Cómo mapear json a objetos Java con Jackson ObjectMapper

1. Declarar dependencia

El primer paso es declarar la dependencia en el proyecto, en éste caso usando *maven*, en el fichero `pom.xml` añadimos:

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.4.4</version>
</dependency>
```

Hecho esto, ya es posible usar la librería en el proyecto.

2. Introducción a Jackson

Veamos una guía de uso rápido de jackson. Para los siguientes ejemplos supondremos la siguiente clase:

```
// Nota: Para atributos públicos, no es necesario usar getters y setters.
public class MiClase {
  public String nombre;
  public int edad;
  // Nota: Si los campos son private o protected, es obligatorio usar getters y
setters.
  // Es recomendable crear el constructor por defecto
}
```

Supongamos también el siguiente json, almacenado en un fichero `mijson.json`:

```
{
   "nombre":"Alicia",
   "edad":13
}
```

Es necesario crear un `ObjectMapper`, y lo típico es hacerlo estático para re-utilizarlo a lo largo de la aplicación. Un buen lugar para él sería una clase `Constanty` declarar el `ObjectMapper` así:

```
public static final ObjectMapper JSON_MAPPER = new ObjectMapper();
```

3. Json a Objeto Java (Des-Serializar)

Para des-serializar el `json` y crear el objeto en Java:

```
MiClase objeto = JSON_MAPPER.readValue(new File("mijson.json", MiClase.class));
// o
MiClase objeto = JSON_MAPPER.readValue(new URL("http://ruta/a/mijson.json",
MiClase.class));
```

4. Objeto Java a Json (Serializar)

Para realizar el proceso inverso, basta con:

```
JSON_MAPPER.writeValue(new File("mijson.json"), objeto);
// ó:
byte[] jsonBytes = JSON_MAPPER.writeValueAsBytes(objeto);
// ó:
String jsonString = JSON_MAPPER.writeValueAsString(objeto);
```

5. Generalizar el tipo de objeto a des-serializar

Al trabajar con una API, serializar y des-serializar objetos es una tarea común, una forma de generalizar el proceso puede ser la siguiente.

Supongamos que nuestra api devuelve arrays de objetos, por ejemplo una lista de Personas, una lista de productos etc. El modelo en Java sería el siguiente:

```
public class Persona {

    private String Nombre;
    private String Apellidos;
    private int DNI;

    public Persona() {}

    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 getDNI() {
        return DNI;
    }

    public void setDNI(int DNI) {
        this.DNI = DNI;
    }
}

public class Producto {
    private String Nombre;
    private String Modelo;
    private int precio;
    private float valoracion;

    public Producto() {}

    public String getNombre() {
        return Nombre;
    }

    public void setNombre(String Nombre) {
        this.Nombre = Nombre;
    }

    public String getModelo() {
        return Modelo;
    }

    public void setModelo(String Modelo) {
        this.Modelo = Modelo;
    }

    public int getPrecio() {
        return precio;
    }

    public void setPrecio(int precio) {
        this.precio = precio;
    }

    public float getValoracion() {
        return valoracion;
    }
```

```
    public void setValoracion(float valoracion) {
        this.valoracion = valoracion;
    }
}
```

Los arrays en json:

```
// Personas
[
   {
      "Nombre":"Bob",
      "Apellidos":"BobBob",
      "DNI": 123456789
   },
   {
      "Nombre":"Alice",
      "Apellidos":"Alice",
      "DNI": 123456789
   },
   {
      "Nombre":"Foo",
      "Apellidos":"bar",
      "DNI": 123456789
   }
]
// Productos
[
   {
      "Nombre":"Tele",
      "Modelo":"modelo1",
      "Precio": 120,
      "Valoracion": 2.5
   },
   {
      "Nombre":"Tele2",
      "Modelo":"Modelo2",
      "Precio": 150,
      "Valoracion": 5
   },
   {
      "Nombre":"Tele3",
      "Modelo":"Modelo3",
      "Precio": 520,
      "Valoracion": 5
   }
]
```

Con estos datos, queremos des-serializar el `json` en un `ArrayList` del tipo de clase que sea, en éste caso `ArrayList<Personas>` y `ArrayList<Producto>`. La forma de hacerlo sería:

```
ArrayList<Persona> personas = JSON_MAPPER.readValue(new File("personas.json"),

JSON_MAPPER.getTypeFactory().constructCollectionType(ArrayList.class,
Persona.class));

// Para productos

ArrayList<Producto> productos = JSON_MAPPER.readValue(new
File("productos.json"),

JSON_MAPPER.getTypeFactory().constructCollectionType(ArrayList.class,
Producto.class));
```