

INSTITUTO POLITÉCNICO NACIONAL



ESCUELA SUPERIOR DE CÓMPUTO

APLICACIONES PARA SERVICIOS EN RED

PACTICA No. 4

SERVIDOR WEB

PROFESOR:

MORENO CERVANTES AXEL ERNESTO

ALUMNOS:

GURROLA SANCHEZ JOEL

TENORIO ALVA ALAN OSMANI

GRUPO: 3CM16

Introducción

Servidor Web

Un servidor web (server) es un ordenador de gran potencia que se encarga de “prestar el servicio” de transmitir la información pedida por sus clientes (otros ordenadores, dispositivos móviles, impresoras, personas, etc.)

Los servidores web (web server) son un componente de los servidores que tienen como principal función almacenar, en web hosting, todos los archivos propios de una página web (imágenes, textos, videos, etc.) y transmitirlos a los usuarios a través de los navegadores mediante el protocolo HTTP (Hypertext Transfer Protocol).

Métodos HTTP más comunes

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque también pueden ser sustantivos, estos métodos de solicitud a veces son llamados verbos HTTP. Cada uno de ellos implementan una semántica diferente, pero algunas características son similares compartidas por un grupo de ellos: ej. Un método de solicitud puede ser seguro, idempotente (en-US) o almacenable en caché .

GET

El método GET solicita una representación de un recurso específico. Las peticiones que usen el método GET sólo deben recuperar datos.

HEAD

El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

POST

El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

PUT

El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

DELETE

El método DELETE borra un recurso en específico.

Archivo ServidorWeb.java

Este archivo lee un puerto, donde correrá la aplicación. Lee un pool de conexiones y acepta la conexión de clientes. Se aprecia de mejor manera en la Figura 2.

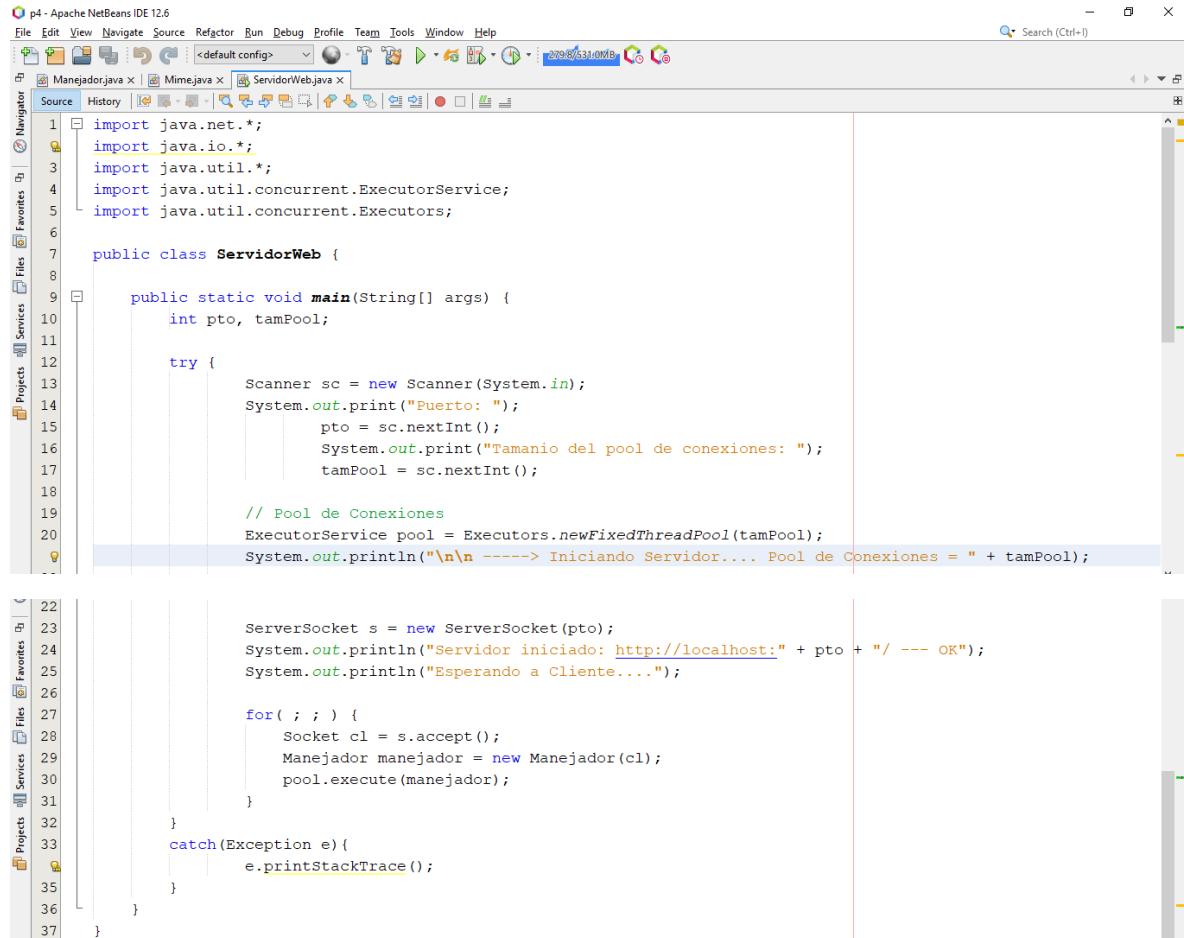
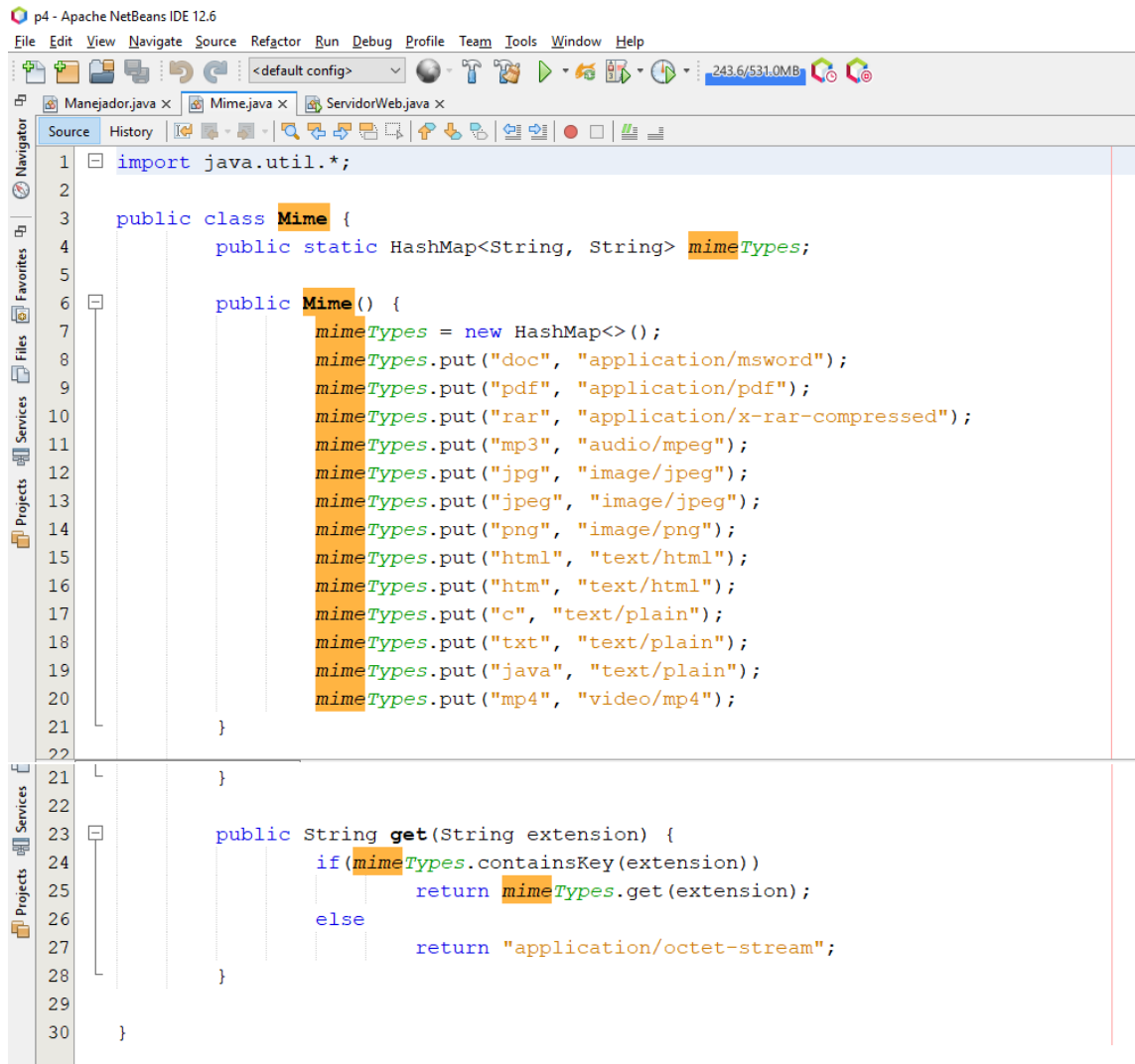


Figura 1

Archivo Mime.java

Este archivo indica el tipo de contenido de la respuesta del servidor, todas las extensiones de la respuesta se ven en la Figura 3.

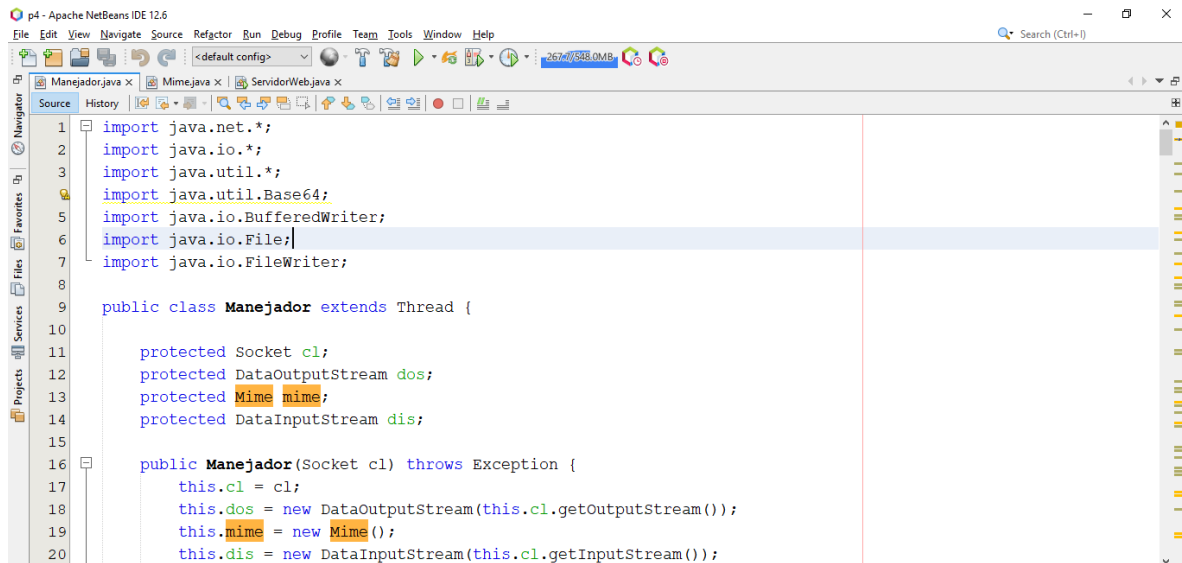


```
1 import java.util.*;
2
3 public class Mime {
4     public static HashMap<String, String> mimeTypes;
5
6     public Mime() {
7         mimeTypes = new HashMap<>();
8         mimeTypes.put("doc", "application/msword");
9         mimeTypes.put("pdf", "application/pdf");
10        mimeTypes.put("rar", "application/x-rar-compressed");
11        mimeTypes.put("mp3", "audio/mpeg");
12        mimeTypes.put("jpg", "image/jpeg");
13        mimeTypes.put("jpeg", "image/jpeg");
14        mimeTypes.put("png", "image/png");
15        mimeTypes.put("html", "text/html");
16        mimeTypes.put("htm", "text/html");
17        mimeTypes.put("c", "text/plain");
18        mimeTypes.put("txt", "text/plain");
19        mimeTypes.put("java", "text/plain");
20        mimeTypes.put("mp4", "video/mp4");
21    }
22
23    public String get(String extension) {
24        if (mimeTypes.containsKey(extension))
25            return mimeTypes.get(extension);
26        else
27            return "application/octet-stream";
28    }
29
30 }
```

Figura 2

Archivo Manejador.java

Este archivo es el que da toda la funcionalidad al servidor, por que es el que controla las peticiones que se hagan desde un navegador y más específicamente, de la aplicación Web llamada Postman, ya que esta última hace peticiones HTTP del tipo HEAD, DELETE y PUT, dichas peticiones no las puede hacer un navegador común



```
1 import java.net.*;
2 import java.io.*;
3 import java.util.*;
4 import java.util.Base64;
5 import java.io.BufferedWriter;
6 import java.io.File;
7 import java.io.FileWriter;
8
9 public class Manejador extends Thread {
10
11     protected Socket c1;
12     protected DataOutputStream dos;
13     protected Mime mime;
14     protected DataInputStream dis;
15
16     public Manejador(Socket c1) throws Exception {
17         this.c1 = c1;
18         this.dos = new DataOutputStream(this.c1.getOutputStream());
19         this.mime = new Mime();
20         this.dis = new DataInputStream(this.c1.getInputStream());
```

Figura 3

La clase tiene un atributo socket, donde se hará la conexión, tiene un flujo de salida, un MIME que es el tipo de respuesta y, por último, contiene un flujo de entrada, el constructor sólo crea los atributos de la clase. Se ve en la Figura 4.

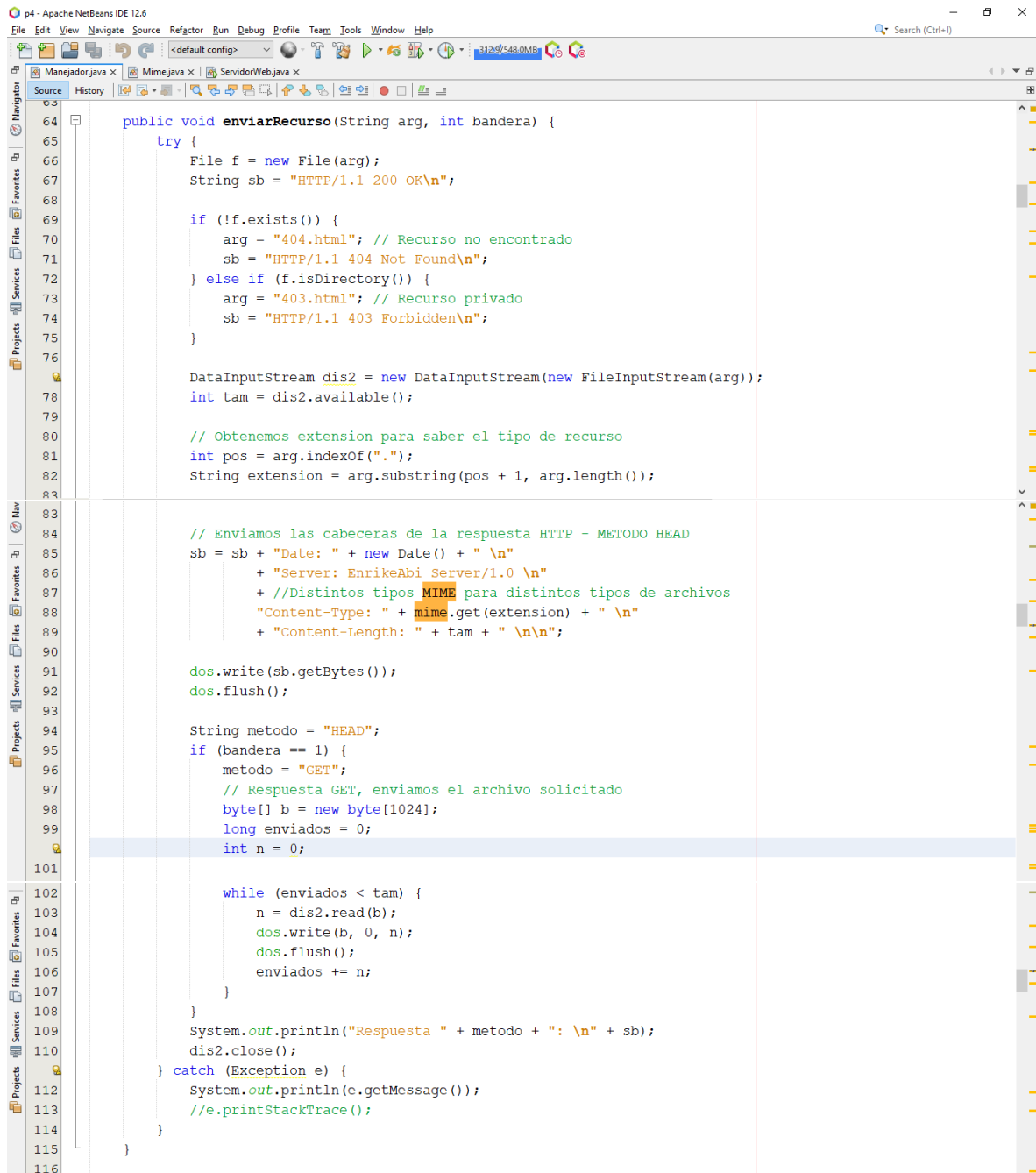
```

21 }
22
23 public void eliminarRecurso(String arg, String headers) {
24     try {
25         System.out.println(arg);
26         File f = new File(arg);
27
28         if (f.exists()) {
29             if (f.delete()) {
30                 System.out.println("-----> Archivo " + arg + " eliminado exitosamente\n");
31
32                 String deleteOK = headers
33                     + "<html><head><meta charset='UTF-8'><title>202 OK Recurso eliminado</title></head>"
34                     + "<body><h1>202 OK Recurso eliminado exitosamente.</h1>"
35                     + "<p>El recurso " + arg + " ha sido eliminado permanentemente del servidor."
36                     + "Ya no se podra acceder más a él.</p>"
37                     + "</body></html>";
38
39                 dos.write(deleteOK.getBytes());
40                 dos.flush();
41                 System.out.println("Respuesta DELETE: \n" + deleteOK);
42             } else {
43                 System.out.println("El archivo " + arg + " no pudo ser borrado\n");
44
45                 String error404 = "HTTP/1.1 404 Not Found\n"
46                     + "Date: " + new Date() + " \n"
47                     + "Server: EnrikeAbi Server/1.0 \n"
48                     + "Content-Type: text/html \n\n"
49                     + "<html><head><meta charset='UTF-8'><title>404 Not found</title></head>"
50                     + "<body><h1>404 Not found</h1>"
51                     + "<p>Archivo " + arg + " no encontrado.</p>"
52                     + "</body></html>";
53
54                 dos.write(error404.getBytes());
55                 dos.flush();
56                 System.out.println("Respuesta DELETE - ERROR 404: \n" + error404);
57             }
58         }
59     }
60 }

```

Figura 4

El método `eliminarRecurso` de la Figura 5, lo que hace es buscar un recurso en una cierta ruta especificada desde la URL y luego verifica si existe dentro del servidor, si no existe manda un código de error 404, pero si existe, lo elimina del servidor.



```
64 public void enviarRecurso(String arg, int bandera) {
65     try {
66         File f = new File(arg);
67         String sb = "HTTP/1.1 200 OK\n";
68
69         if (!f.exists()) {
70             arg = "404.html"; // Recurso no encontrado
71             sb = "HTTP/1.1 404 Not Found\n";
72         } else if (f.isDirectory()) {
73             arg = "403.html"; // Recurso privado
74             sb = "HTTP/1.1 403 Forbidden\n";
75         }
76
77         DataInputStream dis2 = new DataInputStream(new FileInputStream(arg));
78         int tam = dis2.available();
79
80         // Obtenemos extension para saber el tipo de recurso
81         int pos = arg.indexOf(".");
82         String extension = arg.substring(pos + 1, arg.length());
83
84         // Enviamos las cabeceras de la respuesta HTTP - METODO HEAD
85         sb = sb + "Date: " + new Date() + " \n"
86             + "Server: EnriqueAbi Server/1.0 \n"
87             + //Distintos tipos MIME para distintos tipos de archivos
88             "Content-Type: " + mime.get(extension) + " \n"
89             + "Content-Length: " + tam + " \n\n";
90
91         dos.write(sb.getBytes());
92         dos.flush();
93
94         String metodo = "HEAD";
95         if (bandera == 1) {
96             metodo = "GET";
97             // Respuesta GET, enviamos el archivo solicitado
98             byte[] b = new byte[1024];
99             long enviados = 0;
100             int n = 0;
101
102             while (enviados < tam) {
103                 n = dis2.read(b);
104                 dos.write(b, 0, n);
105                 dos.flush();
106                 enviados += n;
107             }
108         }
109         System.out.println("Respuesta " + metodo + ": \n" + sb);
110         dis2.close();
111     } catch (Exception e) {
112         System.out.println(e.getMessage());
113         //e.printStackTrace();
114     }
115 }
116 }
```

Figura 5

En la figura 6 se ve el uso del método `enviaRecurso` el cual es de ayuda para los métodos HTTP Get, Post y Head, lo que se hace es buscar un archivo y luego mandarlo a través de un flujo de salida, también retorna un encabezado con los códigos de error 404 o 202 si la acción se ejecutó correctamente.

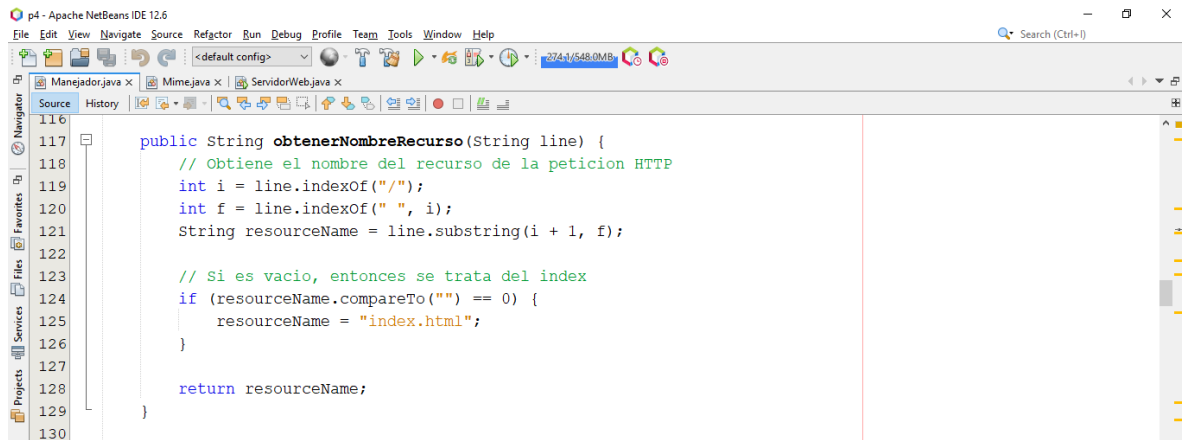
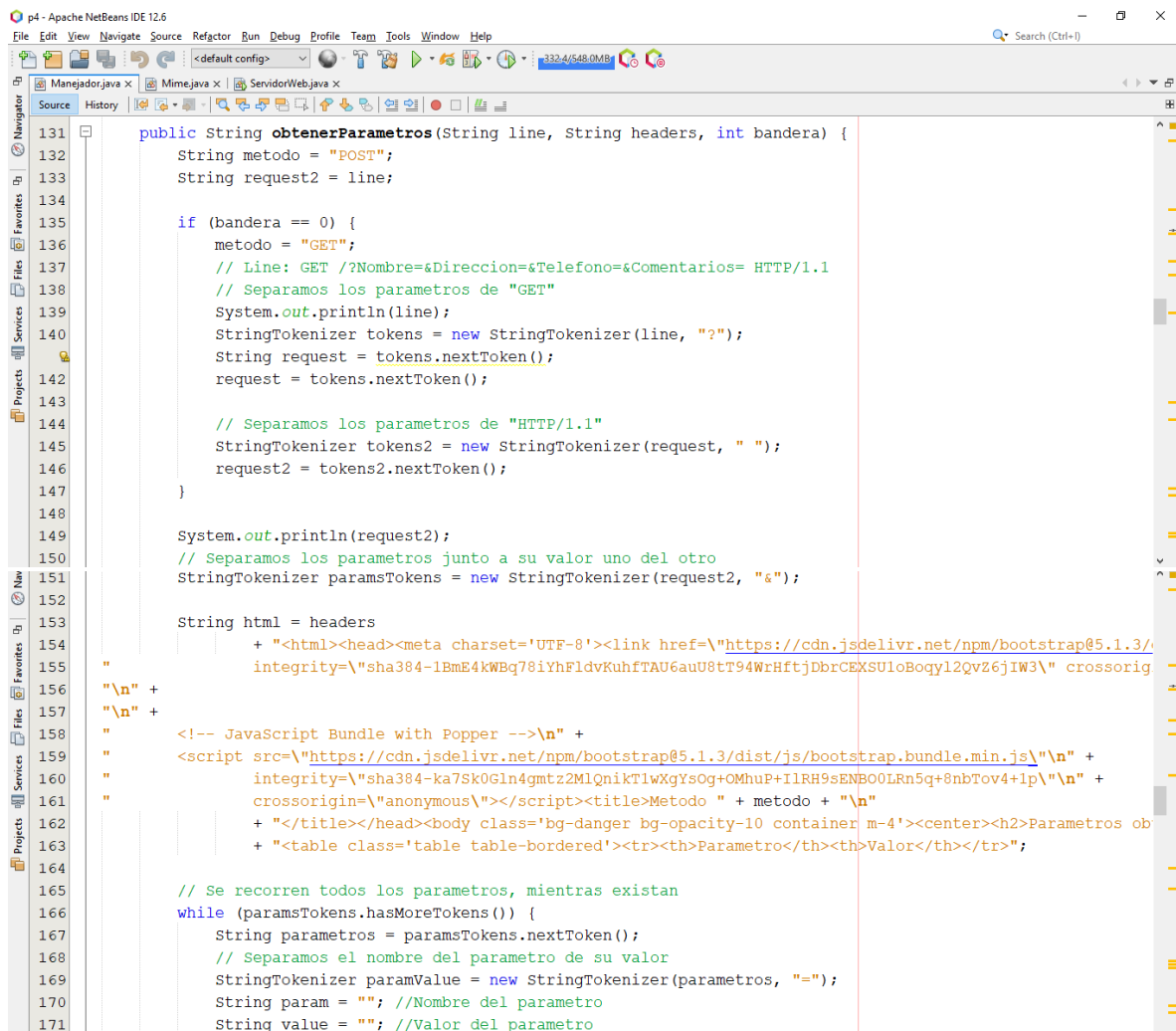
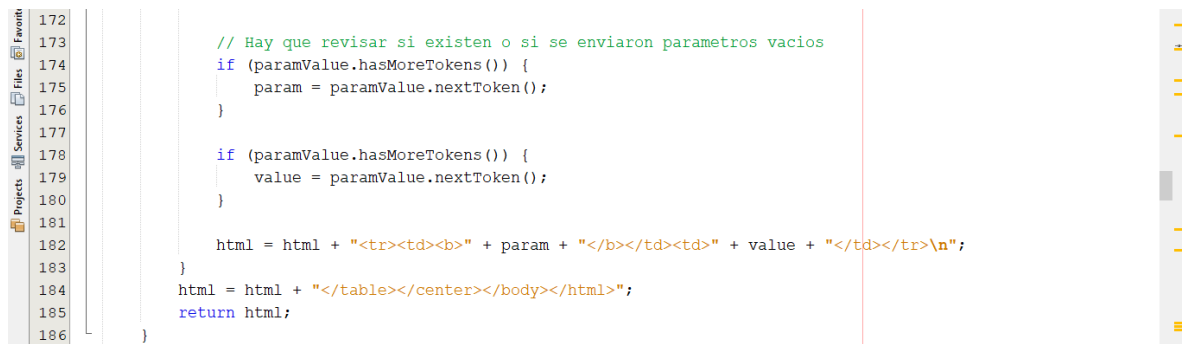


Figura 6

El método de la Figura 7 obtiene el nombre del recurso solicitado a través de la URI



A screenshot of an IDE window showing a Java file. The left sidebar has tabs for 'Projects', 'Services', 'Files', and 'Favorites'. The main editor area displays code from line 172 to 186. The code is a method that processes parameters and builds an HTML string. It uses `paramValue.hasMoreTokens()` to check for more parameters and `paramValue.nextToken()` to retrieve them. The HTML string is built using string concatenation with escape sequences for HTML tags and newlines. The code is as follows:

```
172
173 // Hay que revisar si existen o si se enviaron parametros vacios
174 if (paramValue.hasMoreTokens()) {
175     param = paramValue.nextToken();
176 }
177
178 if (paramValue.hasMoreTokens()) {
179     value = paramValue.nextToken();
180 }
181
182 html = html + "<tr><td><b>" + param + "</b></td><td>" + value + "</td></tr>\n";
183 }
184 html = html + "</table></center></body></html>";
185 return html;
186 }
```

Figura 7

El método `obtenerParametros` recupera el encabezado de respuesta para que se puedan ejecutar ciertos métodos, se ve de mejor forma en la Figura 8.

p4 - Apache NetBeans IDE 12.6

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Manejador.java Mime.java ServidorWeb.java

```
188 @Override
189 public void run() {
190     // Cabeceras de respuestas HTTP
191     String headers = "HTTP/1.1 200 OK\n"
192         + "Date: " + new Date() + " \n"
193         + "Server: Server/1.0 \n"
194         + "Content-Type: text/html \n\n";
195     try {
196         String line = dis.readLine(); // Lee primera linea DEPRECIADO !!!!
197         // Linea vacia
198         if (line == null) {
199             String vacia = "<html><head><title>Servidor WEB</title><body bgcolor='#AACCFF'>Linea Vacia</bod";
200             dos.write(vacia.getBytes());
201             dos.flush();
202         } else {
203             System.out.println("\n-----> Cliente Conectado desde: " + cl.getInetAddress());
204             System.out.println("Por el puerto: " + cl.getPort());
205             System.out.println("Datos: " + line + "\r\n\r\n");
206
207             // Metodo GET
208             if (line.toUpperCase().startsWith("GET")) {
209                 if (line.indexOf("?") == -1) {
210                     // Solicita un archivo
211                     String fileName = obtenerNombreRecurso(line);
212                     // Bandera HEAD = 0, GET = 1
213                     enviarRecurso(fileName, 1);
214                 } else {
215                     // Envia parametros desde un formulario
216                     // Bandera GET = 0, POST = 1
217                     String respuesta = obtenerParametros(line, headers, 0);
218                     // Respuesta GET, devolvemos un HTML con los parametros del formulario
219                     dos.write(respuesta.getBytes());
220                     dos.flush();
221                     System.out.println("Respuesta GET: \n" + respuesta);
222                 }
223             } // Metodo HEAD
224             else if (line.toUpperCase().startsWith("HEAD")) {
225                 if (line.indexOf("?") == -1) {
226                     // Solicita archivo, unicamente enviamos tipo mime y longitud
227                     String fileName = obtenerNombreRecurso(line);
228                     // Bandera HEAD = 0, GET = 1
229                     enviarRecurso(fileName, 0);
230                 } else {
231                     // Respuesta HEAD, devolvemos unicamente las cabeceras HTTP
232                     dos.write(headers.getBytes());
233                     dos.flush();
234                     System.out.println("Respuesta HEAD: \n" + headers);
235                 }
236             } // Metodo POST
237             else if (line.toUpperCase().startsWith("POST")) {
238                 // Leemos el flujo de entrada
239                 int tam = dis.available();
240                 byte[] b = new byte[tam];
241
242                 dis.read(b);
243                 //Creamos un string con los bytes leidos
244                 String request = new String(b, 0, tam);
245
246                 // Separamos los parametros del resto de los encabezados HTTP
247                 String[] reqLineas = request.split("\n");
248                 //Ultima linea del request
249                 int ult = reqLineas.length - 1;
```

```

250
251 // Bandera GET = 0, POST = 1
252 String respuesta = obtenerParametros(reqLineas[ult], headers, 1);
253
254 // Respuesta POST, devolvemos un HTML con los parametros del formulario
255 dos.write(respuesta.getBytes());
256 dos.flush();
257 System.out.println("Respuesta POST: \n" + respuesta);
258 } // Metodo DELETE
259 else if (line.toUpperCase().startsWith("DELETE")) {
260     String fileName = obtenerNombreRecurso(line);
261     eliminarRecurso(fileName, headers);
262 } else if (line.toUpperCase().startsWith("PUT")) {
263     String fileName = obtenerNombreRecurso(line);
264     String ruta = fileName;
265     try {
266         String contenido = "Contenido de ejemplo";
267         File file = new File(ruta);
268         // Si el archivo no existe es creado
269         System.out.println("Respuesta PUT: \n" + headers);
270         if (!file.exists()) {
271             file.createNewFile();
272         } else {
273             headers = "HTTP/1.1 204 OK\n"
274 + "Date: " + new Date() + " \n"
275 + "Server: Wicho Server/1.0 \n"
276 + "Content-Type: text/html \n\n";
277         }
278         FileWriter fw = new FileWriter(file);
279         BufferedWriter bw = new BufferedWriter(fw);
280         bw.write(contenido);
281         bw.close();
282     } catch (Exception e) {
283         e.printStackTrace();
284     }
285     dos.write(headers.getBytes());
286     dos.flush();
287     System.out.println("Respuesta PUT: \n" + headers);
288 }
289
290
291 else {
292     //Metodos no implementados en el servidor
293     String error501 = "HTTP/1.1 501 Not Implemented\n" +
294         "Date: " + new Date() + " \n" +
295         "Server: EnrikeAbi Server/1.0 \n" +
296         "Content-Type: text/html \n\n" +
297         "<html><head><meta charset='UTF-8'><title>Err
298         <body><h1>Error 501: No implementado.</h1>
299         <p>El método HTTP o funcionalidad solicitada
300         </body></html>";
301
302     dos.write(error501.getBytes());
303     dos.flush();
304     System.out.println("Respuesta ERROR 501: \n" + error501);
305 }
306
307 }
308 dis.close();
309 dos.close();
310 cl.close();
311 }
312 catch (Exception e
313
314
315 ) {
316     e.printStackTrace();
317 }
318
319
320

```

Figura 8

El método run() de la Figura 9, es el Hilo de ejecución que está esperando solicitudes de un cliente, en pocas palabras, se puede describir como un menú que analiza qué método HTTP ejecutar, luego manda a llamar a los métodos correspondientes para que se pueda implementar el método elegido, todo esto a través de la ayuda de un ENCABEZADO, una URL y un MIME.

The screenshot shows a web application with a light green background and a dark green header bar containing the title "Peticiones". Below the header, there are three distinct form sections:

- Formulario de Registro GET:** A form with a teal border containing input fields for "Nombre:", "Dirección:", "Teléfono:", and "Comentarios:". Below these fields are two buttons: "Enviar datos" and "Borrar datos".
- Inicio de Sesión:** A form with the sub-header "Prueba POST" containing input fields for "Usuario:" and "Contraseña:", followed by a "Login" button.
- Prueba HEAD, solicitar tamaño:** A form containing a vertical list of buttons: "Tamaño imagen", "Tamaño GIF", "Tamaño PDF", "Tamaño Video", an empty input field, and "Tamaño Otro".

Ilustración 1 Html ocupado para las pruebas

Conclusiones

Esta práctica fue un reto grande, debido a que se tuvo que leer y comprender mucho para poder implementar los métodos HTTP correctamente, pero de igual forma nos llevamos un gran aprendizaje ya que comprendimos de una manera más precisa cómo es que funcionan los métodos POST y GET, ambos muy importantes para el uso de las aplicaciones en red.

También conocimos cómo es que trabajan los MIMES, los encabezados de respuesta y los distintos códigos de error.