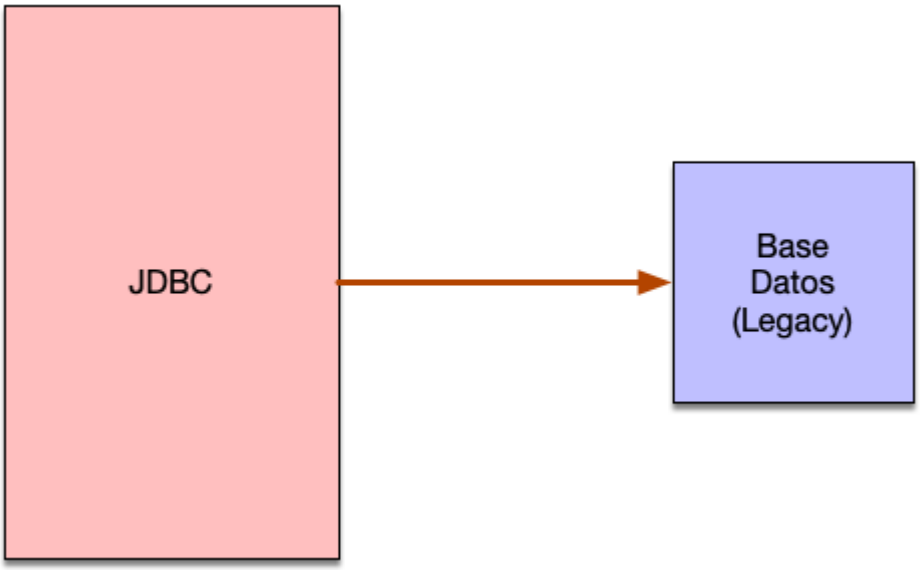
El uso de Spring JdbcTemplate es un clásico cuando hablamos de desarrollo de aplicaciones enterprise. En muchos casos el primer enfoque suele ser hacer uso de JPA para persistir la información en la base de datos o directamente de hibernate . Lamentablemente no todos los diseños entidad relación soportan este enfoque. Ya que a veces son muy antiguos o no han sido correctamente construidos.

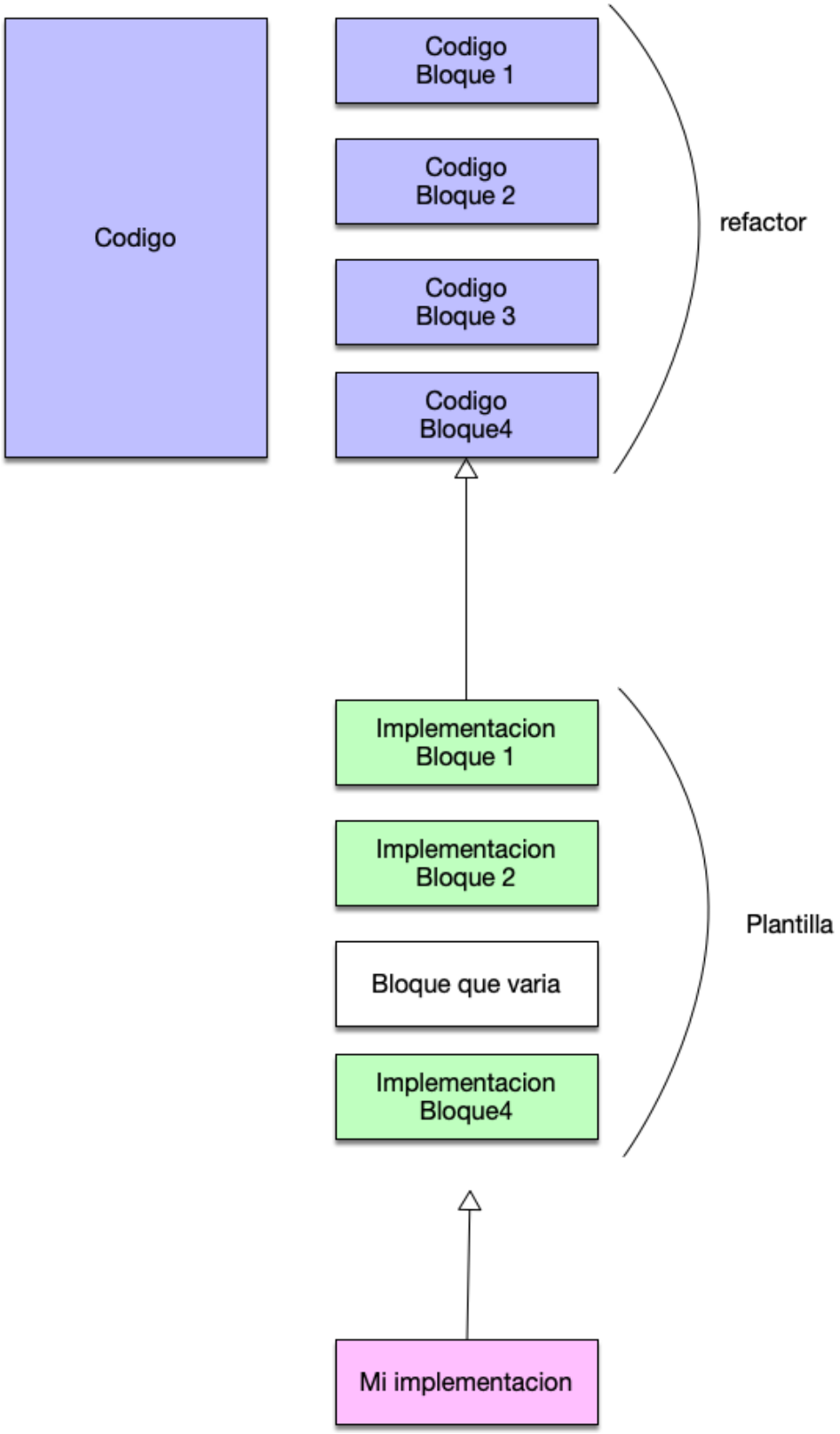
En ese caso una solución más directa es hacer uso del propio API de JDBC.
Lamentablemente sí hay un API que es pesada de trabajar es JDBC en muchos casos para lanzar una consulta de “select * from tabla” se necesitan decenas de lineas de código.



¿Cómo podemos cambiar esto?.

El patrón Template

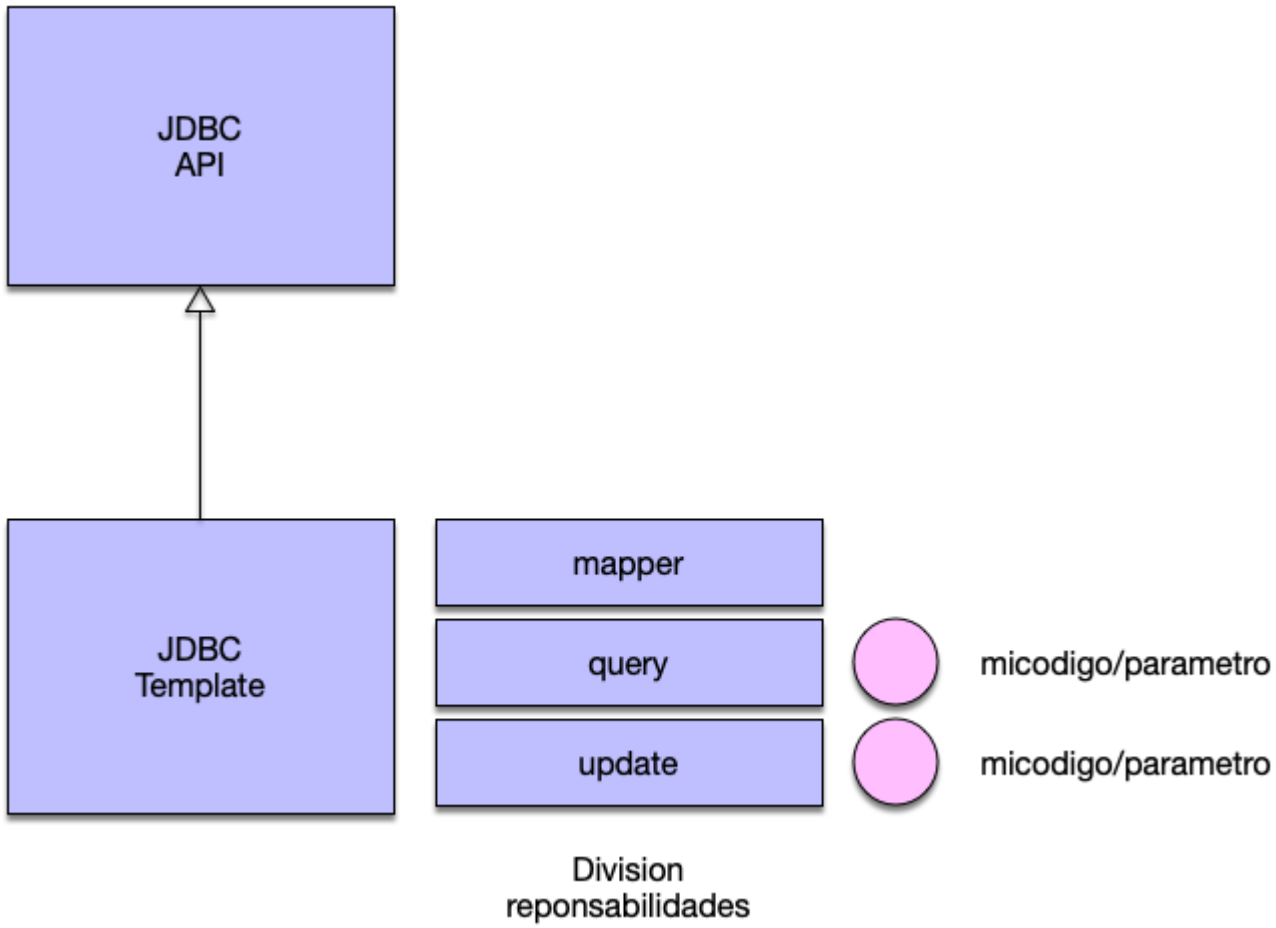
Para solventar este problema Spring Framework hace uso del patrón template. El patrón plantilla o template permite dividir una funcionalidad en partes reutilizables y realizar una implementación por defecto de la mayoría de ellos.


Codigo
Codigo Bloque 1
Codigo Bloque 2
Codigo Bloque 3
Codigo Bloque4
refactor
Implementacion Bloque 1
Implementacion Bloque 2
Bloque que varia
Implementacion Bloque4
Plantilla
Mi implementacion

De esa forma el desarrollador solo tiene que implementar la pequeña lógica que varía .

Usando SpringJdbcTemplate

En nuestro caso esa es la operativa con la que nos encontramos . La mayor parte del código JDBC es muy repetitivo. Los JDBCTemplates nos permiten eliminar la mayor parte de ese código repetido y centrarnos en lo que verdaderamente importa.



En el siguiente ejemplo creamos un JDBCTemplate a través del patrón repositorio y el código es muy sencillo y se centra en la funcionalidad que realmente importa.

```
package com.arquitecturajava.app1;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;



@Repository
public class PersonaRepository {

	@Autowired
	private JdbcTemplate plantilla;

	public List<Persona> buscarTodos() {

		return plantilla.query("select * from Personas", new
PersonaMapper());

	}


	public void  insertar(Persona persona) {

		 plantilla.update("insert into Personas values
(?,?,?)",persona.getNombre(),persona.getApellidos(),persona.getEdad())
;

	}
```

```
        public void borrar(Persona persona) {

                plantilla.update("delete from Personas where
nombre=?",persona.getNombre());
        }

}
```

Spring JdbcTemplate Mappers

Nos falta el código del mapper que se encarga de convertir un ResultSet en una lista de objetos .

Vamos a verlo:

```
package com.arquitecturajava.app1;

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

import org.springframework.jdbc.core.RowMapper;

public class PersonaMapper implements RowMapper<Persona> {

        @Override
        public Persona mapRow(ResultSet rs, int rowNum) throws
SQLException {
                return new
Persona(rs.getString("nombre"),rs.getString("apellidos"),rs.getInt("ed
ad"));
```

```
        }

}
```

De esta forma podemos eliminar prácticamente todo el código JDBC de nuestros clases y centrarnos en la funcionalidad que realmente importa. El uso Spring JdbcTemplate nos puede ayudar en la gestión de muchas de nuestras bases de datos legacy.

Otros artículos relacionados

1. Spring Boot JSP y su configuración
2. Uso de Spring Properties y encriptación
3. @RepositoryRestResource y Spring Framework
4. Spring @import , organizando Spring framework
5. El patrón plantilla