El concepto de Java Mapping hace referencia normalmente a operaciones de mapeo o transformación que necesitamos realizar sobre nuestras clases. El manejo de DTOs en Arquitecturas REST es muy habitual y la necesidad de librerías de mapeo es cada día mayor. MapStrut es una de las librerías que esta en estos momentos mas de moda y permite una transformación entre objetos de forma muy sencilla. Vamos a ver unos ejemplos basándonos en ella.

MapStrut configuración

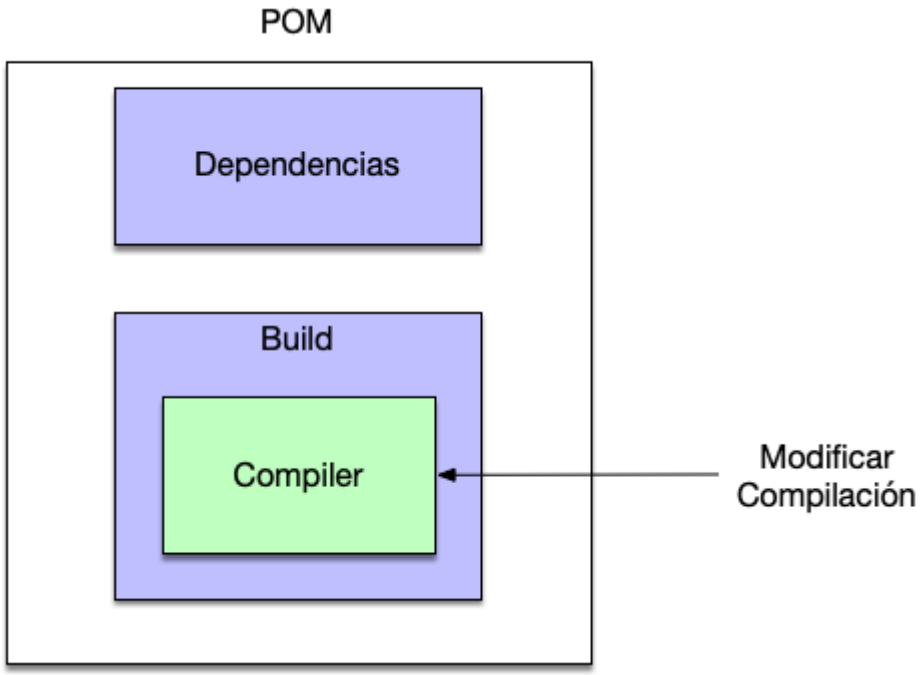
El primer paso es abordar la configuración del framework para que sea capaz de generar dinámicamente los mapeadores . En este caso no solo necesitamos añadir las dependencias de Maven sino que necesitamos modificar el proceso de build para asegurarnos que se procesan las clases y se generan los mapeadores que necesitemos.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.aruitecturajava</groupId>
        <artifactId>javamapping</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <properties>
<org.mapstruct.version>1.3.1.Final</org.mapstruct.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.mapstruct</groupId>
                        <artifactId>mapstruct</artifactId>
```

```
                        <version>${org.mapstruct.version}</version>
                </dependency>
                <dependency>
                <groupId>org.junit.jupiter</groupId>
                <artifactId>junit-jupiter-engine</artifactId>
                <version>5.5.2</version>
                <scope>test</scope>
        </dependency>
        </dependencies>
        <build>
                <plugins>
                        <plugin>
<groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-
plugin</artifactId>
                                <version>3.5.1</version> <!-- or newer
version -->
                                <configuration>
                                        <source>1.8</source> <!--
depending on your project -->
                                        <target>1.8</target> <!--
depending on your project -->
                                        <annotationProcessorPaths>
                                                <path>
<groupId>org.mapstruct</groupId>
<artifactId>mapstruct-processor</artifactId>
<version>${org.mapstruct.version}</version>
                                                </path>
                                                <!-- other annotation
processors -->
                                        </annotationProcessorPaths>
```

```
                        </configuration>
                    </plugin>
                </plugins>
            </build>


</project>
```

Como se puede observar no solo hemos añadido dependencias sino que ademas hemos modificado el proceso de build añadiendo operaciones de pre procesamiento.



Java Mapping y MapStrut

Una vez que tenemos hecho esto el siguiente paso es generar las clases que queremos mapear. Vamos a usar Persona y PersonaDTO para ver un primer mapeo prácticamente automático entre ambas.

```
package com.arquitecturajava;
```

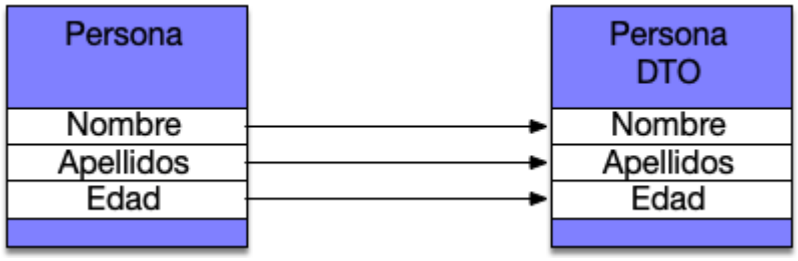
```
public class Persona {

	private String nombre;
	private String apellidos;
	private int edad;
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
	public Persona(String nombre, String apellidos, int edad) {
		super();
		this.nombre = nombre;
		this.apellidos = apellidos;
		this.edad = edad;
	}
	public Persona() {
		super();
```

```
        }
}
```

```
package com.arquitecturajava;

public class PersonaDTO {

        private String nombre;
        private String apellidos;
        private int edad;
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public String getApellidos() {
                return apellidos;
        }
        public void setApellidos(String apellidos) {
                this.apellidos = apellidos;
        }
        public int getEdad() {
                return edad;
        }
        public void setEdad(int edad) {
                this.edad = edad;
        }
        public PersonaDTO(String nombre, String apellidos, int edad) {
                super();
                this.nombre = nombre;
                this.apellidos = apellidos;
```

```
                this.edad = edad;
        }
        public PersonaDTO() {
                super();
        }
}
```

Ambas clases son prácticamente idénticas y con ellas queremos hacer un mapeo sencillo.

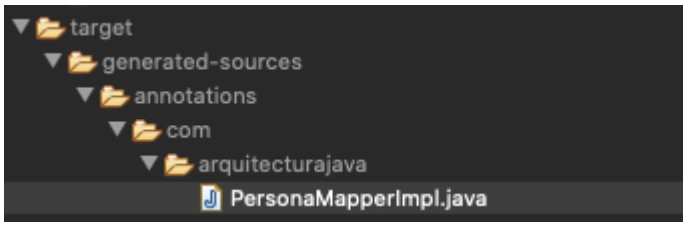


Es aquí donde el concepto Java Mapping y MapStrut brilla ya que nos será suficiente con crear una clase de mapear como la siguiente:

```
package com.arquitecturajava;

import org.mapstruct.Mapper;
import org.mapstruct.factory.Mappers;

@Mapper
public interface PersonaMapper {

        PersonaMapper INSTANCIA=
Mappers.getMapper(PersonaMapper.class);
        PersonaDTO personaToPersonaDto(Persona persona);
}
```

En esta clase MapStrut será capaz de mapear ambos objetos de forma transparente . Eso sí

recordemos que tendremos que ejecutar previamente maven package para que la solución empaquete y a nivel de compilación se generen las nuevas clases de mapeo.



Una vez generada la clase de mapeo podemos mostrar su código para entender que es lo que hace dinámicamente MapStrut.

```
package com.arquitecturajava;

/*
@Generated(
    value = "org.mapstruct.ap.MappingProcessor",
    date = "2020-02-12T10:49:28+0100",
    comments = "version: 1.3.1.Final, compiler: javac, environment:
Java 11.0.1 (Oracle Corporation)"
)
*/
public class PersonaMapperImpl implements PersonaMapper {

    @Override
    public PersonaDTO personaToPersonaDto(Persona persona) {
        if ( persona == null ) {
            return null;
        }

        PersonaDTO personaDTO = new PersonaDTO();

        personaDTO.setNombre( persona.getNombre() );
```

```
        personaDTO.setApellidos( persona.getApellidos() );
        personaDTO.setEdad( persona.getEdad() );

        return personaDTO;
    }
}
```

Cómo se puede observar ha realizado la tediosa operación que nosotros realizaríamos a mano pero de una forma totalmente automatizada . Es momento de construir una prueba unitaria y probar que el mapeo funciona.

```
 package com.arquitecturajava;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class MapStrutTest {

        @Test
         public void testMapeo() {
                          Persona persona= new
Persona("pepe","perez",20);
                        PersonaDTO personaDto=
PersonaMapper.INSTANCIA.personaToPersonaDto(persona);
                        assertEquals("pepe",personaDto.getNombre());
assertEquals("perez",personaDto.getApellidos());
                        assertEquals(20,personaDto.getEdad());
         }
}
```

Mappings y Anotaciones

Acabamos de automatizar nuestro primer mapeo entre Persona y PersonaDTO . Esta librería es muy flexible y permite conversiones muy complejas . Imagemonos que disponemos de dos clases que se parecen pero no son iguales.

```
package com.arquitecturajava;

public class Cliente {

	private String nombreCliente;
	private String apellidosCliente;
	private int edad;
	public Cliente(String nombreCliente, String apellidosCliente,
int edad) {
		super();
		this.nombreCliente = nombreCliente;
		this.apellidosCliente = apellidosCliente;
		this.edad = edad;
	}
	public Cliente() {
		super();
	}

	public String getNombreCliente() {
		return nombreCliente;
	}
	public void setNombreCliente(String nombreCliente) {
		this.nombreCliente = nombreCliente;
	}
	public String getApellidosCliente() {
```

```
                return apellidosCliente;
        }
        public void setApellidosCliente(String apellidosCliente) {
                this.apellidosCliente = apellidosCliente;
        }
        public int getEdad() {
                return edad;
        }
        public void setEdad(int edad) {
                this.edad = edad;
        }
}
```

En este caso la clase Cliente y la clase Persona son prácticamente idénticas pero no coinciden los nombres de los campos . Puedo diseñar un mapper que nos permita mapear estos campos a través del uso de las anotaciones que la librería soporta.

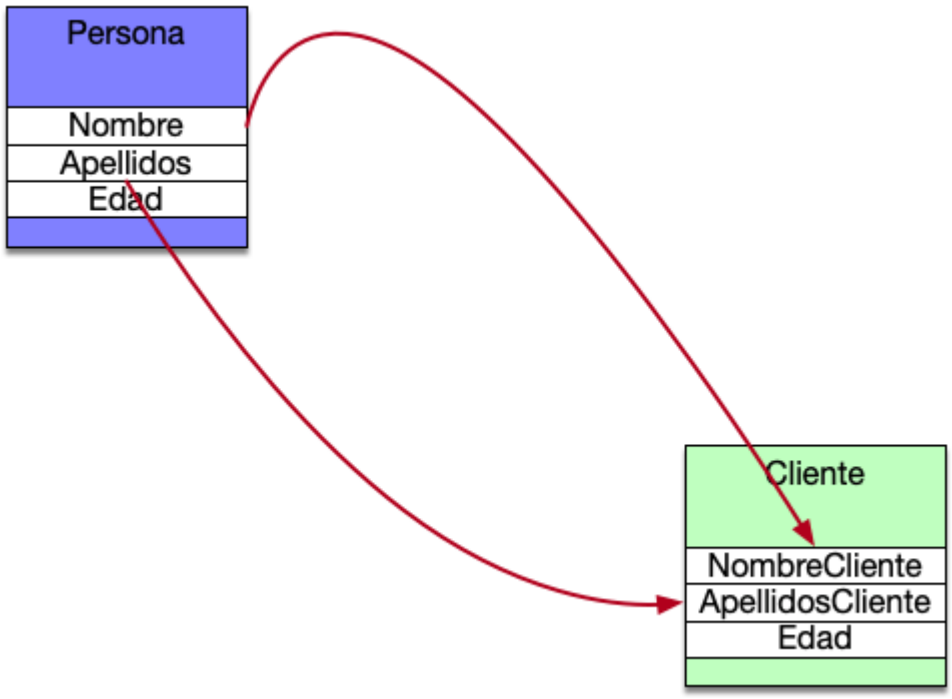
```
package com.arquitecturajava;

import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.factory.Mappers;

@Mapper
public interface PersonaClienteMapper {

        PersonaClienteMapper INSTANCIA=
Mappers.getMapper(PersonaClienteMapper.class);
        @Mapping(source = "nombre", target = "nombreCliente")
        @Mapping(source = "apellidos", target = "apellidosCliente")
        Cliente personaToCliente(Persona persona);
```

```
}
```

De esta forma conseguiré que los mapeos funcionen incluso con campos disjuntos. Veamos el test a ejecutar

```
        @Test
         public void testMapeoCliente() {
                         Persona persona= new
Persona("pepe","perez",20);
                        Cliente cliente=
PersonaClienteMapper.INSTANCIA.personaToCliente(persona);
assertEquals("pepe",cliente.getNombreCliente());
assertEquals("perez",cliente.getApellidosCliente());
                        assertEquals(20,cliente.getEdad());
         }
```

El concepto de Java Mapping es un concepto fundamental y cada día lo necesitaremos más.



MapStrut es una librería de referencia en este ambito.

Otros artículos relacionados

- JPA DTO
- REST DTO
- Ejemplo JPA
- MapsStrut (Paradigma)

Cecilio Álvarez Caules

Cecilio Álvarez Caules Oracle Java Certified Architech