Indice

Lectura de JSON

Antes de comenzar, añadimos la dependencia en el pom.xml. La librería GSON permite convertir los objetos y arrays de JSON a clases de Java.

```
<dependencies>
        <dependency>
            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20240303</version>
        </dependency>

        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.11.0</version>
        </dependency>
    </dependencies>
```

Una vez tenemos la dependencia, para leer un JSON, debemos seguir los siguietnes pasos:

1. Crear un objeto URL con la dirección del JSON -en este ejemplo vamos a hacer a través de una petición Http.
2. Creamos un objeto BufferReader para leer el JSON, como hemos visto en los pasos anteriores.
3. Adicionalmente, vamos a crear un stringBuffer, que es un buffer de cadenas de texto. En lugar de guardar en un bufferWriter/ printWriter para ir enviándolo a un fichero o en un StringBuilder (para ir construyendo el texto), vamos a guardar en un StringBuffer.
4. El siguiente paso es crear un objeto JSONObject con el contenido del StringBuffer, sabiendo que cada linea del JSON SOLAMENTE ES UNA LÍNEA del JSON, hasta que no se lee completamente, el JSON no está completo.

5. Una vez que se ha leído todo el archivo, el contenido del StringBuffer (que es un String) se convierte en un objeto JSONObject. Si visitamos la url facilitada, veremos que efectivamente el contenido es un JSON Object, ya que tiene la estructura { "clave" : "valor" }. Pero además, veremos que dentro de ese JSON Object hay un JsonArray de PRODUCTS, este es el objeto que después debemos iterar.
6. Una vez tenemos el JSONObject, podemos acceder a los distintos elementos del JSON. En este caso, vamos a acceder a los productos, que es un JSONArray. Si quisiéramos acceder a un Json Object que esté anidado, podríamos hacerlo con el método getJSONObject("clave").
7. Podemos hacer un casting, o podemos iterar directamente sobre JsonObject en vez de en object.
8. A partir de aquí, el trabajo es repetir una y otra vez tantas veces como queramos acceder a los objetos anidados dentro del JSON. Por ejemplo, nosotros vamos a obtener el JSON Array que está dentro de cada producto e iterarlo, a su vez, para sacar una lista de categorías.

```
import com.google.gson.Gson;
import org.json.JSONArray;
import org.json.JSONObject;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Scanner;

public class PeticionJSON {

    public void procesarPeticion() {

        // URL -> HTTPCONNECTION -> BUFFEREADER
        String urlString = "https://dummyjson.com/products";
        try {
            URL url = new URL(urlString);
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
            String linea = null;
            StringBuffer stringBuffer = new StringBuffer();
            while ((linea = bufferedReader.readLine()) != null) {
                stringBuffer.append(linea);
            }

            JSONObject peticionProducto = new JSONObject(stringBuffer.toString());
            JSONArray listaProductos = peticionProducto.getJSONArray("products");

            ArrayList<String> categorias = new ArrayList<>();
            for (Object item : listaProductos) {
                // item es un JSONObject -> YO LO SÉ, así que podríamos iterar for
(JSONObject item : listaProductos)
                JSONObject producto = (JSONObject) item;
                System.out.println("Titulo: " + producto.getString("title"));
                System.out.println("Precio: " + producto.getDouble("price"));
```

```
                JSONArray tags = producto.getJSONArray("tags");
                System.out.println("Las categorias del producto son:");
                for (Object tag : tags) {
                    //System.out.println("\t"+tag);
                    categorias.add(tag.toString());
                }

            }


        } catch (MalformedURLException e) {
            System.out.println("No es una web, por favor intentalo de nuevo");
        } catch (IOException e) {
            System.out.println("Error en la pagina, no responde");
        }

    }
}
```

Exportación de GSON a Objecto de Java

La librería de GSON nos permite convertir directamente un JSON a un objeto de Java con el método new Gson.fromJson().

El proceso inverso sería a un objeto, decirle new Gson.toJson() y convertirlo en un JSONN, que en realidad es texto plano.

```
private void exportarDatos() {

        try {

           // Obtenemos los productos nuevamente
            URL url = new URL("https://dummyjson.com/products");
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
            JSONObject jsonObject = new JSONObject(bufferedReader.readLine());
            JSONArray jsonArray = jsonObject.getJSONArray("products");


            for (int i = 0; i < jsonArray.length(); i++) {
                Producto producto = new
Gson().fromJson(jsonArray.getJSONObject(i).toString(), Producto.class);
                String exportacionProducto = String.format("title:%s price:%.2f
stock:%d", producto.getTitle(),
                        producto.getPrice(), producto.getStock());

            // PROCESO INVERSO: Convirtiendo de un objeto de Java a un JSON. Como
vemos, no tiene misterio, es solo convertir en texto plano.
            String productoJson = new Gson().toJson(producto);
```

```
            }

        } catch (MalformedURLException e) {
            System.out.println("Error en URL");
        } catch (IOException e) {
            System.out.println("Error en la creacion de la escritura");
        } finally {
            try {
                printWriter.close();
            } catch (NullPointerException e) {
                System.out.println("Error en el cerrado");
            }
        }


    }
```

Escritura de ficheros

Dentro de Java existen múltiples formas de leer y escribir ficheros. En este apartado se explicarán las más comunes.

FileWriter y FileReader:

Permite escribir y leer caracteres en un fichero. Es el sistema más básico de escritura de ficheros en TEXTO PLANO (no es binario, como el FileOutputStream). Es poco efciente, así que se recomienda usar la envolvente de BufferedWriter (siguiente método que veremos y que se comporta EXACTAMENTE IGUAL salvo por el constructor).

```
    public class FileWriterExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String path = "ruta/del/fichero/"; // especifica el directorio
        File file = new File(path + "nombre-fichero.txt");


        System.out.println("Por favor introduce lo que quieres guardar:");
        String lecturaFrase = scanner.nextLine();


        FileWriter fileWriter = null;
        try  {

            // Antes de nada, verificar que el archivo existe
            if (!file.exists()) {
                System.out.println("Creando fichero ");
                file.createNewFile();
            }
```

```
            fileWriter = new FileWriter(file, true); // append -> anexar o no la
escritura
            fileWriter.write("Linea número 1" + lecturaFrase + "\n");
            fileWriter.newLine();
            fileWriter.write("Linea número 2" + lecturaFrase + "\n");
            fileWriter.newLine();
            fileWriter.write("Linea número 3" + lecturaFrase + "\n");
            fileWriter.newLine();
            System.out.println("Texto guardado correctamente en el archivo.");
        } catch (IOException e) {
            System.out.println("Error en la escritura del fichero, verifica
permisos o la ruta.");
        } finally {
            if (fileWriter != null) {
                try {
                    fileWriter.close();
                } catch (IOException e) {
                    System.out.println("Error al cerrar el archivo.");
                }
            }
        }
    }
}
```

BufferedReader y **BufferedWriter**

La versión rápida de FileReader y FileWriter. Son EXACTAMENTE iguales que las anteriores, pero más rápidas. En lugar de hacerlo caracter a caracter, trabajan línea a línea. BufferedWriter utiliza un buffer interno donde almacena las líneas temporalmente. Luego, cuando el buffer se llena o se cierra, el contenido se escribe en el archivo en una sola operación de escritura. Esto mejora significativamente el rendimiento, especialmente cuando escribes muchas líneas de texto.

```
    public class BufferedWriterExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String path = "ruta/del/fichero/"; // especifica el directorio
        File file = new File(path + "nombre-fichero.txt");

        System.out.println("Por favor introduce lo que quieres guardar:");
        String lecturaFrase = scanner.nextLine();

        // Declarar BufferedWriter fuera del bloque try
        BufferedWriter bufferedWriter = null;

        try {
            // Antes de nada, verificar que el archivo existe. En este código,
damos por hecho que el directorio existe.
            // En otras circunstancias, meteríamos también un bloque para crear el
directorio si no existiese.
```

```
            if (!file.exists()) {
                System.out.println("Creando fichero ");
                file.createNewFile();
            }

            // Envolvemos FileWriter con BufferedWriter
            bufferedWriter = new BufferedWriter(new FileWriter(file, true)); //
'true' para agregar al contenido existente
            bufferedWriter.write("Linea número 1" + lecturaFrase);
            bufferedWriter.newLine();
            bufferedWriter.write("Linea número 2" + lecturaFrase + "\n");
            bufferedWriter.newLine();
            bufferedWriter.write("Linea número 3" + lecturaFrase + "\n");
            bufferedWriter.newLine();
            System.out.println("Texto guardado correctamente en el archivo.");
        } catch (IOException e) {
            System.out.println("Error en la escritura del fichero, verifica
permisos o la ruta.");
        } finally {
            if (bufferedWriter != null) {
                try {
                    bufferedWriter.close();
                } catch (IOException e) {
                    System.out.println("Error al cerrar el archivo.");
                }
            }
        }
    }
}
```

PrintWriter:

Esta clase básicamente hace salots de linea con println. Aparte de eso, NO tiene la opción de append. Para usar este, mejor usa el Buffered.

```
    public class PrintWriterExample {
        public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            String path = "ruta/del/fichero/"; // especifica el directorio
            File file = new File(path + "nombre-fichero.txt");

            System.out.println("Por favor introduce lo que quieres guardar:");
            String lecturaFrase = scanner.nextLine();

            PrintWriter printWriter = null;
            try {
                // Antes de nada, verificar si el archivo existe
                if (!file.exists()) {
                    System.out.println("Creando fichero ");
```

```
                    file.createNewFile();
                }

                // Crear un PrintWriter, NO TIENE OPCIÓN DE APPEND.
                printWriter = new PrintWriter(file);
                printWriter.println("Linea número 1" + lecturaFrase);
                printWriter.println("Linea número 2" + lecturaFrase);
                printWriter.println("Linea número 3" + lecturaFrase);

                System.out.println("Texto guardado correctamente en el archivo.");
            } catch (FileNotFoundException e) {
                System.out.println("Error: archivo no encontrado.");
            } catch (IOException e) {
                System.out.println("Error al escribir en el archivo: " +
e.getMessage());
            } finally {
                if (printWriter != null) {
                    printWriter.close(); // No es necesario capturar IOException
aquí
                }
            }
        }
    }
```

FileInputStream y **FileOutputStream** (binarios y texto ASCII)

Los FileInputStream y FileOutputStream son clases de Java que permiten trabajar con archivos a nivel de bytes, es decir, se utilizan para leer y escribir datos binarios -si son imagenes,videos, etc- o en ASCII, si son cadenas de texto.

No los vamos a usar como tal, los veremos a continuación con el ObjectInputStream y ObjectOutputStream. Igualmente, hay que saber como funcionan:

```
public class ASCIIFileWriterExample {
    public static void main(String[] args) {
        String path = "ruta/del/fichero/";  // especifica la ruta del archivo
        FileOutputStream fileOutputStream = null;

        try {
            // Crear el archivo
            fileOutputStream = new FileOutputStream(path + "ascii.txt");

            // Escribir valores ASCII directamente (por ejemplo, 'A' -> 65, 'B' ->
66)
            String texto = "Hola Mundo";
            for (int i = 0; i < texto.length(); i++) {
                // Convertir cada carácter a su código ASCII y escribirlo
                fileOutputStream.write((int) texto.charAt(i));
            }
```

```
            System.out.println("Texto escrito correctamente en el archivo como
códigos ASCII.");

        } catch (IOException e) {
            System.out.println("Error al escribir en el archivo: " +
e.getMessage());
        } finally {
            try {
                if (fileOutputStream != null) {
                    fileOutputStream.close();
                }
            } catch (IOException e) {
                System.out.println("Error al cerrar el archivo.");
            }
        }
    }
}
```

ObjectInputStream y ObjectOutputStream

Estas clases son para leer OBJETOS SERIALIZADOS que estén en binario dentro de un archivo Obj.

Para poder usarlas, será necesario tirar del interfaz Serializable, que es una interfaz de marcado que no tiene métodos. Simplemente indica que la clase que la implementa puede ser serializada.

Además de eso, tendremos que hacer la envolvente al FileOutputStream y FileInputStream para hacer la lectura.

```
    public class ObjectOutputStreamExample {
        public static void main(String[] args) {
            String path = "ruta/del/fichero/"; // especifica el directorio
            File file = new File(path + "nombre-fichero.obj");

            List<Item> SampleList = new ArrayList<>();

            ObjectOutputStream objectOutputStream = null;
            try{
                if (!file.exists()) {
                    System.out.println("Creando fichero ");
                    file.createNewFile();
                }
                objectOutputStream = new ObjectOutputStream(new
FileOutputStream(file));
                objectOutputStream.writeObject(SampleList);

            } catch (FileNotFoundException e) {
                throw new RuntimeException("No existe el archivo: " +
e.getMessage(), e);
            } catch (IOException e) {
                throw new RuntimeException("Error de escritura: " +
```

```
e.getMessage(), e);
            } finally {
                try {
                    objectOutputStream.close();
                } catch (IOException | NullPointerException e) {
                    System.out.println("Error al cerrar");
                }
            }
        }
    }
```

Lectura de ficheros

Para la lectura de ficheros tenemos nuevamente el FileReader y el BufferReader.

Sobre el FileReader tenemos que saber que la lectura se hace CARACTER A CARACTER. Lo cuál hace que sea INEFICIENTE. Por esa razón, SIEMPRE SIEMPRE SIEMPRE vamos a envolver el FileReader dentro de un Buffer Reader, y nos quedaría así:

```
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class FileReaderExample {
    public static void main(String[] args) {
        String path = "ruta/del/fichero/"; // especifica el directorio
        File file = new File(path + "nombre-fichero.txt");


        FileReader fileReader = null;
        BufferedReader bufferedReader = null;

        try {

            if (!file.exists()) {
                System.out.println("El archivo no existe.");
                return;
            }

            // Crear FileReader y envolverlo en BufferedReader
            fileReader = new FileReader(file);
            bufferedReader = new BufferedReader(fileReader);


            StringBuilder lecturaCompleta = new StringBuilder();

            String linea;
```

```
            while ((linea = bufferedReader.readLine()) != null) {

                // Imprimir cada línea leída... Pero imprimir linea a linea es una
locura. ¿Por qué no imprimir todo junto?
                System.out.println(linea);

                // Usamos el StringBuilder para recomponer el texto completo
                lecturaCompleta.append(lectura);
                lecturaCompleta.append("\n");
            }

             System.out.println(lecturaCompleta.toString());

        } catch (IOException e) {
            System.out.println("Error en la lectura del fichero: " +
e.getMessage());
        } finally {
            // Cerrar el BufferedReader y FileReader
            try {
                if (bufferedReader != null) {
                    bufferedReader.close();
                }
                if (fileReader != null) {
                    fileReader.close();
                }
            } catch (IOException e) {
                System.out.println("Error al cerrar el archivo.");
            }
        }
    }
}
```

NO HACER ASÍ: Si aplicásemos el FileReader directamente sería algo así:

```
    int character;
    while ((character = fileReader.read()) != -1) {
        System.out.print((char) character); // Imprimir cada carácter leído
    }
```

Lectura con ObjectInputStream

Esta es la clase usada para leer objetos SERIALIZADOS. Si por algún casual intentamos leer un archivo que no está serializado, nos saltará una excepción. Igualmente, si modificamos el objeto serializado, también nos saltará una excepción... básicamente se nos va a corromper el archivo.

Nota Como apreciación importante, podemos ver que si el archivo no existe, directamente la función acabe. No tiene sentido leer un archivo serializado que está vacío, ya que si se está leyendo es porque se espera que tenga objetos serializados dentro.

```
    public void ObjectInputStreamSample() {
        File file = new File("ruta/archivo.obj");

        ObjectInputStream objectInputStream = null;
        try {
            if (!file.exists()) {
                System.out.println("El archivo no existe.");
                return;
            }

            objectInputStream = new ObjectInputStream(new FileInputStream(file));

            // Leer el objeto y hacer el cast apropiado
            concesionarioList = (ArrayList<Coche>) objectInputStream.readObject();

        } catch (FileNotFoundException e) {
            System.err.println("La ruta del archivo de entrada no existe: " +
e.getMessage());
        } catch (IOException e) {
            System.err.println("Error de entrada/salida: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("Error de clase: " + e.getMessage());
        } finally {
            // Asegurarse de cerrar el flujo de datos para liberar recursos
            try {
                if (objectInputStream != null) {
                    objectInputStream.close();
                }
            } catch (IOException e) {
                System.err.println("Error al cerrar el archivo: " +
e.getMessage());
            }
        }
    }
```

Clases serializadas

Como bonustrack, tenemos las clases serializadas. Al respecto solo mencionaremos que tienen estos dos distintivos:

- implements Serializable
- private static final long serialVersionUID = 1L;

La serialización es privada, estática (de la clase), final (no se puede modificar) y se asigna un valor único a cada versión del objeto. Si no se asigna, Java lo hace automáticamente. Si se cambia la clase, se debe cambiar el serialVersionUID.

```java
import java.io.Serializable;
import java.util.Objects;

public class Item implements Serializable {
    private static final long serialVersionUID =  1L;
    private static int nextId = 1;

    private int id;
    private String property1;
    private String property2;
    // demás atributos de clase


    //Demás métodos, constructores, etc.
}
```