

# Sesión de desarrollo 2. Java y Node Js

Jesus Rodolfo Izurieta Veliz

June 21, 2020

## Contents

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Desarrollo</b>	<b>2</b>
2.1	Cliente Java . . . . .	2
2.1.1	Bibliotecas importadas . . . . .	2
2.1.2	Obtención del archivo . . . . .	3
2.1.3	Entrada y salida . . . . .	4
2.1.4	Compilación . . . . .	4
2.1.5	Pruebas . . . . .	5

# 1 Introducción

## Enunciado del proyecto

Desarrolle los dos programas que se indican a continuación y redacte un informe que avale las soluciones obtenidas y el trabajo realizado.

1. Cliente para download: Desarrolle un cliente en Java para descargar archivos de su servidor Node.JS. Su programa cliente deberá recibir como argumento la URL del archivo que se desea descargar.

Ejemplo: `java miprog http://localhost/doc/miarchivo.pdf` Verifique que la descarga es correcta. Por ejemplo, si el archivo que solicito es un pdf debería ser abierto por Acrobat.

1. Cliente Node: Probemos ahora, las capacidades de Node.js para desarrollar un programa cliente, escriba un programa en Node.js con las mismas características que el descrito en el apartado anterior. Se sugiere usar los módulos “http”, “url” y “fs”.

# 2 Desarrollo

## 2.1 Cliente Java

Primeramente creamos la clase Client

### 2.1.1 Bibliotecas importadas

Para el desarrollo del cliente necesitaremos las siguientes bibliotecas:

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.Socket;
import java.io.BufferedInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
```

La clase `BufferedInputStream` nos permitirán manejar el stream de datos que interpretaremos como el archivo descargado desde el servidor. Este archivo, lo podremos guardar haciendo uso de la clase `FileOutputStream`. `HttpURLConnection` nos permitirá realizar una conexión http con el servidor, usando la url que instanciaremos usando la clase `URL`. Finalmente, la

clase `IOException` nos permitirá obtener las excepciones provenientes de la lectura del archivo.

### 2.1.2 Obtención del archivo

Ahora desarrollamos el método estático `getFile` que recibirá la url del archivo y el nombre con el que será guardado en el disco

```
public static void getFile(String url, String filename){

    try{
        URL url = new URL(url);

        try (
            BufferedInputStream in = new BufferedInputStream(
                url.openStream()
            );

            FileOutputStream fileOutputStream = new FileOutputStream(filename)
        ) {
            byte dataBuffer[] = new byte[1024];
            int bytesRead;

            while ((bytesRead = in.read(dataBuffer, 0, 1024)) != -1)
            {
                fileOutputStream.write(dataBuffer, 0, bytesRead);
            }
        } catch (IOException e) {
            // handle exception
        }

    } catch (Exception e){}
}
```

Primeramente crearemos una instancia de la clase `URL`, con la que crearemos un stream, que pasemos como parámetro de un `BufferedInputStream`. El stream será leído desde el servidor byte a byte, y quedará almacenado en memoria, para guardarlo en un archivo, usaremos la clase `FileOutputStream`, que almacenará el stream en un archivo con el nombre que le pasemos, después de que este termine de descargarse desde el servidor.

### 2.1.3 Entrada y salida

La entrada de la url debe obtenerse desde la línea de comandos y ya que la respuesta de la petición será un archivo, almacenaremos este en un archivo en el computador del cliente, con un nombre de archivo que también obtendremos de la línea de comandos.

Para obtener la url y el nombre del archivo desde la línea de comandos, tendremos que obtener los parámetros desde stdin. Podremos hacer esto desde el método main mediante el parámetro args, que devuelve un vector de cadenas, del que necesitaremos los primeros dos elementos.

```
public static void main(String[] args) {  
  
    String url = args[0];  
    String filename = args[1];  
}
```

Seguidamente, ya obtenidos estos dos datos, podremos llamar a la función estática getFile que definimos en la clase Client, para esto, añadimos una línea de código de modo que nuestro método main queda como sigue:

```
public static void main(String[] args) {  
  
    String url = args[0];  
    String filename = args[1];  
  
    Client.getFile(url, filename);  
}
```

Con esto el programa está listo para ser compilado.

### 2.1.4 Compilación

Para compilar una clase java, usaremos el programa javac, ejecutando el comando:

```
javac Client.java
```

Con lo que se generará un nuevo archivo Client.class, que podremos ejecutar con java ejecutando el siguiente comando:

```
java Client argumento1 argumanto2
```

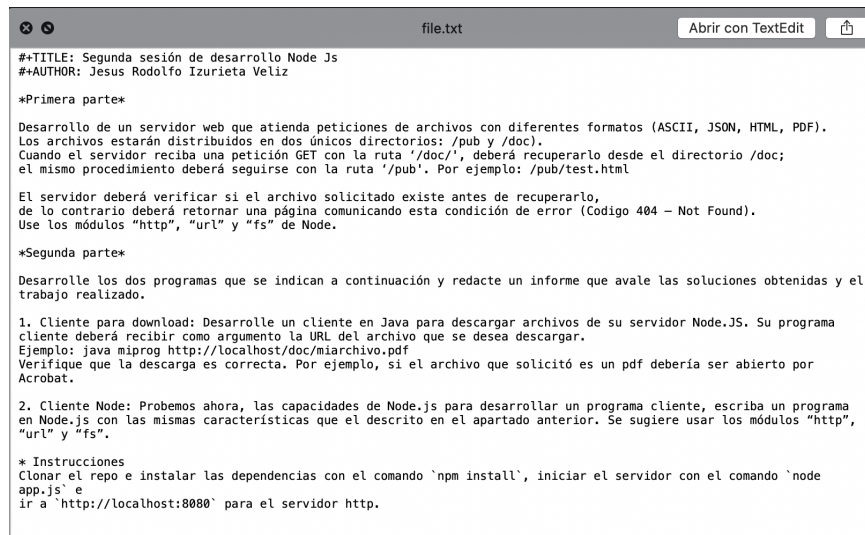
Tras la compilación, pasaremos a realizar algunas pruebas del programa.

### 2.1.5 Pruebas

Después de compilar el archivo Client.java, podremos ejecutarlo, pasando como parámetros la url del archivo que queremos obtener y el nombre con que guardaremos el archivo obtenido. Ejecutamos el comando en la consola:

```
java Client > java Client https://raw.githubusercontent.com/izurietajr/sesion2/master/README.org file.txt
java Client > qq
```

Tras la ejecución, deberíamos obtener el archivo file.txt en el directorio desde el que ejecutamos el comando:



```
file.txt
Abrir con TextEdit

#+TITLE: Segunda sesión de desarrollo Node Js
#+AUTHOR: Jesus Rodolfo Izurieta Veliz

*Primera parte*

Desarrollo de un servidor web que atienda peticiones de archivos con diferentes formatos (ASCII, JSON, HTML, PDF).
Los archivos estarán distribuidos en dos únicos directorios: /pub y /doc).
Cuando el servidor reciba una petición GET con la ruta '/doc/', deberá recuperarlo desde el directorio /doc;
el mismo procedimiento deberá seguirse con la ruta '/pub'. Por ejemplo: /pub/test.html

El servidor deberá verificar si el archivo solicitado existe antes de recuperarlo,
de lo contrario deberá retornar una página comunicando esta condición de error (Codigo 404 - Not Found).
Use los módulos "http", "url" y "fs" de Node.

*Segunda parte*

Desarrolle los dos programas que se indican a continuación y redacte un informe que avale las soluciones obtenidas y el
trabajo realizado.

1. Cliente para download: Desarrolle un cliente en Java para descargar archivos de su servidor Node.JS. Su programa
cliente deberá recibir como argumento la URL del archivo que se desea descargar.
Ejemplo: java miprog http://localhost/doc/miarchivo.pdf
Verifique que la descarga es correcta. Por ejemplo, si el archivo que solicitó es un pdf debería ser abierto por
Acrobat.

2. Cliente Node: Probemos ahora, las capacidades de Node.js para desarrollar un programa cliente, escriba un programa
en Node.js con las mismas características que el descrito en el apartado anterior. Se sugiere usar los módulos "http",
"url" y "fs".

* Instrucciones
Clonar el repo e instalar las dependencias con el comando `npm install`, iniciar el servidor con el comando `node
app.js` e
ir a `http://localhost:8080` para el servidor http.
```