BOLETÍN 01.03. JSON CON JAVA

JSON CON JAVA

Ejercicio 1. Creación de un archivo JSON

A partir de la clasificación de la liga de baloncesto ACB:

Equipo	jugados	victorias	derrotas	favor	contra	diferencia
Real Madrid	4	4	0	374	311	63
Baskonia	4	3	1	346	320	26
Bàsquet Girona	4	3	1	353	333	20
UCAM Murcia	4	3	1	340	322	18
Valencia Basket	4	3	1	346	330	16
Barça	4	3	1	349	335	14
Surne Bilbao Basket	4	3	1	322	310	12
Joventut Badalona	4	3	1	329	319	10
Monbus Obradoiro	4	2	2	320	299	21
BAXI Manresa	4	2	2	350	351	-1
Dreamland Gran Canaria	4	2	2	312	338	-26
Unicaja	4	1	3	335	333	2
Río Breogán	4	1	3	314	328	-14
MoraBanc Andorra	4	1	3	310	329	-19
Lenovo Tenerife	4	1	3	317	353	-36
Casademont Zaragoza	4	1	3	317	354	-37
Coviran Granada	4	0	4	353	382	-29
Zunder Palencia	4	0	4	290	330	-40

Crea un documento JSON llamado ***clasificación.json*** con al menos 4 equipos.

Ejercicio 2. Lectura de un archivo JSON con Java Script API.

A partir del documento JSON anterior, copia el archivo JSON en un documento JavaScript para proceder a su lectura y que devuelva los datos del Obradoiro, suponiendo que es el primero de la lista.

Emplea el método *eval* que recoge un objeto de tipo Reader. Usa una **clase con buffer creada con la API de Java NIO.2**.

Como plantilla, emplea el ejemplo de los apuntes.

```
var poeta = {

    "nombre": "Sylvia",
    "apellidos": "Plath",
    "estaViva": false,
    "edad": 30,
    "direccion": {
        "direccionCalle": "21 2nd Street",
```

```
        "ciudad": "New York",
        "estado": "NY",
        "codigoPostal": "10021-3100"
    },
    "telefonos": [
        {
            "tipo": "casa",
            "numero": "212 555-1234"
        },
        {
            "tipo": "oficina",
            "numero": "646 555-4567"
        }
    ],
    "hijos": [],
    "marido": null

};

print(poeta.nombre);
print(poeta.apellidos);
print(poeta.direccion.ciudad);
print(poeta.telefonos[1].numero);
```

```
Monbus Obradoiro
Jugados: 4
Ganados: 2
Perdidos: 2
Puntos a favor: 320
Puntos en contra: 299
Diferencia de puntos: 21
```

Ejercicio 3. Ejercicio con JSON-B. Examen.

Crea un proyecto Maven con una sencilla clase ***Examen*** que contenga los siguientes atributos:

- **materia**: de tipo *String*.
- **fecha**: de tipo *LocalDateTime*.
- **participantes**: de tipo *List* de *String* con los nombres de los estudiantes.

Crea los métodos get/set que consideres adecuados, así como un método toString() que devuelva la materia, la fecha seguida de la lista de participantes (emplea *StringBuilder*).

Crea una sencilla aplicación que cree un examen de "**Acceso a Datos**" para el **12 de noviembre del 2023 a las 9:45 horas**, con 5 estudiantes con nombres de poetas femeninas del siglo XX.

Guarda el examen en un archivo JSON llamado ***accesoADatos.json*** mediante el api de JSON-B y muestre el contenido del archivo por pantalla, utilizando Files de Java NIO.2 y recupere el archivo para guardarlo en un nuevo objeto Java.

3

Ayuda:

- API Documentation
- Dependencias básicas si no lo consigues con la versión Jakarta: https://javaee.github.io/jsonb-spec/getting-started.html

Ejercicio 4. Gson. Conversión de primitivas JSON

Crea un proyecto Maven en Java y compila el siguiente código. Estudia y explica el resultado obtenido.

```java
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import static java.lang.System.*;

public class GsonDemo {
    public static void main(String[] args) {
        Gson gson = new Gson();

        // Deserialization de una cadena
        String nome = gson.fromJson("\"Sylvia Plath\"", String.class);
        out.println(nome);

        // Serialization
        gson.toJson(256, out); // por pantalla
        out.println(); // salto de línea.
        gson.toJson("<html>", out); // por pantalla.
        out.println(); // salto de línea

        // Gson personalizado deshabilitando el escapado de HTML
        gson = new GsonBuilder().disableHtmlEscaping().create();
        gson.toJson("<html>", out); // Sin escapar HTML
        out.println();
    }
}
```

*El listado anterior declara una clase **GsonDemo** cuyo método **main()** primero instancia Gson, manteniendo su configuración predeterminada. Luego, invoca el método genérico:*

*<T> T **fromJson**(String json, Class<T> classOfT)*

Para deserializar el texto JSON especificado (en json), basado en java.lang.String, en un objeto de la clase especificada (classOfT), que en este caso es String.

4

La cadena JSON "Sylvia Plath" (las comillas dobles son obligatorias), que se expresa como un objeto String de Java, se convierte (sin las comillas dobles) en un objeto String de Java. Una referencia a este objeto se asigna a nome.

Después de imprimir el nombre devuelto, main() llama al método void toJson(Object src, Appendable writer) de Gson para convertir el entero (en clase envolvente) 256 (almacenado por el compilador en un objeto java.lang.Integer) en un entero JSON y mostrar el resultado en la salida estándar.

*main() vuelve a invocar toJson() para mostrar una cadena de Java que contiene <html>. Por defecto, Gson escapa los caracteres HTML < y >, por lo que estos caracteres no se imprimen. Para evitar este escape, es necesario obtener un objeto Gson a través de GsonBuilder, invocando el método **disableHtmlEscaping()** de **GsonBuilder**, que hace main() a continuación. Un segundo intento de imprimir <html> revela que no hay escape.*

Ejercicio 5. Gson. Examen.

Crea un proyecto Maven, igual que el anterior con JSON-B pero **con GSON**, con la sencilla clase Examen que contiene los siguientes atributos:

- **materia**: de tipo *String*.
- **fecha**: de tipo java.util.Date, no *LocalDateTime*. (Veremos por qué)
- **participantes**: de tipo *List* de *String* con los nombres de los estudiantes.

Crea los métodos get/set que consideres adecuados, así como un método toString() que devuelva la materia, la fecha seguida de la lista de participantes (emplea *StringBuilder*).

Crea una sencilla aplicación que cree un examen de "**Acceso a Datos**" para el **12 de noviembre del 2023 a las 9:45 horas**, con 5 estudiantes con nombres de poetas femeninas del siglo XX.

NOTA: para pasar de LocalDate a Date puedes emplear la sentencia:

```
Date.from(LocalDateTime.of(2023, 11, 12, 9, 45)
        .atZone(ZoneId.systemDefault()).toInstant()).
```

También puede hacerse así:

```
Calendar calendar = Calendar.getInstance();
calendar.set(2023, Calendar.NOVEMBER, 12, 9, 45);
Date fechaConcreta = calendar.getTime();
```

5 Guarda el examen en una archivo JSON llamado *accesoADatos.json* (de manera "vistosa" y con formato de fecha *yyyy-MM-dd HH:mm*) mediante el api de Gson y muestre el contenido del archivo por pantalla, utilizando Files de Java NIO.2 y recupere el archivo para guardarlo en un nuevo objeto Java.

Ayuda:

- API Documentation

Ejercicio 6. Gson. Creación de *ClasificacionDAO*.

Crea una clase ***ClasificacionDAO*** para guardar la clasificación de equipos de baloncesto con dos atributos privados y estáticos con los nombres de los archivos para leer y guardar la clasificación:

- OBJECT_FILE: con el nombre fichero ***clasificacion.dat*** para guardar el objeto Java como un flujo a objeto.
- JSON_FILE: con el nombre de fichero ***clasificacion.json*** para guardar el objeto Java en formato JSON.

Además, debe tener un **atributo privado de tipo Gson** para la trabajar con JSON.

El **constructor** por defecto debe crear ese objeto de tipo Gson, pero de modo que tenga una escritura legible.

La clase debe tener 6 métodos:

- ***saveToObject(Clasificacion c):*** que guarda la clasificación en el fichero OBJECT_FILE. Emplea Java NIO.2 para crear el flujo de tipo *Buffered*.
- ***saveToJSON(Clasificacion c, String file):*** que guarda la clasificación en el fichero recogido como argumento. Emplea el objeto de tipo *Gson* y Java NIO.2 para guardar la cadena, a ser posible en una línea. La escritura debe tener un formato legible (no en una línea de texto).
- ***saveToJSON(Clasificacion c):*** que guarda la clasificación en el fichero JSON_FILE. Emplea un objeto de tipo *Gson* y Java NIO.2 para guardar la cadena, a ser posible en una línea. Puedes llamar al método anterior.
- ***getFromObject():*** que obtiene la clasificación a partir del fichero OBJECT_FILE. Emplea Java NIO.2 para crear el flujo de tipo *Buffered*.
- ***getFromJSON(String file):*** que obtiene la clasificación a partir del fichero recogido como argumento. Emplea Java NIO.2

- ***getFromJSON():*** que obtiene la clasificación a partir del fichero JSON_FILE. Invoca al método anterior.

Ejercicio 7. Gson. Creación de *SudokuDAO*.

6

A partir del ejercicio de la tarea del Sudoku, y por medio de las dos estrategias vistas anteriormente, haz que **no serialice en un archivo JSON el alfabero** del Sudoku y sólo lo haga con los datos. Además, debe escribirlo de manera "legible".

Crea una clase ***SudokuDAO*** con las siguientes características:

- **JSON_FILE**: con el nombre de fichero ***sudoku.json*** para guardar el objeto Java en formato JSON.

Además, debe tener un **atributo privado, gson, de tipo Gson** para la trabajar con JSON.

El **constructor** por defecto debe crear ese objeto de tipo Gson, pero de modo que tenga una escritura legible.

La clase debe tener los siguientes métodos:

Para trabajar con objetos Java:

- ***saveToObject(Sudoku c, String ruta)***: que guarda el sudoku en el fichero recogido como argumento. Emplea Java NIO.2 para crear el flujo de tipo Buffered. ¿Cuál es la diferencia entre Files.newOutputStream() y new FileOutputStream()?
- ***getFromObject(String ruta)***: recoge la ruta al fichero y de devuelve el objeto guardado en dicho fichero mediante el método anterior. Emplea Java NIO.2 para crear el flujo de tipo Buffered.

Para trabajar con objetos JSON:

- ***saveToJSON(Sudoku c, String file)***: que guarda el sudoku en el fichero recogido como argumento. Emplea el objeto de tipo Gson y Java NIO.2 para guardar la cadena, a ser posible en una línea de código. La escritura debe tener un formato legible (no en una línea de texto).
- ***saveToJSON(Sudoku c)***: que guarda el sudoku en el fichero JSON_FILE. Emplea un objeto de tipo Gson y Java NIO.2 para guardar la cadena, a ser posible en una línea. Puedes llamar al método anterior.
- ***getFromJSON(String file)***: que obtiene el sudoku a partir del fichero recogido como argumento. Emplea Java NIO.2
- ***getFromJSON()***: que obtiene el sudoku a partir del fichero JSON_FILE. Invoca al método anterior.

Para trabajar con archivos de texto:

- ***Sudoku getFromTXT(String ruta)***: lee el sudoku de un archivo de texto en el que cada línea son los caracteres de cada fila y devuelve el sudoku equivalente.

RETO: CREA UN MÉTODO QUE RESUELVA EL SUDOKU

A modo de ejemplo, puedes ver este método que imprime las soluciones por pantalla:


```
public void resolver() throws Exception {

        List<Sudoku> hijos = getChildren();
        for (Sudoku hijo : hijos) {
            if (hijo.isValid() && hijo.isCompleted()) {
                System.out.println("Solución:");
                System.out.println(hijo);
            } else if (hijo.isValid()) {
                hijo.resolver();
            }
        }

    }
```

- Crea un atributo para **guardar** las soluciones, ***List<Sudoku> solucions***;, y crea el atributo en el constructor (mejor).
- Crea un método ***get*** para las soluciones.
- Haz que el método ***resolver()*** guarde las soluciones hijo en la lista de soluciones.
- Implanta un método en ***SudokuDAO*** que implante un método para guardar las soluciones en un archivo JSON: ***saveSolutionsToJSON(String ruta)***.

Ejercicio 8. Gson. Examen con LocalDate y JsonSerializer/JsonDeserializer

Modifica la clase ***Examen*** que contiene los siguientes atributos:

- **materia**: de tipo *String*.
- **fecha**: *LocalDateTime*.
- **participantes**: de tipo *List* de *String* con los nombres de los estudiantes.

Para que **la fecha la guarde** en formato ***LocalDateTime***, no Date.

Para que la serialización/deserialización funcione correctamente, **debes crear una clase que implante las interfaces siguientes**:

```
public class LocalDateTimeTypeAdapter implements
   JsonSerializer<LocalDateTime>, JsonDeserializer<LocalDateTime>;
```

Emplea el atributo para el **formato** para la fecha en la serialización del objeto de tipo LocalDateTime:

```
private static final DateTimeFormatter formato
                = DateTimeFormatter.ofPattern("d:MM:uuuu HH:mm:ss");
```

8

Crea una sencilla aplicación que cree un examen de "**Acceso a Datos**" para el **12 de noviembre del 2023 a las 9:45 horas**, con 5 estudiantes con nombres de poetas femeninas del siglo XX.

Guarda el examen en una archivo JSON llamado ***accesoADatos.json***, de manera "vistosa" y con formato de fecha anterior mediante el api de Gson y muestre el contenido del archivo por pantalla, utilizando Files de Java NIO.2 y recupere el archivo para guardarlo en un nuevo objeto Java.

Ejercicio 9. Gson. Examen con *TypeAdapter*

Realiza el ejercicio anterior, pero **heredando la interface TypeAdapter**.

Para que la serialización/deserialización funcione correctamente, debes crear una clase que herede la clase ***TypeAdapter***:

```
public class LocalDateTimeAdapter
                extends TypeAdapter<LocalDateTime>;
```

Emplea el formato siguiente para la fecha en la serialización del objeto de tipo LocalDateTime:

```
private static final DateTimeFormatter formato
          = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
```