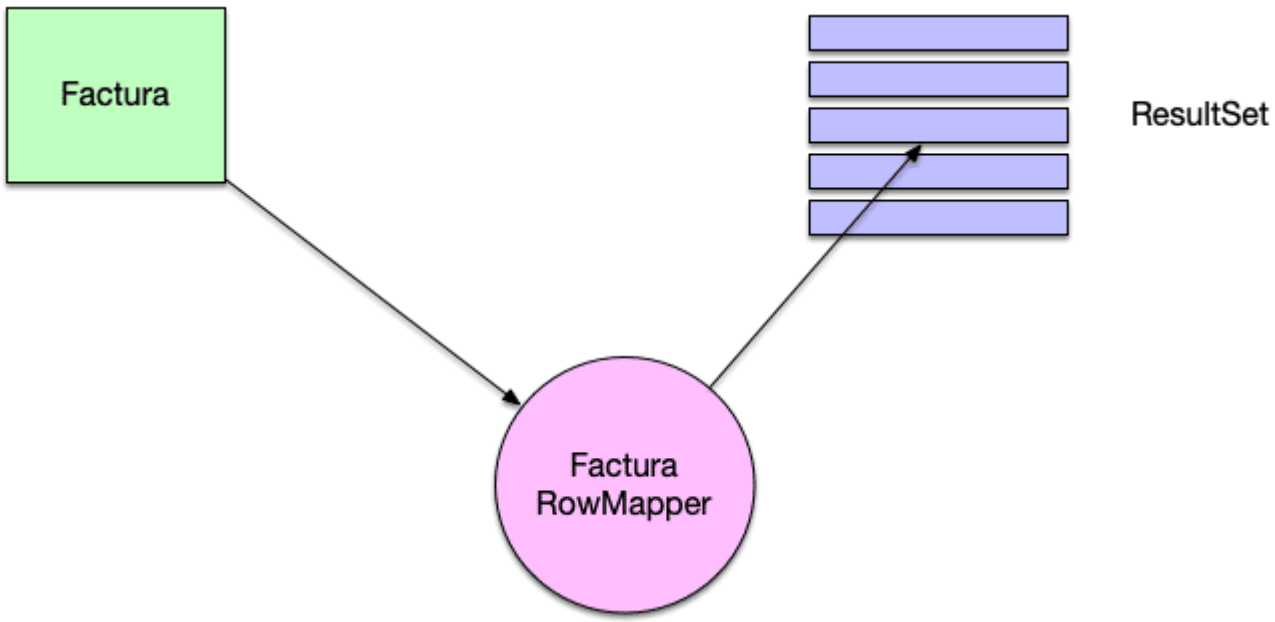
El concepto de Spring BeanPropertyRowMapper nos permite simplificar la forma con la que trabajamos con JDBCTemplates en Spring Framework . Normalmente cuando trabajamos con JDBCTemplates tenemos que diseñar una serie de consultas parametrizadas a nivel de un repositorio. Estas consultas parametrizadas son relativamente sencillas:

```
package com.arquitecturajava.spring01;

import java.sql.SQLException;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Repository;

@Repository
public class FacturaRepository {
	@Autowired
	private JdbcTemplate plantilla;

	public List<Factura> buscarTodas() throws SQLException,
ClassNotFoundException {

		return plantilla.query("select * from Facturas", new
FacturaRowMapper());

	}

	public void insertar(Factura factura) throws SQLException,
ClassNotFoundException {
```

```
                String sql = "insert into Facturas (concepto, importe)
values (?,?)";

                plantilla.update(sql, factura.getConcepto(),
factura.getImporte());
        }

}
```

La simplificación de Spring JDBC sobre el uso de JDBC de forma directa es más que claro . Ahora bien siempre quedan situaciones que mucha gente me pregunta si se pueden simplificar . En este pequeño bloque de código se puede ver. ¿Es necesario construir un FacturaRowMapper? .



Esta clase se encarga de mapear los datos de cada una de las filas del ResultSet a un objeto Factura de Java. El código de esta clase es relativamente sencillo pero hay que construirlo y normalmente para muchas clases:

```
package com.arquitecturajava.spring01;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class FacturaRowMapper implements RowMapper<Factura> {

        @Override
        public Factura mapRow(ResultSet rs, int rowNum) throws
SQLException {
                Factura f= new Factura();
                f.setNumero(Integer.parseInt(rs.getString("numero")));
                f.setConcepto(rs.getString("concepto"));
f.setImporte(Double.parseDouble(rs.getString("importe")));
                return f;
        }

}
```

Como vemos el código es muy muy sencilla y la pregunta que muchas veces los desarrolladores se hacen es si no podríamos eliminarlo. La realidad es que no lo podremos eliminar en todos los casos pero si en casos en los que estemos trabajando con objetos de negocio puros.

Spring BeanPropertyRowMapper y su uso

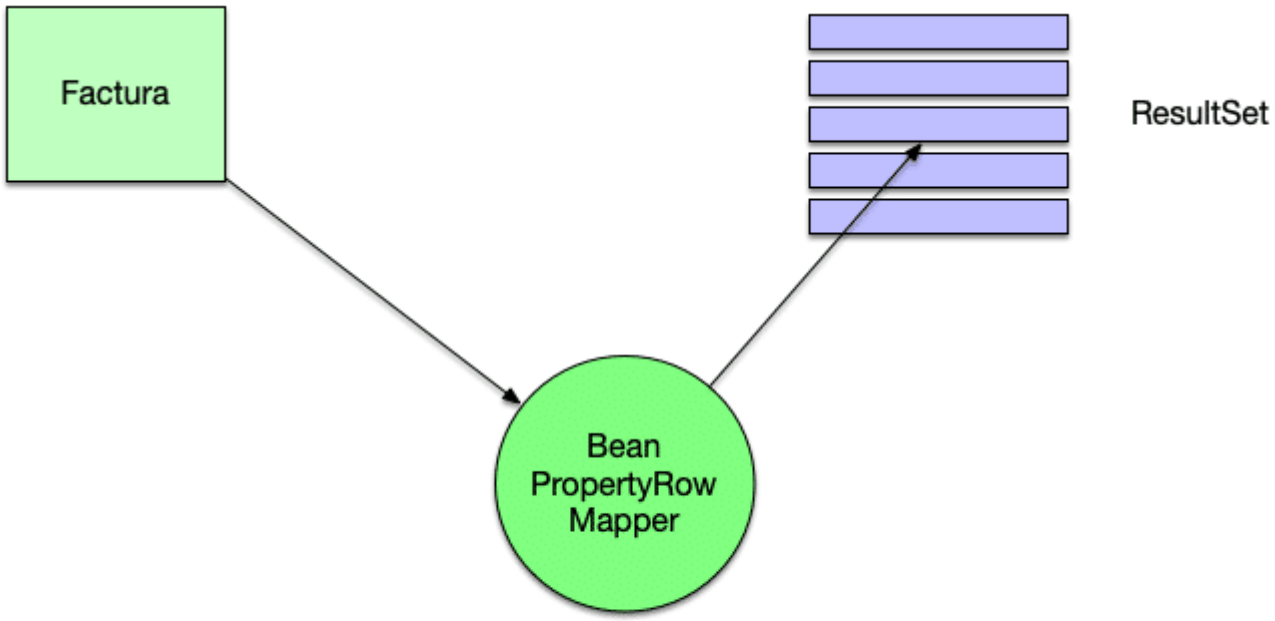
En estas situaciones podremos usar un Spring BeanPropertyRowMapper que asocia directamente cada una de las filas a una propiedad de una clase ya existente.

```
package com.arquitecturajava.spring01;

import java.sql.SQLException;
```

```
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Repository;

@Component
public class FacturaRepository {
	@Autowired
	private JdbcTemplate plantilla;

	public List<Factura> buscarTodas() throws SQLException,
ClassNotFoundException {

		return plantilla.query("select * from Facturas", new
BeanPropertyRowMapper<Factura>(Factura.class));

	}

	public void insertar(Factura factura) throws SQLException,
ClassNotFoundException {

		String sql = "insert into Facturas (concepto, importe)
values (?,?)";

		plantilla.update(sql, factura.getConcepto(),
factura.getImporte());
	}
```

}

De esta manera no será necesario construir un mapeador para cada una de las clases que tenemos y Spring lo procesará de forma totalmente transparente.



Integrado en Spring

Spring siempre tiene sus trucos y sus cosas que nos permiten optimizar los desarrollos eliminando código sobrante.

Otros artículos relacionados

- Spring AOP
- Spring Autowired
- Spring Reactor
- https://spring.io/