptions, opening and closing a database connection, among many small details that we will study in more detail.

The JdbcTemplate template in turn needs a DataSource object to execute the SQL queries. This object in turn is administered by Spring, with this we have the opportunity to configure a test DataSource and / or enable a DataSource for a productive environment.

Even in this last point the concept of interfaces is also used, since the DataSource object is behind an interface, and we must specify the implementation of it when configuring the bean in Spring.

With this we can observe the use of the JdbcTemplate template and its role in the context of the Spring Framework.



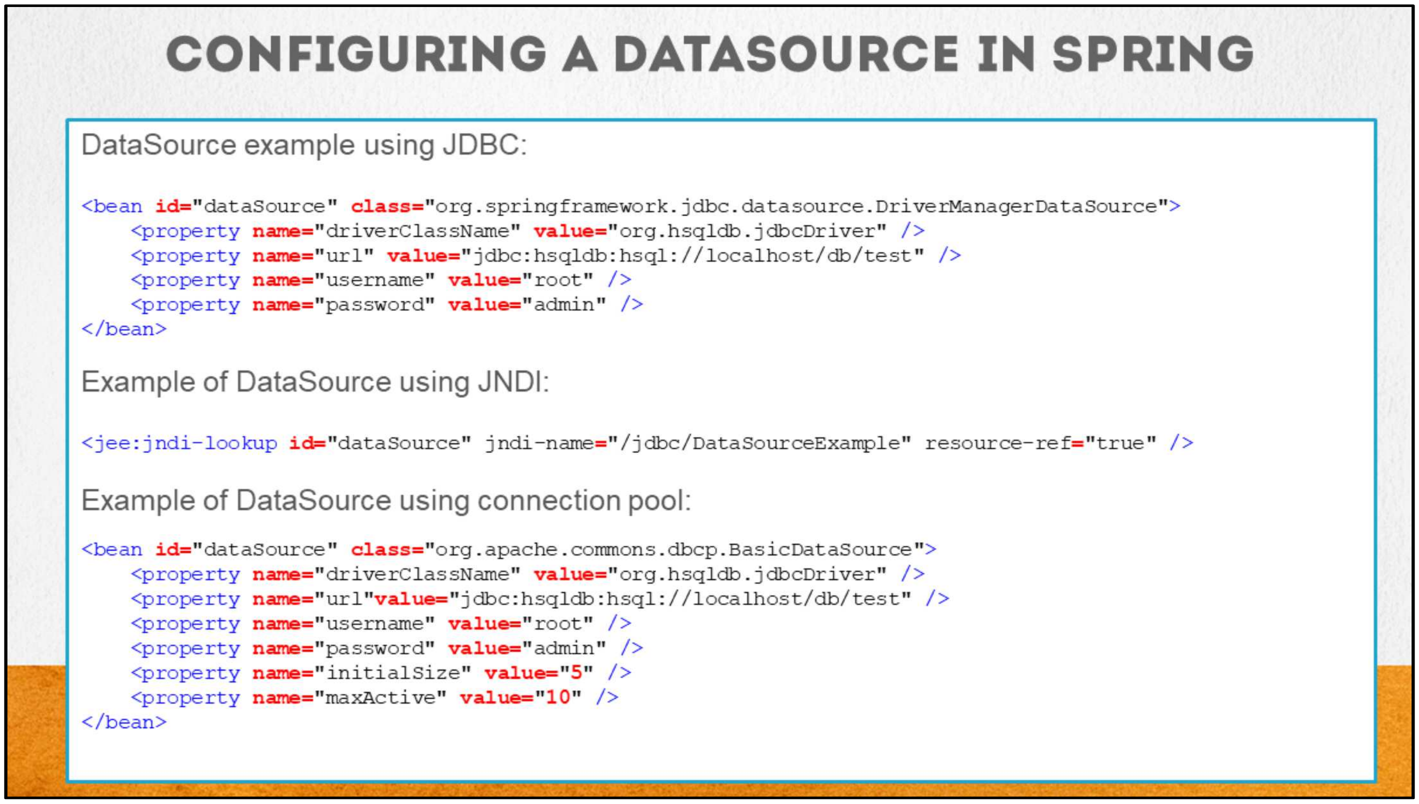
In the figure we can see the role of Spring JDBC in a Java Enterprise architecture.

Although this architecture has already been mentioned previously, the objective now is to point out the places where Spring JDBC applies and the design patterns that will help us efficiently apply this technology, such as Pattern DAO (Data Access Object) and DTO (Data Transfer) Object).

Each Spring template provides DAO classes to simplify access to data. With this, Spring provides a simplified way to create classes that execute the most common tasks such as insert, update, delete and select data, and depending on the technology the DAO class of Spring support will be selected.

In addition, these DAO support classes offer a set of methods to obtain questions directly related to the selected technology, for example, if we are using the Hibernate framework, we could obtain the Hibernate session through the DAO objects of Support of Hibernate.

In this way, these support classes allow us to have control at such a low level or at such a high level as we need, and thus Spring allows full control over the detail of our Java code and take full advantage of the power of access technology. data that we have selected.



CONFIGURING A DATASOURCE IN SPRING

DataSource example using JDBC:

```xml
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
    <property name="url" value="jdbc:hsqldb:hsql://localhost/db/test" />
    <property name="username" value="root" />
    <property name="password" value="admin" />
</bean>
```

Example of DataSource using JNDI:

```xml
<jee:jndi-lookup id="dataSource" jndi-name="/jdbc/DataSourceExample" resource-ref="true" />
```

Example of DataSource using connection pool:

```xml
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
    <property name="url"value="jdbc:hsqldb:hsql://localhost/db/test" />
    <property name="username" value="root" />
    <property name="password" value="admin" />
    <property name="initialSize" value="5" />
    <property name="maxActive" value="10" />
</bean>
```

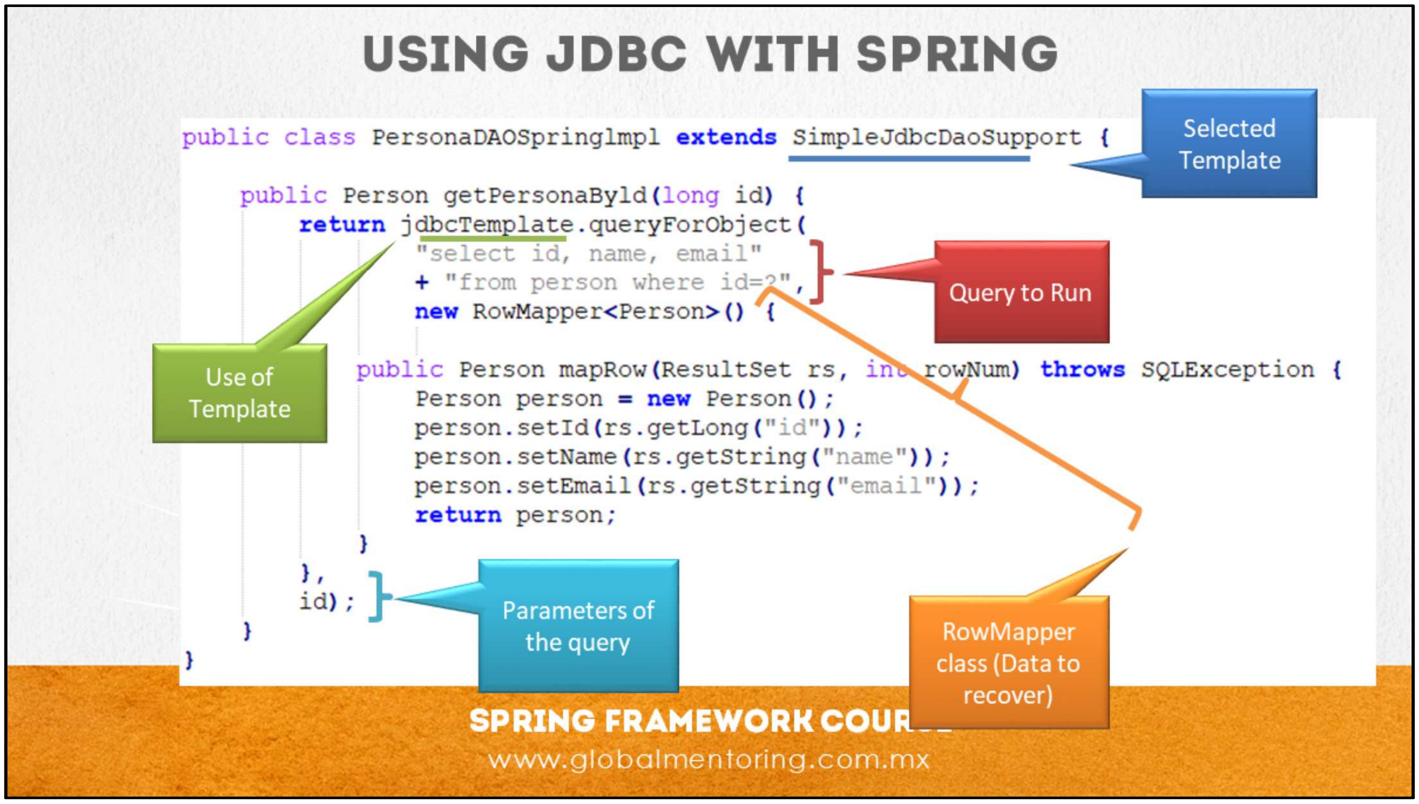
Regardless of what kind of Spring DAO support we use, we will need to configure a reference to a Data Source object. Spring offers several options to configure this object, such as:

- Data Sources defined via JDBC driver
- Data Sources defined via JNDI
- Data sources that use connection pool

The advantage of using a DataSource using JNDI is that most Java servers, including Tomcat, allow you to define connections using JNDI. This allows us to configure the connection completely independent of our Java application, since on the one hand the JNDI resource is defined in the application server, and on the other hand our application sends this resource.

In addition, the resources configured in the application server can usually be configured using a pool of connections and are usually modified and monitored directly by the administrators of the servers, delegating this responsibility outside of our Java application.

In case we do not have a way to use JNDI, we can use a connection pool supported by projects like Jakarta Commons Database Connection Pooling (DBCP) http://commons.apache.org/dbcp/



USING JDBC WITH SPRING

```
public class PersonaDAOSpringImpl extends SimpleJdbcDaoSupport {

    public Person getPersonaById(long id) {
        return jdbcTemplate.queryForObject(
                "select id, name, email"
                + "from person where id=?",
                new RowMapper<Person>() {

                public Person mapRow(ResultSet rs, int rowNum) throws SQLException {
                    Person person = new Person();
                    person.setId(rs.getLong("id"));
                    person.setName(rs.getString("name"));
                    person.setEmail(rs.getString("email"));
                    return person;
                }
            },
            id);
    }
}
```

Once we have knowledge of the templates, support for DAO's and finally, the configuration of a DataSource, it is time to start working with JDBC and Spring.

Because many applications created in Java use pure JDBC, we will study how Spring facilitates the use of this technology, without needing to learn a new framework such as Hibernate or JPA.

However, in later lessons we will study the integration between Spring and Hibernate, because JDBC can often result in very low level technology for our Java applications.

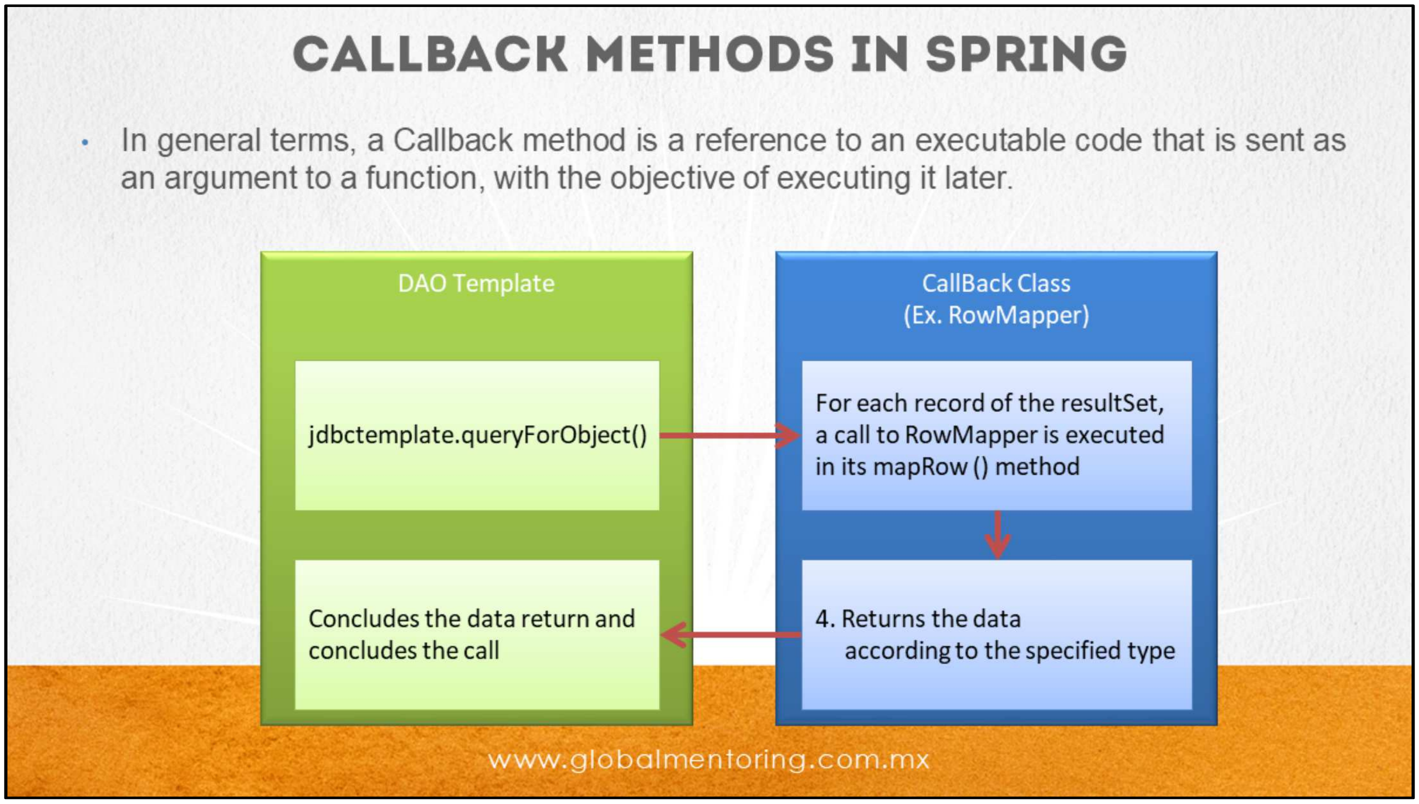
When we work directly with JDBC it is necessary to take care of too many tasks to perform a simple query based on data, tasks such as recovering a connection and closing the database connection, handling exceptions, among several other tasks.

Spring, through the concepts studied (Templates, DAOs Support, etc.) allows us to simplify and divide this task, allowing us to focus only on important tasks for our application.

As we can see in the figure we observe the elements mentioned above, such as the selection of the Spring template that will help us to abstract a lot from the repetitive and tedious code when using JDBC, allowing us to focus on relevant tasks, as in this case, recovering a object person.

Subsequently, we use the selected template and the queryForObject () method, providing 3 parameters (although it may vary). The first parameter is the query to execute, later the class that will allow us to recover the data, known as RowMapper, this we will review more in detail, and finally a parameter to filter our query to only the person we are looking for (id).

This configuration, as we can see, allows us to eliminate practically all the tedious code and that does not add value to our method, allowing us to focus on really relevant tasks when working with the JDBC API and database access.



CALLBACK METHODS IN SPRING

- In general terms, a Callback method is a reference to an executable code that is sent as an argument to a function, with the objective of executing it later.

DAO Template
jdbctemplate.queryForObject()
Concludes the data return and concludes the call
CallBack Class (Ex. RowMapper)
For each record of the resultSet, a call to RowMapper is executed in its mapRow () method
4. Returns the data according to the specified type
www.globalmentoring.com.mx

In general terms, a Callback method is a reference to an executable code that is sent as an argument to a function, with the objective of executing it later. In the case of Spring, it makes intensive use of Callback methods.

Spring JDBC applies this concept to retrieve the information of a resultSet, the result of having executed an SQL query. Once the template method is used, for example the queryForObject method, this method receives 3 parameters.

- SQL query to execute
- Callback code to be executed for each resultSet result of executing the query
- Parameters that complement the query to be executed (PreparedStatement)

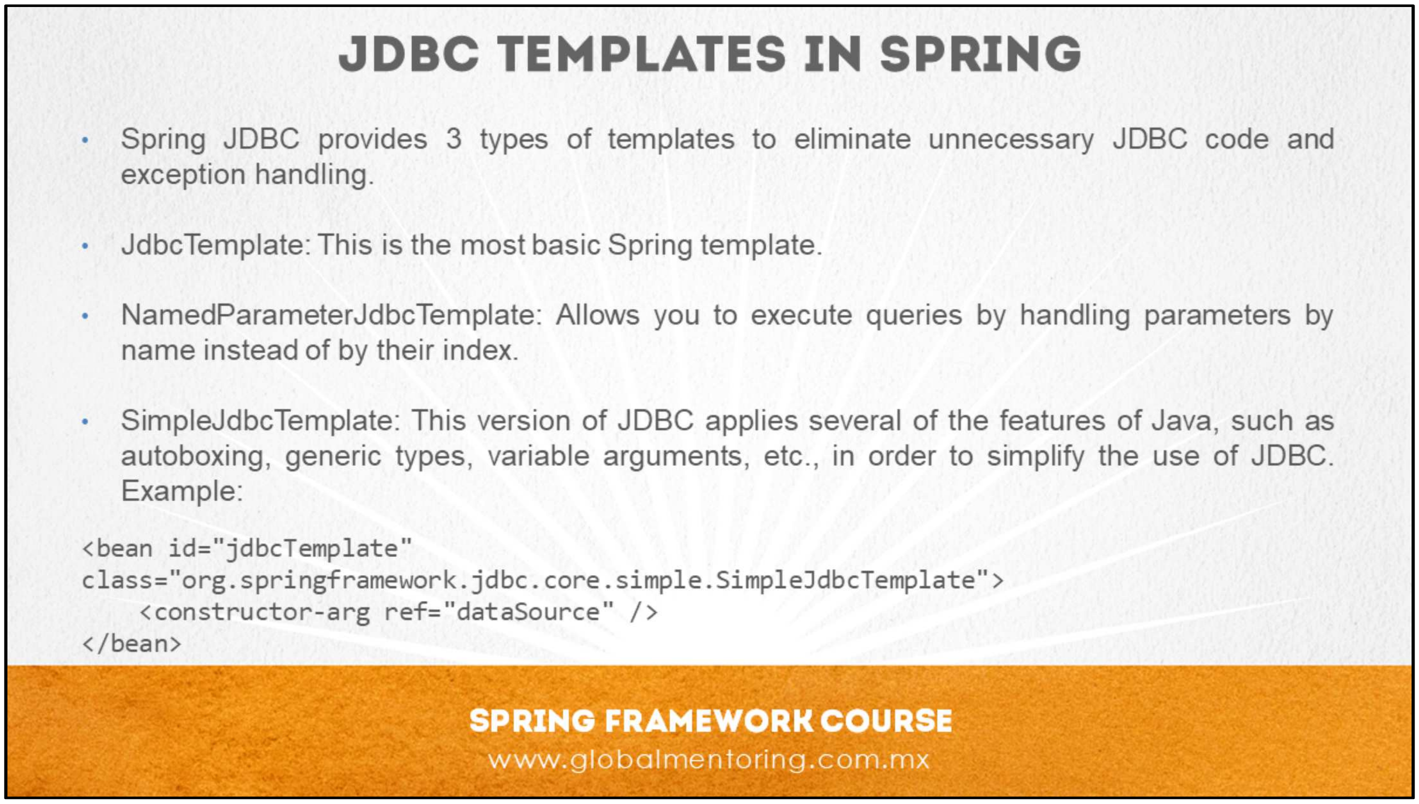
There are several forms and objects that Spring offers to apply this concept, however we will review one of the most common and used in the latest version of Spring, the RowMapper interface. In the figure we can see the configuration of a class of Callback type.

The class that contains the callback code must implement the RowMapper interface and add the fill of each object in the mapRow () method. Ex.`public class PersonaRowMapper implements RowMapper<Persona> {`

```
    public Persona mapRow(ResultSet rs, int rowNum) throws SQLException {
        Persona persona = new Persona();
        persona.setIdPersona(rs.getLong("id_persona"));
        persona.setNombre(rs.getString("nombre"));
        persona.setApellidoPaterno(rs.getString("ape_paterno"));
        return persona;
    }
}
```

El método de la lámina anterior quedaría así:

```
  public Persona getPersonaById(long id) {
        return jdbcTemplate.queryForObject(
                "select id_persona, nombre, ape_paterno"
                + "from persona where id_persona=?",
                new PersonaRowMapper(),
                id);
  }
```



JDBC TEMPLATES IN SPRING

- Spring JDBC provides 3 types of templates to eliminate unnecessary JDBC code and exception handling.
- JdbcTemplate: This is the most basic Spring template.
- NamedParameterJdbcTemplate: Allows you to execute queries by handling parameters by name instead of by their index.
- SimpleJdbcTemplate: This version of JDBC applies several of the features of Java, such as autoboxing, generic types, variable arguments, etc., in order to simplify the use of JDBC. Example:

```
<bean id="jdbcTemplate"
class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate">
    <constructor-arg ref="dataSource" />
</bean>
```

SPRING FRAMEWORK COURSE
www.globalmentoring.com.mx

The NamedParameterJdbcTemplate templates were integrated with SimpleJdbcTemplate, in order to take advantage of both templates.

However, in version 3.1 of Spring onward, support for versions prior to version 5 of Java is no longer supported, so there are only 2 templates to use JdbcTemplate and NamedParameterJdbcTemplate, leaving the SimpleJdbcTemplate template depreciated.

Below we can see a code example to use this template:

```
<bean id="jdbcTemplate"
    class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg ref="dataSource" />
</bean>
```

Once the bean is defined, we can inject it into our DAO class.

```
public class JdbcPersonDAO {
    ...
    @Autowired
    private JdbcTemplate jdbcTemplate;

    public int getCountPeople(){
        return jdbcTemplate.queryForInt("select count(*) from person");
    }
}
```

ONLINE COURSE

SPRING FRAMEWORK

By: Eng. Ubaldo Acosta

SPRING FRAMEWORK COURSE

www.globalmentoring.com.mx