POJO to JSON es una de las operaciones más habituales que tenemos que hacer hoy en día en la programación Java. ¿Cómo podemos realizar estas operaciones de forma sencilla con las librerías de Java ? . En este caso voy a elegir Jackson como librería de apoyo para implementar estas funcionalidades. Como siempre mostramos las dependencias de nuestro proyecto Maven.

```
<dependencies>
		<dependency>
			<groupId>com.fasterxml.jackson.core</groupId>
			<artifactId>jackson-databind</artifactId>
			<version>2.11.0</version>
		</dependency>
</dependencies>
```

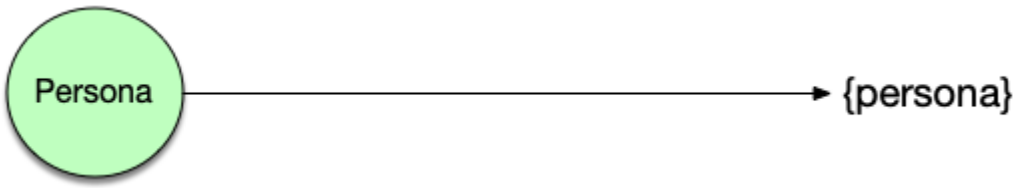
POJO to JSON

Una vez instalada la dependencia nos podemos construir una clase sencilla que vayamos a serializar a JSON.

```
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();
	}
}
```

Es momento de crear el programa principal que se encarga de convertir un objeto Persona a JSON usando Jackson.




```
package com.arquitecturajava;
```

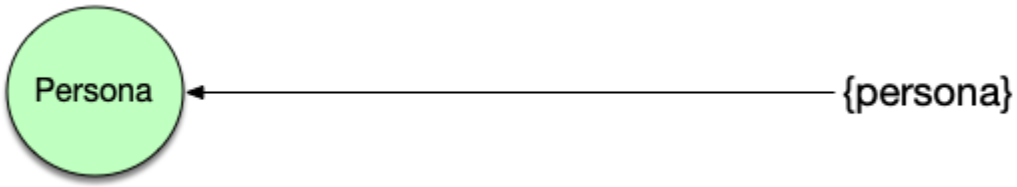
```
import java.io.File;
import java.io.IOException;
import java.util.List;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;

public class Principal {

	public static void main(String[] args) {

		try {

			ObjectMapper mapeador = new ObjectMapper();
			Persona p = new Persona("pepe", "perez", 25);
			 mapeador.writeValue(new
File("mijson.json"),p);


		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}
}
```

Este sencillo programa se encargará de convertir nuestro objeto Java a JSON y generarnos un fichero en la carpeta principal del proyecto con el contenido en el nuevo formato.

```
{"nombre":"pepe","apellidos":"perez","edad":25}
```

Leyendo JSON

Es momento de realizar la operación contraria y convertir un fichero JSON en un POJO de Java.



Para ello en vez de utilizar el método write para escribir utilizaremos el método read para leer un fichero que tenemos en la misma carpeta con otro contenido

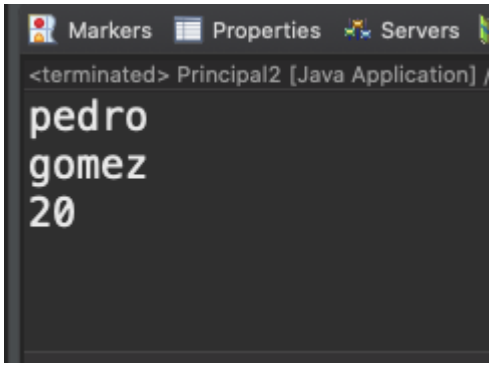
```
{
  "nombre": "pedro",
  "apellidos": "gomez",
  "edad": 20
}
```

El código también es muy sencillo:

```
package com.arquitecturajava;

import java.io.File;
import java.io.IOException;
import java.util.List;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;

public class Principal2 {

        public static void main(String[] args) {
```

```
                try {

                        ObjectMapper mapeador = new ObjectMapper();
                Persona persona = mapeador.readValue(new
File("persona.json"), Persona.class);

                        System.out.println(persona.getNombre());
                        System.out.println(persona.getApellidos());
                        System.out.println(persona.getEdad());
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }
}
```

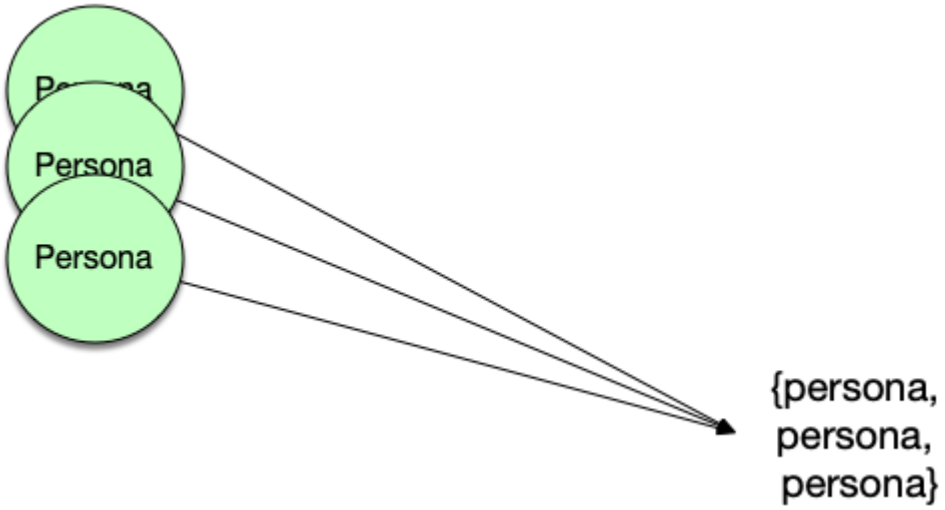
De esta manera leemos el fichero persona.json y lo convertimos a un objeto Persona . Recordemos que necesitaremos un constructor por defecto para que estas operaciones se puedan realizar . El resultado aparecerá en la consola de Eclipse.



POJO JSON y Colecciones

En la mayoría de las ocasiones no queremos leer o escribir un solo objeto en fichero sino

que lo que queremos es trabajar con una colección de elementos .



Veamos el mismo ejemplo pero con una lista de Personas.

```
package com.arquitecturajava;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Principal5 {

        public static void main(String[] args) {

                try {

                        ObjectMapper mapeador = new ObjectMapper();
```

```
                        Persona p1 = new Persona("pepe", "perez", 25);
                        Persona p2 = new Persona("antonio", "sanchez",
30);
                        Persona p3 = new Persona("ana", "jimenez",
40);
                        List<Persona> lista= new ArrayList<Persona>();
                        lista.add(p1);
                        lista.add(p2);
                        lista.add(p3);
                        mapeador.writeValue(new
File("milista.json"),lista);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        System.out.println(e);
                }

        }
}
```

Esto nos generará un fichero con los datos en formato JSON de nuestra lista de elementos.

```
[{"nombre":"pepe","apellidos":"perez","edad":25},{"nombre":"antonio","
apellidos":"sanchez","edad":30},{"nombre":"ana","apellidos":"jimenez",
"edad":40}]
```

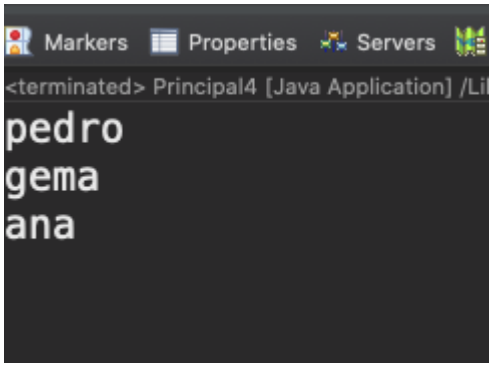
JSON to POJO y listas

Nos queda la operación contraria leer un fichero JSON y convertirlo en una lista de Personas a recorrer.

```
package com.arquitecturajava;
```

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;

public class Principal4 {

	public static void main(String[] args) {
		try {
			ObjectMapper mapeador = new ObjectMapper();

			//JSON file to Java object
			List<Persona> lista = mapeador.readValue(new
File("personas.json"),new TypeReference<List<Persona>>() {});
			for (Persona p: lista) {
				System.out.println(p.getNombre());
			}
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}
```

En este caso el código es un poco más peculiar porque necesita usar la clase TypeReference para saber de que tipo son los objetos que vamos a leer , mostramos los nombres en la consola.



Otros artículos relacionados:

- REST DTO y JSON
- JSON Viewer
- JSON Web Tokens