Tema 3

Almacenamiento en ficheros JSON.

3.1. Introducción.

JSON (JavaScript Object Notation).

Características:

- Independiente del lenguaje aunque originado para Javascript.
- Posibilidad de validar con esquemas, basándose en cómo funciona *XML Schema*.
- Gran popularidad otros sistemas -> p.ej: *MongoDB*, utilizan su formato de datos.
- Existe un derivado binario: BSON (Binary JSON).

Comparativa con XML:

- Más ligero.
- Menos versátil, por ejemplo: utiliza UTF-8 sin muchas posibilidades de cambio.
- No admite comentarios.

Herramientas:

- JSON Beautify para dejarlo más legible o más compacto.
- JSONLint para validar.
- Json Diff para encontrar diferencias entre dos archivos.
- JSON Parser para transformar entre JSON y objetos.
- XML to JSON Converter para transformar entre XML y JSON.

Elementos:

- **:** separa nombre de valor.
- **,** separa elementos.
- **{}** delimita objetos.
- **[]** delimita arrays.
- **""** delimita *strings.*

3.1.1. Ejemplos.

Obtenidos aquí y aquí:.

```
{
    "firstName": "John",
    "lastName": "Smith",
    "isAlive": true,
    "age": 27,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "office",
            "number": "646 555-4567"
        }
    ],
    "children": [
        "Catherine",
        "Thomas",
        "Trevor"
    ],
    "spouse": null
}
```

```
{
   "squadName": "Super hero squad",
   "homeTown": "Metro City",
   "formed": 2016,
   "secretBase": "Super tower",
   "active": true,
   "members": [
      {
         "name": "Molecule Man",
         "age": 29,
         "secretIdentity": "Dan Jukes",
         "powers": [
            "Radiation resistance",
            "Turning tiny",
            "Radiation blast"
         ]
      },
      {
         "name": "Madame Uppercut",
         "age": 39,
         "secretIdentity": "Jane Wilson",
         "powers": [
            "Million tonne punch",
            "Damage resistance",
            "Superhuman reflexes"
         ]
      },
      {
         "name": "Eternal Flame",
         "age": 1000000,
         "secretIdentity": "Unknown",
         "powers": [
            "Immortality",
            "Heat Immunity",
            "Inferno",
            "Teleportation",
            "Interdimensional travel"
         ]
      }
   ]
}
```


3.1.2. Tipos de datos.

Tipos de datos básicos:

- `Number`: no hay distinción entre entero y coma flotante. No admite valores como NaN. Cada lenguaje que interprete JSON puede tratar los números de forma diferente lo que puede causar problemas de portabilidad.
- `String`: entre comillas dobles (no comillas simples). Permite escape de caracteres con \.
- `Boolean`:true o false.
- `Array`: lista ordenada de cero o más elementos, sin necesidad de que sean del mismo tipo. Entre corchetes y con elementos separados por coma.
- `Object`: conjunto de pares clave-valor; entre llaves y separados por comas. En el ejemplo anterior, *address*.
- `null`.

Ni objetos ni arrays admiten coma detrás del último elemento aunque hay muchos procesadores que ignoran este error.

3.2. JSON con Java.

Muchas bibliotecas[1]. Ejemplos:

- Jackson.
- Gson.
- json-io.
- Genson.
- JSON in Java.

Se está desarrollando una especificación de API en Jakarta pero aun no hay muchas implementaciones y no están casi documentadas.

De cualquier forma, si quieres empezar a usarla, puedes usar estos enlaces sobre JSON-P (gestión a más bajo nivel) o JSON-B (a nivel más alto).

3.3. GSON.

Desarrollada por Google.

[1] Libraries.

Puedes consultar la documentación aquí.

> **NOTA:**
> Lo que hacemos a continuación es serializar y deserializar por lo que tendríamos los mismos problemas vistos para ficheros planos.

Dependencia de Maven:

```
<dependency>
   <groupId>com.google.code.gson</groupId>
   <artifactId>gson</artifactId>
   <version>2.10</version>
</dependency>
```

3.3.1. Leer con GSON.

Lo primero que vamos a probar es a leer el contenido.

Hemos creado una clase con más datos que contiene la lista de jugadores. Así el ejemplo será más completo:

```
public class LigaNFL {
   private String temporada;
   private String campeon;
   private List <JugadorNFL> jugadores;

   // Constructores, getters, setters y toString.

}
```

Creamos un elemento de tipo `GSON` que nos permite obtener información de fuentes JSON.

Con el método `fromJson` lo cargamos:

```
Gson gson = new Gson();

try(Reader reader =
      Files.newBufferedReader(Paths.get("jugadoresNFL.json"))) {
   LigaNFL liga =  gson.fromJson(reader, LigaNFL.class);

   System.out.println(liga);
} catch (IOException ex) {
   System.out.println("ERROR: No se encuentra el archivo.");
}
```

3.3.1.1. Leer un array de JSON.

JSON permite archivos con un array (concepto que no existe en XML):

```
[
   {
      "nombre": "Herbert",
      "numero": 10,
      "lesionado": false,
      "fechaContrato": {
         "year": 2022,
         "month": 11,
         "dayOfMonth": 8,
         "hourOfDay": 12,
         "minute": 23,
         "second": 4
      }
   },
   {
      "nombre": "Allen",
      "numero": 13,
      "lesionado": true,
      "fechaContrato": {
         "year": 2022,
         "month": 11,
         "dayOfMonth": 8,
         "hourOfDay": 12,
         "minute": 23,
         "second": 4
      }
   }
]
```

Para leerlo usaríamos:


```
Gson gson = new Gson();

try(Reader reader =
      Files.newBufferedReader(Paths.get("listaJugadoresNFL.json"))) {
   // Observa que la clase es un array
   List<JugadorNFL> jugadores =
      Arrays.asList(gson.fromJson(reader, JugadorNFL[].class));

   for(JugadorNFL jugador : jugadores) {
      System.out.println(jugador);
   }
} catch (IOException ex) {
   System.out.println("ERROR: No se encuentra el archivo.");
}
```

3.3.2. Escribir con GSON.

Teniendo un objeto `LigaNFL` ya creado, podríamos escribir con el siguiente código:

```
Gson gson = new Gson();

try(Writer writer =
      Files.newBufferedWriter(Paths.get("jugadoresNFL.json"))) {
   gson.toJson(liga, writer);
} catch (IOException ex) {
   System.out.println("Error al escribir en el fichero.");
}
```

Pero nos lo escribe todo en una línea. Si queremos un archivo legible por los humanos:

```
// para crear el GSON con prettyPrint
Gson gson = new GsonBuilder().setPrettyPrinting().create();

try(Writer writer =
      Files.newBufferedWriter(Paths.get("jugadoresNFL.json"))) {
   gson.toJson(liga, writer);
} catch (IOException ex) {
   System.out.println("Error al escribir en el fichero.");
}
```

Lo que nos dejaría el siguiente archivo:


```
{
   "temporada": "2050",
   "campeon": "LA Chargers",
   "jugadores": [
      {
         "nombre": "Herbert",
         "numero": 10,
         "lesionado": false,
         "fechaContrato": {
            "year": 2022,
            "month": 11,
            "dayOfMonth": 8,
            "hourOfDay": 12,
            "minute": 16,
            "second": 42
         }
      },
      {
         "nombre": "Allen",
         "numero": 13,
         "lesionado": true,
         "fechaContrato": {
            "year": 2022,
            "month": 11,
            "dayOfMonth": 8,
            "hourOfDay": 12,
            "minute": 16,
            "second": 42
         }
      }
   ]
}
```

3.3.2.1. Escribir listas de objetos.

En JSON no es necesario que haya un elemento raíz por lo que podemos escribir una lista directamente:

```
// para crear el GSON con prettyPrint
Gson gson = new GsonBuilder().setPrettyPrinting().create();

try(Writer writer =
      Files.newBufferedWriter(Paths.get("listaJugadoresNFL.json"))) {
   gson.toJson(jugadores, writer);
} catch (IOException ex) {
   System.out.println("Error al escribir en el fichero.");
}
```

Ejercicios 3.1

Los siguientes ejercicios debes hacerlos para que escriban y lean un archivo que empiece con un objeto y otro que empiece con un array.

1. Crea un proyecto para escribir elementos de tipo participante de la carrera que hemos usado todo el curso.
2. Añade la funcionalidad necesaria para leer los participantes de un archivo y mostrarlos por pantalla.

3.4. JSON in Java.

JSON in Java es un proyecto para usar JSON con Java de forma ligera y sin dependencias externas.

Dependencia Maven:

```
<dependency>
   <groupId>org.json</groupId>
   <artifactId>json</artifactId>
   <version>20220924</version>
</dependency>
```

Se usan principalmente las clases:

- `JSONObject`.
- `JSONArray`.

3.4.1. Leer con JSON in Java.

Vamos a ver cómo podemos leer lo que queramos del archivo. Con esta librería es más artesanal pero nos permite usar nuestros *setters* o constructores para comprobar los argumentos y asegurarnos de que el resultado es correcto.

Nos vamos a encontrar dos casos principales: (ambos pueden ser complejos y contener otros objetos y/o arrays.)

- Leer un objeto.
- Leer un array.

3.4.1.1. Leer un objeto.

Suponemos un archivo como este (Observa que no sigue el orden en el que más adelante añadiremos los elementos:):

```
{
   "lesionado": "no",
   "numero": "10",
   "fechaContrato": {
      "mes": 11,
      "dia": 8,
      "anio": 2022
   },
   "nombre": "Herbert"
}
```

Para leer un objeto nos ayudamos de un método auxiliar que sacará su contenido y lo cargará en un objeto java:

```java
public static JugadorNFL json2JugadorNFL(JSONObject jugadorJSON) {
   JugadorNFL jugador = null;

   String nombre;
   short numero;
   boolean lesionado = false;
   GregorianCalendar fechaContrato;

   JSONObject fechaJSON;

   nombre = jugadorJSON.getString("nombre");
   numero = (short)jugadorJSON.getInt("numero");
   if(jugadorJSON.get("lesionado").equals("si")) {
      lesionado = true;
   }

   // para obtener la fecha primero hay que sacar todo el elemento JSON.
   fechaJSON = (JSONObject)jugadorJSON.get("fechaContrato");
   fechaContrato = new GregorianCalendar(fechaJSON.getInt("anio"),
      fechaJSON.getInt("mes"), fechaJSON.getInt("dia"));

   jugador = new JugadorNFL(nombre, numero, lesionado, fechaContrato);

   return jugador;
}
```

Con él, podemos leer el fichero:

```
public static void leerJugador() {
   try(Reader reader =
         Files.newBufferedReader(Paths.get("jugadorNFL.json"))) {
      JugadorNFL jugador;

      JSONTokener parser = new JSONTokener(reader);
      JSONObject jugadorJSON = new JSONObject(parser);

      jugador = ConversorJugador.json2JugadorNFL(jugadorJSON);

      System.out.println(jugador);
   } catch(JSONException ex) {
      System.out.println(
         "ERROR: no se puede leer el contenido del archivo");
   }
   catch (IOException ex) {
      System.out.println("ERROR: No se encuentra el archivo.");
   }
}
```

3.4.1.2. Leer un array.

Si tenemos un array de objetos:

```
[
   {
      "lesionado": "no",
      "numero": "10",
      "fechaContrato": {
         "mes": 11,
         "dia": 8,
         "anio": 2022
      },
      "nombre": "Herbert"
   },
   {
      "lesionado": "si",
      "numero": "13",
      "fechaContrato": {
         "mes": 11,
         "dia": 8,
         "anio": 2022
      },
      "nombre": "Allen"
   }
]
```

Lo leeríamos de la siguiente manera:

```
public static void leerJugadores() {
   try(Reader reader =
   Files.newBufferedReader(Paths.get("listaJugadoresNFL.json"))) {
      JugadorNFL jugador;
      JSONObject jugadorJSON;
      List<JugadorNFL> jugadores = new ArrayList<>();

      JSONTokener parser = new JSONTokener(reader);
      JSONArray jugadoresJSON = new JSONArray(parser);

      Iterator <Object> iterador = jugadoresJSON.iterator();

      while(iterador.hasNext()) {
         jugadorJSON = (JSONObject)iterador.next();
         jugador = ConversorJugador.json2JugadorNFL(jugadorJSON);
         jugadores.add(jugador);
      }

      for(JugadorNFL juga : jugadores) {
         System.out.println(juga);
      }
   } catch(JSONException ex) {
      System.out.println(
         "ERROR: no se puede leer el contenido del archivo");
   }
   catch (IOException ex) {
      System.out.println("ERROR: No se encuentra el archivo.");
   }
}
```

3.4.2. Escribir con JSON in Java.

Hay que ir creando el objeto JSON a partir del objeto Java que tengamos:

```
public static JSONObject jugadorNFL2Json(JugadorNFL jugador) {
    JSONObject jugadorJSON = new JSONObject();
    JSONObject fechaJSON = new JSONObject();
    String lesionado;
    GregorianCalendar fechaContrato;

    jugadorJSON.put("nombre", jugador.getNombre());
    // convertimos el shor a un string.
    jugadorJSON.put("numero", ""+jugador.getNumero());

    // convertimos un booleano a texto legible.
    lesionado = jugador.isLesionado()?"si":"no";
    jugadorJSON.put("lesionado", lesionado);

    //gestión de la fecha.
    fechaContrato = jugador.getFechaContrato();
    fechaJSON.put("anio", fechaContrato.get(Calendar.YEAR));
    fechaJSON.put("mes", fechaContrato.get(Calendar.MONTH));
    fechaJSON.put("dia", fechaContrato.get(Calendar.DAY_OF_MONTH));

    // añadirla al jugador
    jugadorJSON.put("fechaContrato", fechaJSON);

    return jugadorJSON;
}
```

Ese tipo de dato lo podemos escribir en un archivo:

```
public static void escribirJugador() {
   JugadorNFL jugador = new JugadorNFL("Herbert", (short)10, false,
   new GregorianCalendar());

   JSONObject jugadorJSON = ConversorJugador.jugadorNFL2Json(jugador);

   try(Writer writer =
         Files.newBufferedWriter(Paths.get("jugadorNFL.json"))) {
      writer.write(jugadorJSON.toString(2));
   }
   catch (IOException ex) {
      System.out.println("ERROR: no se pudo crear el fichero.");
   }
}
```

También podemos crear un fichero a partir de un array JSON:

```
public static void escribirJugadores() {
   JSONArray lista;
   List<JugadorNFL> jugadores = new ArrayList<>();
   List<JSONObject> jugadoresJSON = new ArrayList<>();

   jugadores.add(new JugadorNFL("Herbert", (short)10, false,
      new GregorianCalendar()));
   jugadores.add(new JugadorNFL("Allen", (short)13, true,
      new GregorianCalendar()));
   jugadores.add(new JugadorNFL("James Jr.", (short)3, false,
      new GregorianCalendar()));
   jugadores.add(new JugadorNFL("Mahomes", (short)15, false,
      new GregorianCalendar()));

   for(JugadorNFL jugador : jugadores) {
      JSONObject jugadorJSON =
         ConversorJugador.jugadorNFL2Json(jugador);
      jugadoresJSON.add(jugadorJSON);
   }

   lista = new JSONArray(jugadoresJSON);

   try(Writer writer =
   Files.newBufferedWriter(Paths.get("listaJugadoresNFL.json"))) {
      writer.write(lista.toString(2));
   }
   catch (IOException ex) {
      System.out.println("ERROR: no se pudo crear el fichero.");
   }
}
```

Ejercicios 3.2

1. Al proyecto de jugadores NFL con JSON in Java que hay como ejemplo en el aula virtual, añádele funcionalidad para que se escriban objetos de tipo `LigaNFL` en lugar de solo listas de jugadores.
2. Añade más funcionalidad para que ahora también se lean esos archivos y se carguen en un objeto de clase `LigaNFL`. Asegúrate de comprobar que los datos leídos son correctos y prueba a cambiar algún tipo de dato por otro (por ejemplo un número por una palabra con letras).

3. Implementa PokemonDAO con JSON in Java.

3.5. Bibliografía.

- Carlos Tessier Fernández (2021). Curso Acceso a Datos.
- Curso MEFP-IFCS02-AD. Ministerio de Educación y Formación Profesional.
- Documentación de la API de Java.
- JSON. Wikipedia.
- JSON. MDN.
- How to read and write JSON using Gson in Java. Atta.
- Introduction to JSON in Java. Educba.

