Traitement du JSON en Java ?

- Nous avons vu comment JSON est si facile à manipuler en JavaScript ☺
- Et en Java ?
  - Pas de support natif
  - Plusieurs librairies permettent de traiter du JSON en Java
  - Nous utiliserons Genson : http://owlike.github.io/genson/

Genson

A fast & modular Java <-> JSON library

Genson

- Désérialisation

  ≡ Transformation de JSON vers du Java
- Sérialisation

  ≡ Transformation de Java vers du JSON
- Genson peut réaliser les deux sens ☺ ☺

Instanciation de Genson

- Directement :

```
Genson genson=new Genson();
```

- Via un Builder pour une configuration précise :

```
Genson genson=new GensonBuilder()
   .useDateFormat(new SimpleDateFormat("yyyy-MM-dd"))
   .useIndentation(true)
   .useConstructorWithArguments(true)
.create();
```

Genson avec des collections standards

- JSON objects <-> Java Map with String as keys
- JSON arrays <-> Java List or Array
- JSON numbers <-> Java Long or Double
- JSON string <-> Java String

Exemple

```
int[] arrayOfInts1 = new int[]{1, 2, 3};
String json = genson.serialize(arrayOfInts1);
// json = [1,2,3]

List arrayOfInts2 = genson.deserialize(json,List.class);
int[] arrayOfInts3 = genson.deserialize(json,int[].class);
```

- Notez que la transformation de Java->JSON est totalement automatique.
- Dans l'autre sens, il faut préciser vers quelle classe désérialiser.
  - En effet l'information de classe est nécessaire en Java mais n'existe pas en JSON

Mode POJO(Plain Old Java Object) /JavaBean

```
public class Person {
  private String name;
  private int age;
  private Address address;

  public Person() {} // constructeur sans argument

  public Person(String name, int age, Address address) {
    this.name = name;
    this.age = age;
    this.address = address;
  }
  // getters & setters
}

public class Address {
  public int building;
  public String city;

  public Address() {}

  public Address(int building, String city) {
    this.building = building;
    this.city = city;
  }
}
```

JavaBean ?

- Un JavaBean est un POJO (Plain Old Java Object) qui est sérialisable, a un constructeur sans arguments, et permet l'accès à des propriétés utilisant des méthodes getter et setter dont les noms sont déterminés par une convention simple.

Mode Pojo (Plain Old Java Object)

- Java -> JSON

```
Person someone =
        new Person("Eugen", 28, new Address(157, "Paris"));
String json = genson.serialize(someone);
```

Donne

{"address":{"building":157,"city":"Paris"},"age":28,"name":"Eugen"}

- Dans le sens JSON -> Java il faut préciser la classe Java cible !

```
Person person =
                genson.deserialize(json, Person.class);
```

Untyped (Object) Java Structures

```
Map<String, Object> p1 = new HashMap<String, Object>() {{
  put("name", "Foo");
  put("age", 28);
}};
String json = genson.serialize(p1);
// {"age":28,"name":"Foo"}
Map<String, Object> p2 = genson.deserialize(json, Map.class);
```

- Pendant la sérialisation s'il n'y a pas d'information de typage (type Object), Genson va utiliser le type de l'objet lors de l'exécution.
- Observez l'utilisation d'une classe anonyme avec un bloc d'initialisation lors de l'exécution.

(petit exemple de classe anonyme)

```
(instanciation d'une classe avec extension… ce sera une classe anonyme)
Personne etudiant=new Personne() {
    public String toString() {
        return "Etudiant "+super.toString();
    }
}
⇔ équivalent à, sauf que ci-dessus la classe n'a pas le nom Etudiant mais est anonyme)
class Etudiant extends Personne {
    public String toString() {
        return "Etudiant "+super.toString();
    }
}
Personne etudiant=new Etudiant();
```

(petit ex. d'un bloc d'initialisation)

- En plus d'un constructeur, une classe peut avoir un bloc d'initialisation entre {}

```
class Etudiant extends Personne {
   {setEmploi("étudiant");}
}
```

- Et donc les deux ensembles :

```
Map<String, Object> person =
                        new HashMap<String, Object>() {{
   put("name", "Foo");
   put("age", 28);
}};
```

A quoi sert une classe anonyme ?

- Elles permettent de déclarer et d'instantier une classe en même temps
- Les classes anonymes vous permettent d'écrire du code plus concis
- Elles sont utilisées souvent dans le contexte d'écrire un Listener

Genson est capable d'encore bien plus...

- http://owlike.github.io/genson/Documentation/UserGuide/
- http://owlike.github.io/genson/Documentation/Javadoc/
- Rappel : ce cours demande que vous consultiez par vous-même la documentation pour découvrir certains points non expliqués directement durant les présentations !