

Tarea PSP05



Yanet López Rodríguez
13/02/2024

Indice

• Portada	pg. 1
• Indice	pg. 2
• Introducción	pg. 3
• Ejercicio 1	pg. 4 - 5
◦ Manual de usuario	pg. 4
◦ Pruebas	pg. 5
• Ejercicio 2	pg. 6 - 10
◦ Manual de usuario	pg. 6 - 9
◦ Pruebas	pg. 10
• Conclusiones	pg. 11
• Recursos	pg. 11
• Bibliografía	pg. 11



Introducción

La tarea está dividida en dos ejercicios.



Ejercicio 1.

Modifica el ejemplo del servidor HTTP (Proyecto java ServerHTTP, apartado 5.1 de los contenidos) para que incluya la cabecera Date.



Ejercicio 2.

Modifica el ejemplo del servidor HTTP (Proyecto java ServerHTTP, apartado 5.1 de los contenidos) para que implemente multihilo, y pueda gestionar la concurrencia de manera eficiente.



Ejercicio 1.

Modifica el ejemplo del servidor HTTP (Proyecto java ServerHTTP, apartado 5.1 de los contenidos) para que incluya la cabecera Date.

Para incluir la cabecera “Date” en nuestras páginas HTML, primero modificaremos la clase Paginas, en esta crearemos un nuevo método que retorne un String con la fecha actual.

```
//CREAMOS UNA NUEVA FUNCIÓN QUE RETORNE UN String CON LA FECHA ACTUAL
private static String fechaHora(){
    return ZonedDateTime.now().format(DateTimeFormatter.RFC_1123_DATE_TIME);
}
```

Ahora creamos un nuevo String llamado “Date” donde almacenaremos la fecha y hora actual.

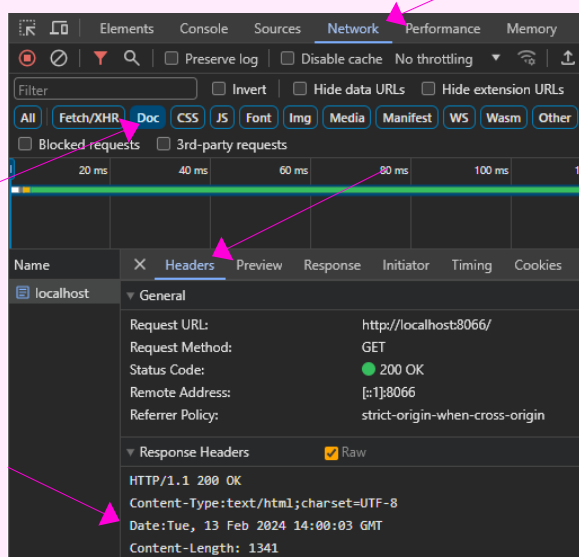
```
public class Paginas {

    public static final String primeraCabecera ="Content-Type:text/html;charset=UTF-8";

    //ALMACENAMOS EN EL String "Date" la fecha y hora actual
    public static final String Date = fechaHora();
```

Y por último, al mostrar la página HTML en el servidor (utilizando la clase PrintWriter) le añadimos una nueva línea de encabezado que mostrará la fecha actual. Debemos añadir la nueva línea en los tres if. En total imprimirá cuatro líneas de encabezado HTTP, "lineaInicial_OK", "Date", "primeraCabecera" y “Content-Length”.

```
//SI LA URL SOLICITADA ESTÁ VACÍA O IGUAL A "/"
if (peticion.length() == 0 || peticion.equals("/")) {
    html = Paginas.html_index;
    printWriter.println(Mensajes.lineaInicial_OK);
    printWriter.println(Paginas.primeraCabecera);
    printWriter.println("Date:" + Paginas.Date);
    printWriter.println("Content-Length: " + html.length() + 1);
    printWriter.println("\n");
    printWriter.println(html);
}
```



Para visualizar los encabezados de nuestras páginas HTML debemos abrir nuestro buscador, en mi caso usaré Chrome, pulsar F12 → Network → Doc → Headers y activar el CheckBox de Raw. Aquí veremos las cabeceras "lineaInicial_OK", "Date", "primeraCabecera" y “Content-Length”.

Pruebas

Añadimos la línea de encabezado "Date".

```
//SI LA URL SOLICITADA ESTÁ VACÍA O IGUAL A "/"
if (peticion.length() == 0 || peticion.equals("/")) {
    html = Paginas.html_index;
    printWriter.println(Mensajes.lineaInicial_OK);
    printWriter.println(Paginas.primerCabecera);
    printWriter.println("Date:" + Paginas.Date);
    printWriter.println("Content-Length: " + html.length() + 1);
    printWriter.println("\n");
    printWriter.println(html);
}
```

▼ Response Headers <input type="checkbox"/> Raw	
Content-Length:	1341
Content-Type:	text/html; charset=UTF-8
Date:	Tue, 13 Feb 2024 14:00:03 GMT

▼ Response Headers <input type="checkbox"/> Raw	
Content-Length:	1341
Content-Type:	text/html; charset=UTF-8
Date:	Tue, 13 Feb 2024 14:08:43 GMT

Comentamos la línea de encabezado "Date".

```
//SI LA URL SOLICITADA ESTÁ VACÍA O IGUAL A "/"
if (peticion.length() == 0 || peticion.equals("/")) {
    html = Paginas.html_index;
    printWriter.println(Mensajes.lineaInicial_OK);
    printWriter.println(Paginas.primerCabecera);
    //printWriter.println("Date:" + Paginas.Date);
    printWriter.println("Content-Length: " + html.length() + 1);
    printWriter.println("\n");
    printWriter.println(html);
}
```

▼ Response Headers <input type="checkbox"/> Raw	
Content-Length:	1341
Content-Type:	text/html; charset=UTF-8

Ejercicio 2.

Modifica el ejemplo del servidor HTTP (Proyecto java ServerHTTP, apartado 5.1 de los contenidos) para que implemente multihilo, y pueda gestionar la concurrencia de manera eficiente.

Para que pueda gestionar la concurrencia de manera eficiente lo primero que haremos será modificar la clase ServidorHTTP, el método que muestra el mensaje de inicio lo dejaremos igual, sin embargo cambiaremos el main por completo.

En el main crearemos un try catch, dentro del try primero creamos un objeto ServerSocket en el puerto 8066 para escuchar las peticiones entrantes, declaramos una nueva variable Socket para almacenar el socket del cliente que realiza la petición, llamamos a la función imprimeDisponible() para imprimir por pantalla el mensaje de inicio.

Iniciamos un bucle infinito para esperar y procesar continuamente las peticiones entrantes, esperamos y aceptamos la conexión entrante del cliente, y almacenamos el Socket en la variable "cliente". Mostramos por pantalla que estamos atendiendo a el cliente, creamos una instancia de la clase Multihilo (la clase para manejar múltiples solicitudes de clientes al mismo tiempo) y le pasamos el Socket del cliente como argumento.

Iniciamos el hilo para atender al cliente y por último mostramos por pantalla que ya se ha atendido a el cliente. En el catch capturamos cualquier excepción que pueda ocurrir.

```
class ServidorHTTP {

    public static void main(String[] args) {

        try {
            //CREAMOS UN OBJETO ServerSocket EN EL PUERTO 8066 PARA ESCUCHAR LAS PETICIONES ENTRANTES
            ServerSocket socServidor = new ServerSocket(8066);

            //DECLARAMOS UNA NUEVA VARIABLE Socket PARA ALMACENAR EL SOCKET DEL CLIENTE QUE REALIZA LA PETICIÓN
            Socket cliente;

            //LLAMAMOS A LA FUNCIÓN imprimeDisponible() PARA IMPRIMIR EN LA CONSOLA EL MANSAGE DE INICIO HOMOSEXUAL
            imprimeDisponible();

            //INICIAMOS UN BUCLE INFINITO PARA ESPERAR Y PROCESAR CONTINUAMENTE LAS PETICIONES ENTRANTES
            while (true) {
                //ESPERAMOS Y ACEPTAMOS LA CONEXIÓN ENTRANTE DEL CLIENTE, Y ALMACENAMOS EL SOCKET EN LA VARIABLE "cliente"
                cliente = socServidor.accept();
                //MOSTRAMOS POR PANTALLA QUE ESTAMOS ATENDIENDO A EL CLIENTE
                System.out.println("\nAtendiendo al cliente");

                //CREAMOS UNA INSTANCIA DE LA CLASE MULTIHILLO (LA CLASE PARA MANEJAR MÚLTIPLES SOLICITUDES DE CLIENTES
                //AL MISMO TIEMPO) Y LE PASAMOS EL SOCKET DEL CLIENTE COMO ARGUMENTO
                MultiHilo hilo = new MultiHilo(cliente);
                //INICIAMOS EL HILO PARA ATENDER AL CLIENTE.
                hilo.start();

                //MOSTRAMOS POR PANTALLA QUE YA SE HA ATENDIDO A EL CLIENTE
                System.out.println("Cliente atendido");
            }
            //CAPTURAMOS CUALQUIER EXCEPCIÓN
        } catch (IOException ex) {
            Logger.getLogger(ServidorHTTP.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    // MOSTRAMOS EL MANSAGE DE INICIO HOMOSEXUAL
    private static void imprimeDisponible() {
        System.out.println("El Servidor WEB se está ejecutando y permanece a la "
            + "escucha por el puerto 8066.\nEscribe en la barra de direcciones "
            + "de tu explorador preferido:\n\nhttp://localhost:8066\npara "
            + "solicitar la página de bienvenida\n\nhttp://localhost:8066/"
            + "quijote\n para solicitar una página del Quijote,\n\nhttp://"
            + "localhost:8066/q\n para simular un error");
    }
}
```


Ahora crearemos una nueva clase llamada MultiHilo, esta clase la usaremos para atender las solicitudes de los clientes de forma concurrente en el servidor HTTP. Cada instancia de esta clase se encarga de manejar una conexión individual y procesar las solicitudes de ese cliente en particular.

En esta clase primero declaramos un nuevo Socket y lo marcamos como final, creamos el constructor de la clase MultiHilo y creamos el método run() el cual se ejecutará cuando se inicie el subproceso de ejecución de MultiHilo.

Declaramos una nueva variable de tipo InputStreamReader y la inicializamos, declaramos dos nuevas variables de tipo String, creamos un objeto InputStreamReader a partir del flujo de entrada del socketCliente para leer la petición del cliente, creamos un objeto BufferedReader a partir del InputStreamReader para poder leer líneas de texto de la petición.

Creamos un objeto PrintWriter a partir del flujo de salida del socketCliente para escribir la respuesta que se enviará al cliente, leemos la primera línea de la petición del cliente y la asignamos a la variable y eliminamos todos los espacios en blanco de la variable "peticion" para facilitar su análisis.

```
public class MultiHilo extends Thread{

    //DECLARAMOS UN NUEVO Socket Y LO MARCAMOS COMO FINAL
    private final Socket socketCliente;

    //CREAMOS EL CONSTRUCTOR DE LA CLASE MULTIHILLO
    public MultiHilo(Socket socketCliente) {
        this.socketCliente = socketCliente;
    }

    //CREAMOS EL MÉTODO run() EL CUAL SE EJECUTARÁ CUANDO SE INICIE EL SUBPROCESO DE EJECUCIÓN DE MultiHilo
    @Override
    public void run() {
        //DECLARAMOS UNA NUEVA VARIABLE DE TIPO InputStreamReader Y LA INICIALIZAMOS
        InputStreamReader InputStream = null;
        try {
            //DECLARAMOS DOS NUEVAS VARIABLES DE TIPO String
            String peticion;
            String html;
            //CREAMOS UN OBJETO InputStreamReader A PARTIR DEL FLUJO DE ENTRADA DEL socketCliente PARA LEER LA PETICIÓN DEL CLIENTE
            InputStream = new InputStreamReader(socketCliente.getInputStream());
            //CREAMOS UN OBJETO BufferedReader A PARTIR DEL InputStreamReader PARA PODER LEER LÍNEAS DE TEXTO DE LA PETICIÓN
            BufferedReader bufLeer = new BufferedReader(InputStream);
            //CREAMOS UN OBJETO PrintWriter A PARTIR DEL FLUJO DE SALIDA DEL socketCliente PARA ESCRIBIR LA RESPUESTA QUE SE
            //ENVIARÁ AL CLIENTE
            PrintWriter printWriter = new PrintWriter(socketCliente.getOutputStream(), true);
            //LEEMOS LA PRIMERA LÍNEA DE LA PETICIÓN DEL CLIENTE Y LA ASIGNAMOS A LA VARIABLE "peticion"
            peticion = bufLeer.readLine();
            //ELIMINAMOS TODOS LOS ESPACIOS EN BLANCO DE LA VARIABLE "peticion" PARA FACILITAR SU ANÁLISIS
            peticion = peticion.replaceAll(" ", "");
        }
    }
}
```

Si la petición del cliente comienza con "GET", extraemos la subcadena de "peticion" (entre 'GET' y 'HTTP/1.1') que contiene la url solicitada por el cliente.

```
//SI LA PETICIÓN DEL CLIENTE COMIENZA CON "GET"
if (peticion.startsWith("GET")) {
    //EXTRAEMOS LA SUBCADENA DE "peticion" (ENTRE 'GET' Y 'HTTP/1.1')
    //QUE CONTIENE LA URL SOLICITADA POR EL CLIENTE
    peticion = peticion.substring(3, peticion.lastIndexOf("HTTP"));
}
```

Según si la URL solicitada, esta vacía, es igual a "/" o a "/quijote" o ninguna de estas opciones, asignamos el contenido de la página correspondiente a la variable "html" y escribimos en el flujo de salida:

- La línea inicial.
- La primera cabecera.
- La fecha actual.
- La longitud del contenido.
- Una línea en blanco para indicar el fin de las cabeceras.
- El contenido de la página solicitada.

```
//SI LA URL SOLICITADA ESTÁ VACÍA O IGUAL A "/"
if (peticion.length() == 0 || peticion.equals("/")) {
    //ASIGNAMOS EL CONTENIDO DE LA PÁGINA "html_index" A LA VARIABLE "html"
    html = Paginas.html_index;
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA LÍNEA INICIAL DE UNA RESPUESTA EXITOSA
    printWriter.println(Mensajes.lineaInicial_OK);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA PRIMERA CABECERA
    printWriter.println(Paginas.primerCabecera);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA FECHA ACTUAL
    printWriter.println("Date:" + Paginas.Date);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA LONGITUD DEL CONTENIDO
    printWriter.println("Content-Length: " + html.length() + 1);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA UNA LÍNEA EN BLANCO PARA INDICAR EL FIN DE LAS CABECERAS
    printWriter.println("\n");
    //ESCRIBIMOS EN EL FLUJO DE SALIDA EL CONTENIDO DE LA PÁGINA SOLICITADA
    printWriter.println(html);
}

//SI LA URL SOLICITADA ES IGUAL A "/quijote"
else if (peticion.equals("/quijote")) {
    //ASIGNAMOS EL CONTENIDO DE LA PÁGINA "html_quijote" A LA VARIABLE "html"
    html = Paginas.html_quijote;
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA LÍNEA INICIAL DE UNA RESPUESTA EXITOSA
    printWriter.println(Mensajes.lineaInicial_OK);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA PRIMERA CABECERA
    printWriter.println(Paginas.primerCabecera);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA FECHA ACTUAL
    printWriter.println("Date:" + Paginas.Date);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA LONGITUD DEL CONTENIDO
    printWriter.println("Content-Length: " + html.length() + 1);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA UNA LÍNEA EN BLANCO PARA INDICAR EL FIN DE LAS CABECERAS
    printWriter.println("\n");
    //ESCRIBIMOS EN EL FLUJO DE SALIDA EL CONTENIDO DE LA PÁGINA SOLICITADA
    printWriter.println(html);
}

//SI LA URL SOLICITADA NO ESTÁ VACÍA NI ES IGUAL A "/" NI A "/quijote"
else {
    //ASIGNAMOS EL CONTENIDO DE LA PÁGINA "html_noEncontrado" A LA VARIABLE "html"
    html = Paginas.html_noEncontrado;
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA LÍNEA INICIAL DE NotFound
    printWriter.println(Mensajes.lineaInicial_NotFound);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA PRIMERA CABECERA
    printWriter.println(Paginas.primerCabecera);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA FECHA ACTUAL
    printWriter.println("Date:" + Paginas.Date);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA LA LONGITUD DEL CONTENIDO
    printWriter.println("Content-Length: " + html.length() + 1);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA UNA LÍNEA EN BLANCO PARA INDICAR EL FIN DE LAS CABECERAS
    printWriter.println("\n");
    //ESCRIBIMOS EN EL FLUJO DE SALIDA EL CONTENIDO DE LA PÁGINA SOLICITADA
    printWriter.println(html);
}
```


Por último cerramos el catch, el cual capturara cualquier excepción y añadimos un finally, este se ejecutará siempre, aun que se produzca alguna excepción. En el finally crearemos otro try catch para cerrar el InputStreamReader liberando los recursos. Si se produce algun excepción el catch la capturará.

```
//CAPTURAMOS CUALQUIER EXCEPCIÓN DE TIPO IOException QUE PUEDA OCURRIR
} catch (IOException ex) {
    Logger.getLogger(ServidorHTTP.class.getName()).log(Level.SEVERE, null, ex);
//INDEPENDIENTEMENTE DE SI SE PRODUCE UNA EXCEPCIÓN O NO
} finally {
    try {
        //CERRAMOS EL InputStreamReader PARA LIBERAR RECURSOS
        InputStream.close();
        //CAPTURAMOS CUALQUIER EXCEPCIÓN DE TIPO IOException QUE PUEDA OCURRIR
    } catch (IOException ex) {
        Logger.getLogger(ServidorHTTP.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Para comprobar que el servidor funciona bien trabajando con muchos hilos a la vez cree la clase Prueba, en esta clase hacemos llamadas simultáneas al servidor con diferentes hilos.

```
public class Prueba {

    public static void main(String[] args) {
        //CREAMOS UN ExecutorService CON UN NÚMERO DETERMINADO DE HILOS
        ExecutorService executor = Executors.newFixedThreadPool(10);

        //REALIZAMOS LLAMADAS SIMULTÁNEAS AL SERVIDOR CON DIFERENTES HILOS
        for (int i = 0; i < 10; i++) {
            Runnable worker = new LlamadaHTTP();
            executor.execute(worker);
        }

        //FINALIZAMOS EL ExecutorService
        executor.shutdown();
    }

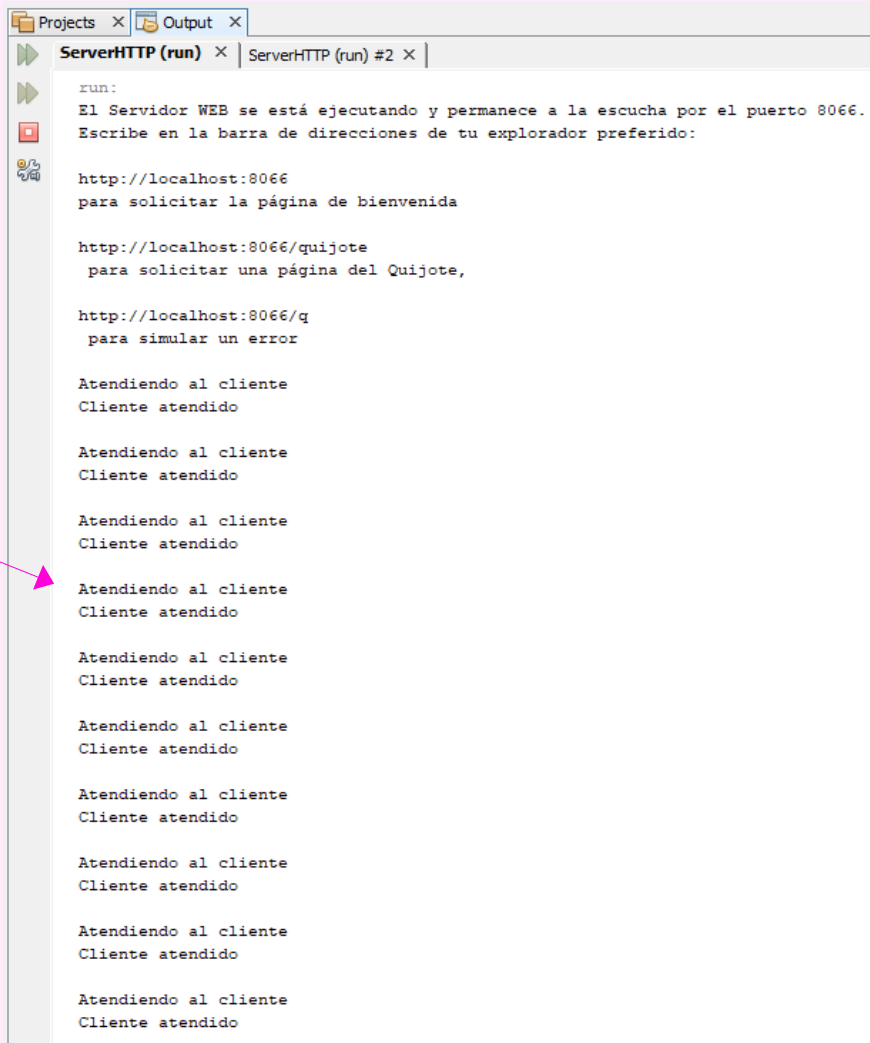
    public static class LlamadaHTTP implements Runnable {
        @Override
        public void run() {
            try {
                //REALIZAMOS LA LLAMADA HTTP AL SERVIDOR
                URL url = new URL("http://localhost:8066/");
                HttpURLConnection connection = (HttpURLConnection) url.openConnection();
                connection.setRequestMethod("GET");

                //OBTENEMOS EL CODIGO DE RESPUESTA
                int responseCode = connection.getResponseCode();
                System.out.println("Código de respuesta: " + responseCode);

                //CERRAMOS LA CONEXIÓN
                connection.disconnect();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Pruebas

Iniciamos el servidor y ejecutamos la prueba.



```
run:
El Servidor WEB se está ejecutando y permanece a la escucha por el puerto 8066.
Escribe en la barra de direcciones de tu explorador preferido:

http://localhost:8066
para solicitar la página de bienvenida

http://localhost:8066/quijote
para solicitar una página del Quijote,

http://localhost:8066/q
para simular un error

Atendiendo al cliente
Cliente atendido

Atendiendo al cliente
Cliente atendido

Atendiendo al cliente
Cliente atendido

Atendiendo al cliente
Cliente atendido

Atendiendo al cliente
Cliente atendido

Atendiendo al cliente
Cliente atendido

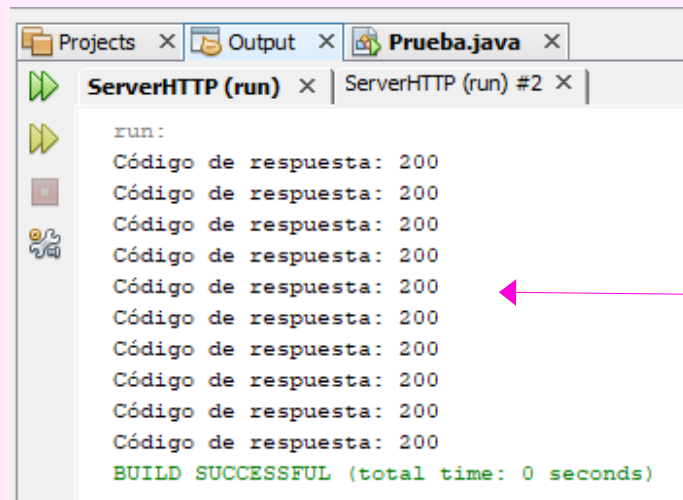
Atendiendo al cliente
Cliente atendido

Atendiendo al cliente
Cliente atendido

Atendiendo al cliente
Cliente atendido

Atendiendo al cliente
Cliente atendido
```

El código "200" es un código de respuesta HTTP que indica que la solicitud fue exitosa



```
run:
Código de respuesta: 200
Código de respuesta: 200
Código de respuesta: 200
Código de respuesta: 200
Código de respuesta: 200
Código de respuesta: 200
Código de respuesta: 200
Código de respuesta: 200
Código de respuesta: 200
Código de respuesta: 200
Código de respuesta: 200
BUILD SUCCESSFUL (total time: 0 seconds)
```



Conclusiones.



En el **ejercicio 1**, mi conclusión ha sido que al incluir la cabecera Date, se asegura una respuesta más precisa y actualizada para cada solicitud realizada al servidor. Esto mejora la experiencia del cliente al recibir información en tiempo real. La inclusión de la cabecera Date en el servidor permite que los clientes conozcan la fecha exacta de la respuesta del servidor y esto puede ser útil para realizar análisis y seguimiento de las solicitudes y respuestas.



En el **ejercicio 2**, mi conclusión ha sido que al implementar el servidor HTTP como multihilo, crea a una mejora significativa en el rendimiento del servidor, ya que se pueden gestionar múltiples solicitudes simultáneamente. Y que al estar implementado como multihilo, el servidor HTTP tiene una mayor tolerancia a fallos. Si un hilo se bloquea o encuentra un error, los demás hilos pueden continuar ejecutándose y seguir atendiendo solicitudes de otros clientes.



Recursos.

Recursos necesarios para realizar la Tarea:

- 💡 IDE NetBeans.
- 💡 Contenidos de la unidad.
- 💡 Ejemplos expuestos en el contenido de la unidad.



Bibliografía.



<https://www.biblia.es/biblia-online.php>



<https://www.churchofjesuschrist.org/study/scriptures?lang=spa>



<https://pastoralsj.org/biblia>



<https://www.biblija.net/biblija.cgi?l=es>



<https://www.bibliatodo.com/la-biblia>