# JSON

… y ya que hablamos de mejoras. ¿No te parece tedioso tener que ir poniendo los < > todo el rato? Quiero decir, a veces parece que escribimos más etiquetas que información. Pues bueno, pues sí. Hay alternativas.

**JSON** (*JavaScript Object Notation*) es un formato de texto que forma parte del sistema de JavaScript y que se deriva de su sintaxis. Hoy en día se le considera un formato independiente del lenguaje. Su objetivo es el acceso, almacenamiento e intercambio de información, y se le considera una alternativa al lenguaje XML.

Se le supone como ventaja que es más sencillo escribir parsers para él, pero estas ventajas no son relevantes salvo que tengas grandes volúmenes de información que transmitir. De todas formas, es habitual usar XML y JSON en la isma aplicación, así que el uno no sustituye al otro.

A lo largo de esta asignatura y otras del módulo usaremos uno u otro formato, por lo que es necesario conocer ambos. Por ejemplo, para transmitir información desde un móvil (PMDM) usaremos JSON, pero como ya se ha mencionado, en la misma asignatura usaremos XML para definir los interfaces de las actividades.

### JSON - Sintaxis

La sintaxis de JSON funciona de modo similar a JavaScript. Por ejemplo:

- El acceso a la información se hace mediante claves.
- La información se separa por comas.
- Las llaves agrupan objetos.
- Los corchetes agrupan conjuntos de datos.

Los tipos de datos disponibles con JSON son:

- Números: Se permiten números negativos y opcionalmente pueden contener parte fraccional separada por puntos. Ejemplo: 123.456
- Cadenas: Representan secuencias de cero o más caracteres. Se ponen entre doble comilla y se permiten cadenas de escape. Ejemplo: "Hola"
- Booleanos: Representan valores booleanos y pueden tener dos valores: true y false
- null: Representan el valor nulo.
- Array: Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes. Ejemplo ["juan","pedro","jacinto"]

- Objetos: Son colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y puestas entre llaves. El nombre tiene que ser una cadena entre comillas dobles. El valor puede ser de cualquier tipo (ver siguiente ejemplo).

Por tanto, podemos transformar un XML a un JSON (y viceversa) a mano. Por ejemplo, si tenemos:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<students>
    <student id="1">
        <name>Juan</name>
        <surname>Torres</surname>
        <asignature>ADT</asignature>
    </student>
    <student id="2">
        <name>Ana</name>
        <surname>Sanz</surname>
        <asignature>PMDM</asignature>
    </student>
</students>
```

Su JSON sería:

```json
{
  "students": {
    "student": [
      {
        "id": "1",
        "name": "Juan",
        "surname": "Torres",
        "asignature": "ADT"
      },
      {
        "id": "2",
        "name": "Ana",
        "surname": "Sanz",
        "asignature": "PMDM"
      }
    ]
  }
}
```

**JSON - Escritura**

A diferencia de lo que hemos visto hasta ahora, Java no dispone de librerías nativas que le permitan procesar JSON. Para eso, es necesario descargarse las librerías de terceros, como por ejemplo la librería **GSON** perteneciente a Google. En el momento de escribir estas líneas, la última versión de las mismas es la **gson-2.13.1**

Es razonablemente sencillo crear un programa que genera ficheros JSON mediante Gson. La librería es capaz de convertir objetos directamente a JSON, como puedes ver en el ejemplo.

```
// The List of Students
List<Student> students = new ArrayList<>();
students.add(new Student(1, "Juan", "Torres", "ADT"));
students.add(new Student(2, "Ana", "Perez", "PMDM"));
students.add(new Student(3, "Luis", "Martín", "PSP"));

// The gson
Gson gson = null;
try {
    // The builder
    gson = new GsonBuilder().setPrettyPrinting().create();

    FileWriter writer = new FileWriter("JSONExample.json");

    // Just dump the list
    gson.toJson(students, writer);
    writer.flush();
} catch (Exception e) {
    System.out.println("An error occurred.");
}
```

El objeto Student no deja de ser un POJO normal y corriente. En este caso, lo que convertimos es el Listado de Students.

```
public class Student {

    private int id;
    private String name;
    private String surname;
    private String asignature;
```

**JSON – Lectura**

De igual forma, el proceso de lectura sería algo simple. Nótese que en este caso usamos un objeto Type para que el Gson sea capaz de convertir directamente de texto a objeto.

```
// The gson
Gson gson = null;
try {
    // The builder
    gson = new GsonBuilder().setPrettyPrinting().create();

    FileReader reader = new FileReader("JSONExample.json");

    // what type of token are we reading? An Student
    Type listType = new TypeToken<List<Student>>(){}.getType();
    List<Student> loadedStudents = gson.fromJson(reader, listType);

    System.out.println("Students: ");
    for (Student student : loadedStudents) {
        System.out.println(student);
    }
} catch (Exception e) {
    System.out.println("An error occurred.");
}
```

El fichero generado JSONExample.json obtenido es:

```
[
  {
    "id": 1,
    "name": "Juan",
    "surname": "Torres",
    "asignature": "ADT"
  },
  {
    "id": 2,
    "name": "Ana",
    "surname": "Perez",
    "asignature": "PMDM"
  },
  {
    "id": 3,
    "name": "Luis",
    "surname": "Martín",
    "asignature": "PSP"
  }
]
```

Obviamente, existen muchos más métodos a disposición del programador, con los que poder acceder a la información con mucho más detalle.

Éste ejemplo recupera los objetos del JSON uno a uno de forma independiente: el JsonArray, los JsonObject,…

```
try {
    FileReader reader = new FileReader("JSONExample.json");

    // Parse the JSON into a JsonArray
    JsonArray jsonArray = JsonParser.parseReader(reader).getAsJsonArray();

    // Iterate the list to get each "student"
    System.out.println("Students: ");
    for (JsonElement jsonElement : jsonArray) {

        // Each element is a JsonObject
        JsonObject obj = jsonElement.getAsJsonObject();

        // We get the attributes
        int id = obj.get("id").getAsInt();
        String name = obj.get("name").getAsString();
        String surname = obj.get("surname").getAsString();
        String asignature = obj.get("asignature").getAsString();

        System.out.printf("ID: %d, Name: %s %s, Asignature: %s%n",
                id, name, surname, asignature);
    }
} catch (Exception e) {
    System.out.println("An error occurred.");
}
```

## Práctica - 6

Diseña un XML que permita gestionar todos los alumnos del centro. En el centro hay dos ciclos, DAM y DAW, pero para cada uno hay una clase de primero y una de segundo (cuatro clases en total). Cada alumno sólo puede pertenecer a una clase, pero es posible que haya alumnos con el mismo nombre.

Realiza las siguientes consultas:

- Muestra todos los alumnos del centro
- Muestra todos los alumnos de DAM
- Muestra todos los alumnos de DAW
- Muestra todos los alumnos de 1º y de 2º de cada ciclo (4 consultas).
- Muestra a un alumno por nombre
- Muestra un alumno por nombre de un ciclo y curso concreto
- Número total de alumnos de cada ciclo