Clase del día – 21/04/2021

En la clase de hoy vamos a explicar cómo utilizar **JSON** para serializar y des-serializar objetos.

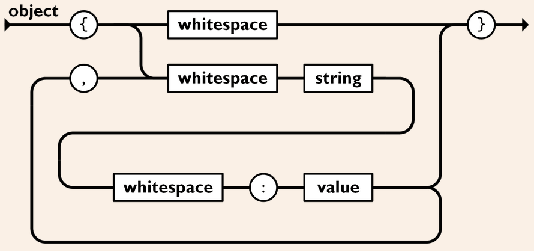
**JSON** (JavaScript Object Notation) es un formato texto para el intercambio de datos. JSON corresponde a la sintaxis utilizada en Javascript para escribir objetos.

JSON es un formato independiente del lenguaje de programación, de manera que es posible escribir fácilmente programas en cualquier lenguaje que creen mensajes en formato JSON así como programas que lean mensajes en formato JSON.

En JSON es posible crear dos estructuras: objetos y arreglos.

Un **objeto**es una colección no ordenada de parejas nombre:valor separadas por coma. Un objeto comienza con una llave que abre “{“ y termina con una llave que cierra “}”.

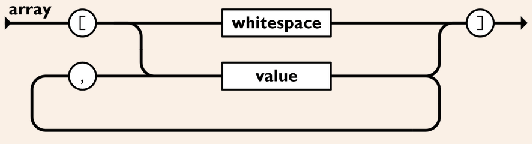
La sintaxis de un objeto es la siguiente:



 Fuente: www.json.org

Un **arreglo**es una colección ordenada de valores separados por coma. Un arreglo comienza con un corchete que abre “[“ y termina con un corchete que cierra “]”.

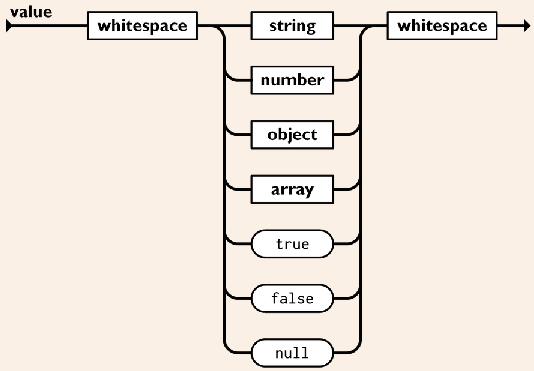
La sintaxis de un arreglo es la siguiente:



Fuente: www.json.org

Un **valor**puede ser una cadena de caracteres entre comillas, o un número, o un objeto, o un arreglo, o las constantes true, false o null.

La sintaxis de un valor es la siguiente:



Fuente: www.json.org

Una **cadena de caracteres** (string) es una secuencia de cero o más caracteres Unicode encerrados entre comillas.

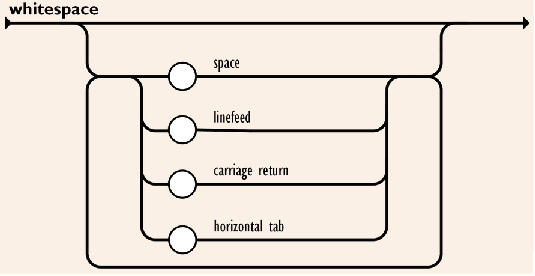
Una cadena de caracteres puede contener las siguientes secuencias de escape:

----------------------------------------------------------------  
Secuencia  
de escape   Descripción  
----------------------------------------------------------------  
\"                comillas  
\\                diagonal inversa  
\/                diagonal  
\b               back space  
\f                form feed  
\n               line feed  
\r                carriage return  
\t                tabulador  
\uxxxx        caracter unicode con código hexadecimal xxxx

----------------------------------------------------------------

Un **número**sigue la sintaxis de los números decimales en lenguaje C.

Un **whitespace**es un separador de tokens, de acuerdo a la siguiente sintaxis:



Fuente: www.json.org

Ahora veremos un ejemplo, utilizando GSON (implementación de JSON desarrollada por Google).

Se requiere descargar el archivo gson-2.8.6.jar de la siguiente URL:

<https://repo1.maven.org/maven2/com/google/code/gson/gson/2.8.6/gson-2.8.6.jar>

Descargar de la plataforma el programa [EjemploGSON.java](https://m4gm.com/moodle/mod/resource/view.php?id=1819) y colocarlo en el mismo directorio dónde está el archivo gson-2.8.6.jar que se descargó anteriormente.

Para compilar el programa [EjemploGSON.java](https://m4gm.com/moodle/mod/resource/view.php?id=1819) se debe ejecutar el siguiente comando:

javac -cp gson-2.8.6.jar [EjemploGSON.java](https://m4gm.com/moodle/mod/resource/view.php?id=1819)

Para ejecutar el programa en Windows:

java -cp gson-2.8.6.jar;. EjemploGSON

Para ejecutar el programa en Linux:

java -cp gson-2.8.6.jar:. EjemploGSON

Ahora se explicará como funciona este programa.

Primeramente se declaran los imports de las clases que se utilizarán: Gson, GsonBuilder y Timestamp.

Se define una clase Empleado que contiene los campos: nombre, edad, sueldo y fecha\_ingreso. Por conveniencia se define un constructor para inicializar los campos al crear una instancia.

Notar que la fecha se maneja como fecha-hora debido a que en los sistemas globales (Internet) la fecha no indica qué sucede antes y que sucede después a nivel mundial; recordar lo que se vimos sobre tiempo UTC y tiempo local.

En la función main se crea un arreglo de 3 empleados, cada elemento se inicializa con una instancia de la clase Empleado, con diferentes datos.

Entonces se crea una instancia de la clase Gson. Notar que se utiliza el método setDateFormat para utilizar el formato de fecha ISO8601, el cual es el formato que utiliza Javascript para manejar fecha-hora.

Después se utiliza el método **toJson**de la clase Gson para serializar el arreglo de empleados. Esta clase produce la siguiente string:

[{"nombre":"Hugo","edad":20,"sueldo":1000.0,"fecha\_ingreso":"2020-01-01T20:10:00.000"},{"nombre":"Paco","edad":21,"sueldo":2000.0,"fecha\_ingreso":"2019-10-01T10:15:00.000"},{"nombre":"Luis","edad":22,"sueldo":3000.0,"fecha\_ingreso":"2018-11-01T00:00:00.000"}]

Finalmente se utiliza el método **fromJson**de la clase Gson para deserializar la string anterior y producir un nuevo arreglo de empleados. Entonces se despliegan los datos de los empleados:

Hugo 20 1000.0 2020-01-01 20:10:00.0  
Paco 21 2000.0 2019-10-01 10:15:00.0  
Luis 22 3000.0 2018-11-01 00:00:00.0

Actividades individuales a realizar

1. Compilar y ejecutar el programa [EjemploGSON.java](https://m4gm.com/moodle/mod/resource/view.php?id=1819)

2. Agregar un nuevo campo a la clase Empleado, el campo "jefe" de tipo Empleado.

3. Modificar el constructor de la clase Empleado para incluir la inicialización del campo "jefe", tal como se muestra:

  static class Empleado  
  {  
    String nombre;  
    int edad;     
    float sueldo;  
    Timestamp fecha\_ingreso;  
    Empleado jefe;  
    Empleado(String nombre,int edad,float sueldo,Timestamp fecha\_ingreso,Empleado jefe)  
    {  
      this.nombre = nombre;  
      this.edad = edad;  
      this.sueldo = sueldo;  
      this.fecha\_ingreso = fecha\_ingreso;  
      this.jefe = jefe != null ? jefe : this;  
    }  
  }

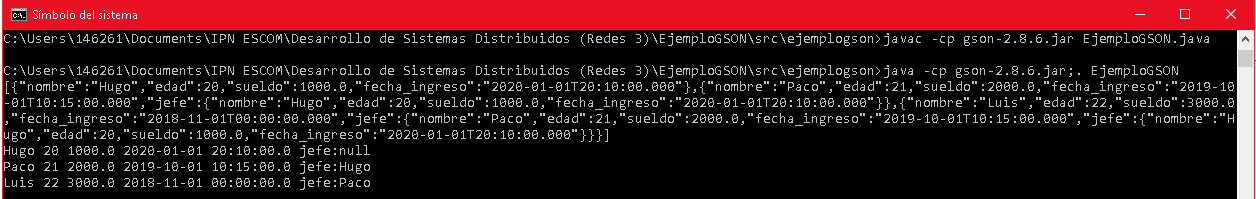
4. Crear los tres empleados de manera que Hugo sea jefe de si mismo (Hugo es el jefe máximo), Hugo sea jefe de Paco, y Paco sea jefe de Luis, tal como se muestra:

    Empleado[] e = new Empleado[3];  
    e[0] = new Empleado("Hugo",20,1000,Timestamp.valueOf("2020-01-01 20:10:00"),null);  
    e[1] = new Empleado("Paco",21,2000,Timestamp.valueOf("2019-10-01 10:15:00"),e[0]);  
    e[2] = new Empleado("Luis",22,3000,Timestamp.valueOf("2018-11-01 00:00:00"),e[1]);

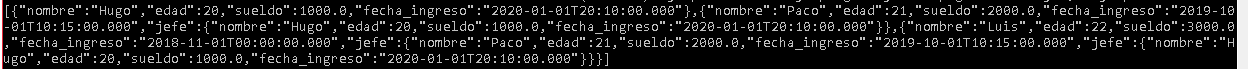
5. Cambiar el despliegue del arreglo deserializado, de la siguiente manera:

    for (int i = 0; i < v.length; i++)  
      System.out.println(v[i].nombre + " " + v[i].edad + " " + v[i].sueldo + " " + v[i].fecha\_ingreso + " jefe:" + (v[i].jefe != null ? v[i].jefe.nombre : null));

6. Compilar y ejecutar el programa.



6.1 ¿Qué despliega la serialización?



6.2 Al serializar el empleado Hugo ¿Se muestra la auto-referencia que hace el empleado Hugo a sí mismo? No se muestra

6.3 ¿Hay alguna redundancia en los objetos serializados? Cuando se hace referencia a los jefes, se vuelven a desplegar todos los campos que conforman al objeto Empleado de dicho jefe. Esto es más notable en el caso de Luis, pues tanto los campos de Paco y de Hugo vuelven a mostrarse.

6.4 ¿Qué despliega la deserialización?



6.5 ¿Qué concluye sobre GSON y la forma en que serializa y deserializa las referencias a objetos? Esta resulta ser una manera muy efectiva para realizar un envío o recibimiento de los objetos y datos creados a partir de Java, pues nos brinda una manera en la que podemos enviarlos con mayor facilidad

7. Ver el video: https://www.youtube.com/watch?v=6M1gUOSzaA4&t=4s&ab\_channel=DesarrolloWeb.com