Acceso a Datos

UT2. MANEJO DE FICHEROS XML Y JSON
OBJETOS JSON

JSON

- JSON es un estándar abierto que utiliza texto plano para codificar información en la forma *atributo: valor*.

- Entre estas ventajas se encuentra la simplicidad con la que permite interactuar con arrays y el uso de objetos. Por sus características, en especial su simpleza y eficiencia, JSON se ha convertido en una muy interesante alternativa a XML.

JSON

XML	JSON

<empleados>
<empleado>
<nombre>Jorge</nombre>
<apellido>Mesa</apellido>
<edad>28</edad>
</empleado>
<empleado>
<nombre>Ana</nombre>
<apellido>Sánchez</apellido>
<edad>21</edad>
</empleado>
</empleados>
	{
	"empleados": {
	"empleado": [
	{
	"nombre": "Jorge",
	"apellido": "Mesa",
	"edad": 28
	},
	{
	"nombre": "Ana",
	"apellido": "Sánchez",
	"edad": 21
	}
	]
	}
	}

JSON - Librería

- Para poder realizar la serialización y deserialización entre objetos Java y su representación en notación JSON, es necesaria la librería de código abierto GSON (también conocido como Google Gson).
- Nos vamos a descargar la versión 2.10.1 de la siguiente página web https://mvnrepository.com/artifact/com.google.code.gson/gson/2.10.1 y vamos a añadirla a nuestro proyecto en eclipse.
- Project → Properties → Java Build Path → Add External JARS

JSON – Serialización objeto a JSON

```
import java.util.Arrays;

import com.google.gson.Gson;

public class Ejercicio1 {

    public static void main(String[] args) {
        Empleado emp1 = new Empleado("Pepe", "Lopez", 20,
                Arrays.asList("Gerente", "Jefe de zona"));
        Gson gson = new Gson();
        System.out.println(gson.toJson(emp1));
    }
}
```

```
{"nombre":"Pepe","apellidos":"Lopez","edad":20,"puestos":["Gerente","Jefe de zona"]}
```

JSON – Serialización objeto a JSON

- Si nos fijamos bien en el ejemplo anterior, la representación JSON del objeto Empleado viene bastante comprimida. Todo en una línea y sin espacios o tabulaciones. Es posible que, en algunos casos, sobre todo con objetos más complejos, queramos mostrar la representación JSON de una forma más clara.

- Gson nos permite crear representaciones JSON un poco más vistosas. Para ello debemos crear una instancia de **Gson** con **GsonBuilder**. Activamos el modo **PrettyPrinting** invocando al método **setPrettyPrinting** y creamos la instancia con el método **create**.

JSON – Serialización objeto a JSON

```java
import java.util.Arrays;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class Ejercicio2 {

    public static void main(String[] args) {
        Empleado emp1 = new Empleado("Pepe", "Lopez", 20,
                Arrays.asList("Gerente", "Jefe de zona"));
        Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();
        System.out.println(prettyGson.toJson(emp1));
    }

}
```

```json
{
  "nombre": "Pepe",
  "apellidos": "Lopez",
  "edad": 20,
  "puestos": [
    "Gerente",
    "Jefe de zona"
  ]
}
```

JSON – Serialización objeto a JSON

- Si queremos que aparezca el nombre del objeto en el JSON, tendríamos que añadir al objeto la anotación correspondiente de GSON. Para ello, nos creamos la clase SerializeEmpleado y le indicamos con la anotación @SerializedName que serialize el objeto con el nombre del empleado:

```
import com.google.gson.annotations.SerializedName;

public class SerializeEmpleado {

    @SerializedName("empleado")
    private Empleado empleado;

    public Empleado getEmpleado() {
        return empleado;
    }

    public void setEmpleado(Empleado empleado) {
        this.empleado = empleado;
    }

    public SerializeEmpleado(Empleado empleado) {
        super();
        this.empleado = empleado;
    }

}
```

JSON – Serialización objeto a JSON

```java
import java.util.Arrays;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class Ejercicio3 {

    public static void main(String[] args) {

        Empleado emp1 = new Empleado("Pepe", "Lopez", 20,
                Arrays.asList("Gerente", "Jefe de zona"));
        Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();
        // si queremos mostrar el nombre del objeto en el JSON
        SerializeEmpleado s_empleado = new SerializeEmpleado(emp1);
        System.out.println(prettyGson.toJson(s_empleado));

    }

}
```

```json
{
  "empleado": {
    "nombre": "Pepe",
    "apellidos": "Lopez",
    "edad": 20,
    "puestos": [
      "Gerente",
      "Jefe de zona"
    ]
  }
}
```

JSON – Deserializar JSON a objeto Java

- Es tan sencillo como crear un objeto **Gson** e invocar a su método **fromJson**. Como parámetros le pasaremos el objeto JSON como String y la clase del objeto en que se deserializará, en nuestro caso Empleado.class. Para poder deserializar correctamente el JSON a Empleados, los atributos se tienen que llamar igual.

```
import com.google.gson.Gson;

public class Ejercicio4 {

    public static void main(String[] args) {
        String json = "{\"nombre\":\"Pepe\",\"apellidos\":\"Lopez\","
                + "\"edad\":20,\"puestos\":[\"Gerente\",\"Jefe de zona\"]}";

        Gson gson = new Gson();
        Empleado emp1 = gson.fromJson(json, Empleado.class);
        System.out.println(emp1.toString());
    }

}
```

```
Empleado [nombre=Pepe, apellidos=Lopez, edad=20, puestos=[Gerente, Jefe de zona]]
```

JSON – Deserializar varios objetos

```
[  {
       "nombre": "Pepe",
       "apellidos": "Lopez",
       "edad": "20",
       "puestos": [
         "Gerente",
         "Jefe de zona"
       ]
     },
     {
       "nombre": "Maria",
       "apellidos": "Gutierrez",
       "edad": "30",
       "puestos": ["Jefa RR.HH"]
     }
   ]
```

JSON – Deserializar varios objetos

- Como vemos tenemos dos empleados en el array. Para que Gson comprenda que tiene que deserializar el objeto JSON en una lista de objetos propios hacemos lo mismo que en los ejemplos anteriores. La única diferencia es que ahora el segundo parámetro del método fromJSON no es la clase a la que queremos deserializar el objeto JSON sino un objeto Type (java.lang.reflect.Type) que habremos creado mediante TypeToken. Al crear una instancia de TypeToken, lo tipamos con la lista de "Empleado", invocamos a su método getType y ya tenemos nuestra instancia de Type.

- Para más información sobre TypeToken:

https://www.javadoc.io/doc/com.google.code.gson/gson/latest/com.google.gson/com/google/gson/reflect/TypeToken.html

JSON – Deserializar varios objetos

```
import java.lang.reflect.Type;
import java.util.Iterator;
import java.util.List;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

public class Ejercicio5 {

    public static void main(String[] args) {

        String json = "[{ \"nombre\": \"Pepe\",\"apellidos\": \"Lopez\", "
                + "\"edad\": 20,\"puestos\": [ \"Gerente\",\"Jefe de zona\"]},"
                + "{ \"nombre\": \"Maria\", \"apellidos\": \"Gutierrez\", "
                + "\"edad\": 30, \"puestos\": [\"Jefa RR.HH\"] }]";

        Gson gson = new Gson();
        Type tipoListaEmpleados = new TypeToken<List<Empleado>>() {
        }.getType();
        List<Empleado> empleados = gson.fromJson(json, tipoListaEmpleados);
        Iterator<Empleado> itemp = empleados.iterator();

        while (itemp.hasNext()) {
            System.out.println(itemp.next().toString());
        }
    }

}
```

JSON – Serializar varios objetos

- De manera similar a lo anterior podemos serializar varios objetos a JSON, a través de un ArrayList.

```
public static void ArrayToJSON() {
    ArrayList<Empleado> empleados = new ArrayList<Empleado>();
    Empleado emp1 = new Empleado("Pepe", "Lopez", 20,
            Arrays.asList("Gerente", "Jefe de zona"));
    Empleado emp2 = new Empleado("Maria", "Gutierrez", 30,
            Arrays.asList("Jefa de RR.HH"));
    empleados.add(emp1);
    empleados.add(emp2);
    Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();
    System.out.println(prettyGson.toJson(empleados));
}
```

Dudas y preguntas