Emmagatzemament amb fitxers JSON

JSON és un altre format de text lleuger per a l'intercanvi de dades. Les sigles JSON provenen de *JavaScript Object Notation* (Notació d'objectes de JS), i es tracta d'un subconjunt de la notació literal d'objectes d'aquest llenguatge, que s'ha adoptat junt amb XML com un dels grans estàndards d'intercanvi i emmagatzemament de dades.

Un dels avantatges de JSON respecte a XML és la facilitat d'escriure *parsers*, però més important que això, és que expressa el mateix que XML però amb de forma molt més concreta i concisa, pel que s'usa habitualment en entorns on el flux de dades és important, com és el cas dels servidors de Google, Yahoo, etc. que atenen a milions d'usuaris.

1. El format JSON

http://www.JSON.org

Els tipus de dades que podem representar en JSON són:

- **Números**, tant enters com decimals,
- **Cadenes**, expressades entre cometes i amb la possibilitat d'incloure seqüències d'escapament,
- **Booleans**, per representar els valors *true* i *false*,
- **Null**, per representar el valor nul,
- **Array**, per representar llistes de zero o més valors, de qualsevol tipus, entre corxets i separats per comes,
- **Objectes**, com a col·leccions de parells `<clau>:<valor>` , separats per comes i entre claus, i de qualsevol tipus de valor.

Ho veurem millor amb un exemple conegut: El dels mòduls amb el què estem treballant.

Recordem la taula:

Mòdul	Hores	Qualificació
Accés a Dades	6	8.45
Programació de serveis i processos	3	9.0
Desenvolupament d'interfícies	6	8.0
Programació Multimèdia i dispositius mòbils	5	7.34
Sistemes de Gestió Empresarial	5	8.2

Mòdul	Hores	Qualificació
Empresa i iniciativa emprenedora	3	7.4

Aquesta, es podria representar en JSON de la següent forma:

```
{
    "curs": [
            {
                "nom": "Accés a Dades",
                "hores": 6,
                "qualificacio":  8.45
            },
            {
                "nom": "Programació de serveis i processos",
                "hores": 3,
                "qualificacio": 9.0
            },
            {
                "nom": "Desenvolupament d'interfícies",
                "hores": 6,
                "qualificacio": 8.0
            },
            {
                "nom": "Programació Multimèdia i dispositius mòbils",
                "hores": 5,
                "qualificacio": 7.34
            },
            {
                "nom": "Sistemes de Gestió Empresarial",
                "hores": 5,
                "qualificacio": 8.2
            },
            {
                "nom": "Empresa i iniciativa emprenedora",
                "hores": 3,
                "qualificacio": 7.4
            }
        ]
}
```

Veiem com **curs** és un vector o una llista de mòduls (encara que no utilitzem ara l'etiqueta modul), que en aquest cas són objectes amb tres elements: el nom que és una cadena de caràcters, les hores que és un enter, i la qualificació, que es representa com un número decimal.

Fixem-nos que, a l'igual que XML, també necessitem d'un objecte arrel, en aquest cas l'element *curs*.

> Mode friki on

> A Internet hi ha molts serveis web que ofereixen respostes en format JSON. Veiem el següent exemple extret de https://swapi.co/, l'API de personatges de Star Wars:


```
{
        "name": "Luke Skywalker",
        "height": "172",
        "mass": "77",
        "hair_color": "blond",
        "skin_color": "fair",
        "eye_color": "blue",
        "birth_year": "19BBY",
        "gender": "male",
        "homeworld": "https://swapi.co/api/planets/1/",
        "films": [
                "https://swapi.co/api/films/2/",
                "https://swapi.co/api/films/6/",
                "https://swapi.co/api/films/3/",
                "https://swapi.co/api/films/1/",
                "https://swapi.co/api/films/7/"
        ],
        "species": [
                "https://swapi.co/api/species/1/"
        ],
        "vehicles": [
                "https://swapi.co/api/vehicles/14/",
                "https://swapi.co/api/vehicles/30/"
        ],
        "starships": [
                "https://swapi.co/api/starships/12/",
                "https://swapi.co/api/starships/22/"
        ],
        "created": "2014-12-09T13:50:51.644000Z",
        "edited": "2014-12-20T21:17:56.891000Z",
        "url": "https://swapi.co/api/people/1/"
}
```


2. JSON i Java

Existeix un gran ventall de llibreríes Java per a la manipulació de documents JSON (GSON, Jackson, JSON.simple...). Al nostre cas, anem a utilitzar la llibreria `org.json` , que podem consultar al repositori de Maven: https://mvnrepository.com/artifact/org.json/json

Al següent apartat comentarem les funcionalitats de la llibrería, i en un document annexe, veurem com incorporar-la als nostres projectes a través del gestor de dependències de Gradle.

2.1. `org.JSON`

http://www.studytrails.com/java/json/java-org-json/

La llibrería ofereix un conjunt de classes per parsejar documents JSON per a Java, a més de conversors entre JSON i XML.

D'entre les classes que ofereix, podríem destacar:

- **org.json.JSONObject**: Emmagatzema parells clau-valor de forma desordenada. Els valors poden ser Boolean, JSONArray, Number, String i JSONObject.NULL. Els seus constructors prenen com a entrada diferents representacions (String, mapes, beans) i les emmagatzemen com a un conjunt d'elements clau-valor.
- **org.json.JSONTokener**: Parseja una cadena JSON, i és utilitzada internament per JSONObject i alters classes per parsejar strings JSON.
- **org.json.JSONArray**: Emmagatzema una seqüència de valors, i representa un array JSON.
- **org.json.JSONWriter**: Ofereix una forma de produir text JSON. Disposa d'un mètode `append(String)`, que afig més text a un objecte JSON de tipus text, a més dels mètodes `key(String)` i `value(String)` pe afegir claus i valors a una cadena JSON. La classe, a més, també permet escriure un array.

3. Exemple: `JSONLib`

Continuant amb el nostre exemple, anem a crear un nou projecte anomenat `JSONLib` per escriure i llegir la informació de la taula de mòduls. El funcionament serà semblant al dels exemples anteriors, amb la diferència que ho gestionarem des de Gradle:

```
$ gradle run --args="write moduls.json"
```

Generarà un fitxer json amb els mòduls definits al codi, i

```
$ gradle run --args="read moduls.json"
```

Llegirà un fitxer JSON indicat, i el mostrarà per pantalla

3.1. Escriptura del fitxer JSON

Creació de la llista de mòduls

La nostra classe principal `JSONLib` tindrà una llista amb els mòduls del curs, a partir de la qual generar el JSON. Per tal de crear-la, es disposa del mètode `creaCurs`:

```
private void creaCurs() {
    // Aquest mètode inicializa l'objecte "Curs" de la classe JSONLib
    // que no és més que un vector de mòduls

    // Definim els vectors per inicialitzar dades
    String[] moduls={"Accés a Dades", "Programació de serveis i processos", "Desenvolup
    int[]  hores={6, 3, 6, 5, 5, 3};
    double[] notes={8.45, 9.0, 8.0, 7.34, 8.2, 7.4};

    // Recorrem els vectors, creant els objectes
    // de tipus Modul i guardant-los en Curs
    for (int i=0;i<moduls.length;i++){
        Modul m = new Modul(moduls[i], hores[i], notes[i]);
        this.Curs.add(m);
    }

}
```

Creació de l'objecte JSON

Una vegada tenim en l'atribut *Curs* la informació, anem a crear l'objecte JSON amb la funció `creaJSON` . Aquesta funció tindrà les següents característiques:

- Retornarà un objecte de tipus `JSONObject` amb el JSON equivalent a l'estructura de mòduls.
- Aquest mètode crarà un objecte arrel (tipus `JSONObject`) amb la clau "curs" i el valor del qual serà la llista de mòduls, que serà de tipus `JSONArray` .
- Per tal d'afegir parells clau-valor a un objecte JSON, farem ús del mètode `objecteJSON.put(clau, valor)` .

Primerament, anem a donar una ullada a la classe `Modul` , a la qual hem incorporat un nou mètode anomenat `getModulJSON` : Com veiem, en aquest mòdul creem un objecte JSON que anomenem modul i al que afegim (amb put) els atributs de la classe. El resultat és un objecte JSON que conté la informació del mòdul:

```
public JSONObject getModulJSON(){
    JSONObject modul = new JSONObject();

    modul.put("nom", this.nom);
    modul.put("hores", this.hores);
    modul.put("nota",  this.nota);

    // Si volguerem afegir un element nul,
    // hauriem de fer:
    // modul.put("atribut", JSONObject.NULL);

    return modul;

};
```

Amb aquest mètode de la classe `Modul`, que hem incorporat al mateix paquet que `JSONLib`, ja podem passar a veure el mètode `creaJSON`, d'aquesta última classe:

```
private JSONObject creaJSON(){

    // L'arrel del document serà un objecte "Curs"
    JSONObject curs=new JSONObject();
    // Que contindrà una llista de mòduls:
    JSONArray jsarray = new JSONArray();

    // Anem a omplir la llista obtenint els
    // diferents mòduls com a JSONObject
    // (veure el mètode getModulJSON de la classe Modul)
    for(Modul m:this.Curs)  {
        JSONObject modul_json=m.getModulJSON();
        // Afegim l'ojbecte JSON al vector
        jsarray.put(modul_json);
    }

    // Creem l'element arrel amb la clau "curs"
    // i posem com a valor el vector.
    curs.put("curs", jsarray);

    // Retornem el curs
    return (curs);

}
```

Emmagatzemament en disc de l'objecte JSON

El mètode anterior ens retorna un objecte de tipus `JSONObject`, que caldrà emmagatzemar al disc. Per a això, s'ha creat el mètode `EscriuJSON`, que pren el nom del fitxer a escriure i l'objecte JSON i crea aquest fitxer en disc.

Per fer això, farem ús d'un `FileWriter`, i del mètode `toString` per a la classe `JSONObject`, que el que fa és tornar-nos una cadena de text amb el contingut del JSON. Aquest mètode ens permet indicar-li el número d'espais que utilitzarem en la indentació del fitxer, que en este cas hem posat 4.

```
private void EscriuJSON(String filename, JSONObject jso){

    try {
        FileWriter file = new FileWriter(filename);
        file.write(jso.toString(4)); // 4 són els espais d'indentació
        file.close();

    } catch (IOException e) {
        System.out.println("Error, no es pot crear el fitxer "+filename);
    }
}
```

Amb aquests mètodes ja tenim la capacitat de generar fitxers JSON amb la informació indicada al vector, al següent apartat anem a veure com llegir aquest fitxer.

3.2. Lectura del fitxer JSON

Per tal de llegir el fitxer JSON, hem incorporat dos mètodes més a la classe JSONLib: `LligJSON` i `MostraJSON` .

El mètode `LligJSON` rep com a argument el fitxer a llegir, i fa ús de la classe `FileReader` per llegir el fitxer caràcter a caràcter i crear un string amb tot el contingut del JSON. Una vegada tenim el contingut del fitxer en una cadena de text, haurem de convertir-lo a objecte JSON. Per fer això, la classe `JSONObject` ofereix un constructor que permet crear l'objecte a partir d'un fitxer JSON ben format. Així doncs, només haurem de crear l'objecte passant-li aquesta cadena al constructor:

```
private JSONObject LligJSON(String filename){
    try {
        // Amb FileReader llegirem caràcter a
        // caràcter el fitxer i l'afegim al string myJson
        FileReader file = new FileReader(filename);
        String myJson="";

        int i;
        while ((i=file.read()) != -1)
            myJson=myJson+((char) i);

        //System.out.println(myJson);
        file.close();

        // I fem ús del constructor de JSONObject
        // al que li passem un string amb el JSON:
        return (new JSONObject(myJson));


    } catch (Exception e)
    {
        System.out.println("Error llegint el fitxer");
        return null;
    }

}
```

Com veiem, el mètode ens retorna el JSONObject corresponent al fitxer que tenim al disc.

Ara ens quedarà mostrar-lo per pantalla, per a això, s'ha creat el mètode `MostraJSON` que, donat un objecte de tipus JSON, accedirà element a element i anirà mostrant-nos-els per pantalla.

Per a això, fem ús de diferents mètodes `get` de les classes de JSON. concretament `getJSONArray` per obtenir el valor d'un element que és un array (com és el cas del curs), i `get` , per obtindre els valors dels diferents camps:

```
private void MostraJson(JSONObject json){

    // amb el mètode getJSONArray obtenim el primer
    // element "curs", que era una llista
    JSONArray jsa=json.getJSONArray("curs");

    // I ara recorrem aquesta llista:
    for (int i = 0; i < jsa.length(); i++) {
        // Agafem cada element de l'array amb "get"
        JSONObject modul=(JSONObject)jsa.get(i);
        // Amb el get anterior tindrem objectes JSON
        // de mòduls, tipus:
        // {"nom": "Modul", "hores": hores, "nota": nota }
        // Als valors d'aquests parells també accedirem amb get:
        System.out.println("Modul: "+ modul.get("nom"));
        System.out.println("Hores: "+ modul.get("hores"));

        System.out.println("Nota: "+modul.get("nota"));

    }

}
```

4 Exercicis d'ampliació:

4.1 `loadJson`

Crear una funció que, en compte de mostrar per pantalla, cree un `ArrayList<Modul>` . El prototip serà `private ArrayList<Modul> MostraJson(JSONObject json)` . Serà semblant a la que ferem en l'exemple de XML.

4.2 Afegir a la classe `Modul` , mètodes (amb els arguments necesaris) per a:

- `getModulJSON` implementat als exemples.
- `getModulXML` que obté un element `XML` amb la representació d'un mòdul.
- sobrecarrega el constructor per a que rebent un `JSONObject` cree el Modul
- sobrecarrega el constructor per a que rebent un `Element` cree el Modul (Node XML)