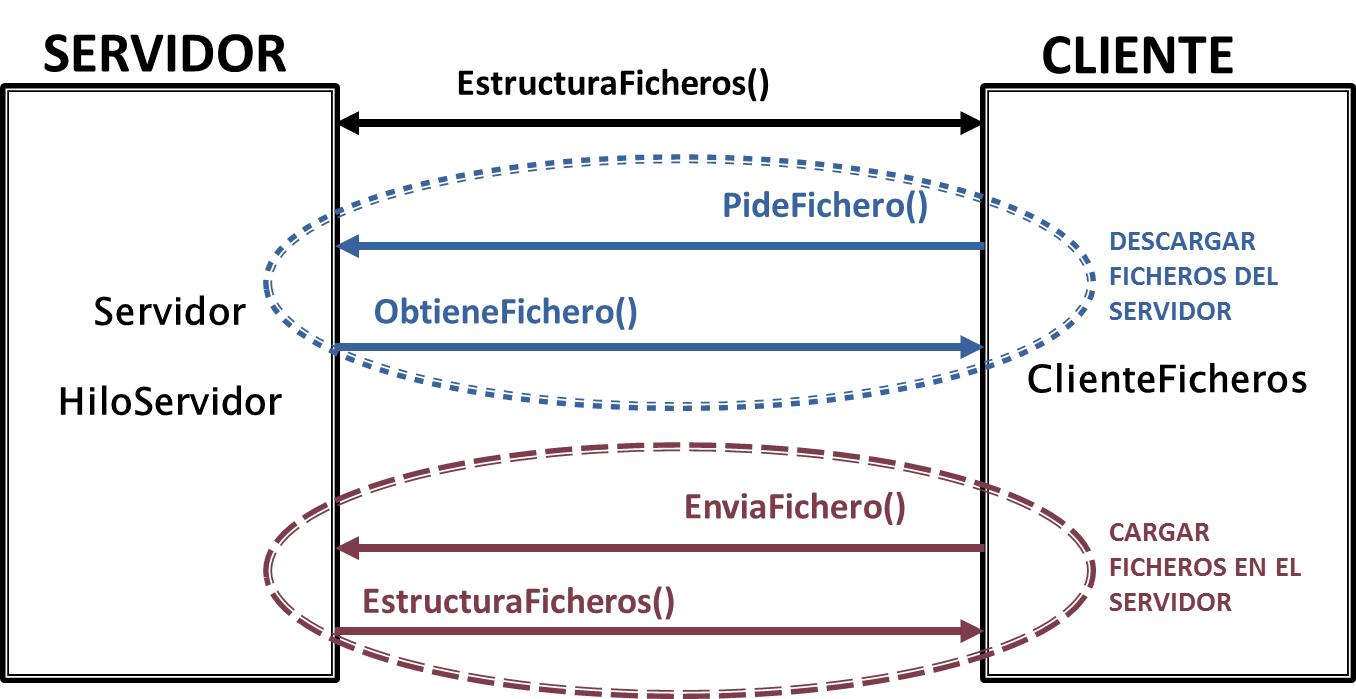
**ACTIVIDAD 5**

**Enunciado**

Completar el desarrollo de un programa en Java para gestionar un **servidor de ficheros**. El código de ciertas clases del programa está disponible en formato electrónico, mientras que el de otras será preciso codificar a partir del contenido de este documento. También se piden evidencias del correcto funcionamiento de ciertas funcionalidades del programa.

Cuando se inicia el servidor se elige la carpeta a la que van a acceder los clientes cuando se conecten al servidor. Los clientes se conectarán al servidor y podrán subir o descargar ficheros del mismo. La comunicación entre cliente y servidor se realizará mediante sockets y objetos. El funcionamiento general del servidor de ficheros se muestra en la siguiente figura:

**

El funcionamiento general del servidor de ficheros se describe a continuación:

* Desde el programa servidor se elige la carpeta o directorio a la que los clientes podrán acceder.
* Se inicia el servidor en un puerto pactado.
* Cuando un cliente se conecta se crea un objeto **EstructuraFicheros** con la información del directorio elegido.
* El servidor envía al cliente un objeto **EstructuraFicheros** nada más conectarse.
* Cuando el cliente solicita descargar un fichero, realiza la petición mediante un objeto **PideFichero** con el nombre del fichero a descargar. El servidor recibe la petición y en respuesta le envía un objeto **ObtieneFichero,** con los bytes y el tamaño del fichero solicitado.
* Cuando el cliente solicita cargar un fichero en el servidor, la petición se realiza mediante un objeto **EnviaFichero** que contiene el nombre del fichero, su contenido en bytes y su tamaño. El servidor acepta la petición, crea el nuevo fichero en su directorio y en respuesta le envía un objeto **EstructuraFicheros** con la estructura de ficheros actualizada para el directorio en el que se hizo la carga.

En la siguiente tabla se muestran numeradas las clases Java que conforman el programa, indicando el formato en el que van a estar disponibles así como el número aproximado de líneas de código:

|  |  |  |  |
| --- | --- | --- | --- |
| **Número de clase** | **Nombre de la clase** | **Formato disponible** | **Número aproximado de líneas de código** |
| 1 | EstructuraFicheros | Electrónico | 100 |
| 2 | Servidor | Papel | 50 |
| 3 | HiloServidor | Papel | 120 |
| 4 | PideFichero | Papel | 20 |
| 5 | ObtieneFichero | Papel | 20 |
| 6 | EnviaFichero | Papel | 35 |
| 7 | ClienteFicheros | Electrónico | 350 |

A continuación se incluye una breve descripción del funcionamiento de las diferentes clases Java así como el código asociado a las que no se proporcionan en formato electrónico:

1. Clase **EstucturaFicheros**

Se utiliza para definir la estructura del directorio seleccionado con sus ficheros y directorios. Tiene varios atributos:

* name: nombre del directorio
* path: nombre completo
* isDir: indica si es un directorio o no
* numeFich: número de ficheros y directorios del directorio seleccionado
* EstructuraFicheros[] lista: array de ficheros y directorios

La clase dispone de 2 constructores así como de un conjunto de métodos e tipo get para obtener los valores de los atributos. El primer constructor usa el método *getListaFiles()* para obtener los ficheros y directorios.

1. Clase **Servidor**

Define el puerto que va a quedar en escucha. Mediante **JFileChooser** se elige el directorio al que van acceder los clientes. Se crea un proceso repetitivo esperando a la petición de los clientes. Por cada cliente que se conecte se le enviará una **EstructuraFicheros**(Directorio).

Su código Java asociado es el siguiente:

**import** java.io.File;

**import** java.io.IOException;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** javax.swing.JFileChooser;

**public** **class** Servidor {

**static** Integer *PUERTO*=44441;

**static** **public** EstructuraFicheros *NF*;

**static** ServerSocket *servidor*;

**public** **static** **void** main (String [] args) **throws** IOException {

String Directorio = "";

JFileChooser f = **new** JFileChooser();

f.setFileSelectionMode(JFileChooser.***DIRECTORIES\_ONLY***);

f.setDialogTitle("Selecciona el directorio donde están los ficheros");

**int** returnVal = f.showDialog(f, "Seleccionar");

**if** (returnVal == JFileChooser.***APPROVE\_OPTION***) {

File file = f.getSelectedFile();

Directorio = file.getAbsolutePath();

}

// Si no se selecciona nada, salir

**if** (Directorio.equals("")) {

System.***out***.println("Debes seleccionar un directorio.");

System.*exit*(1);

}

*servidor* = **new** ServerSocket(*PUERTO*);

System.***out***.println("Servidor iniciado en el puerto "+*PUERTO*);

**while** (**true**) {

Socket cliente = *servidor*.accept();

System.***out***.println("Bienvenido al cliente");

*NF* = **new** EstructuraFicheros(Directorio);

HiloServidor hilo = **new** HiloServidor (cliente, *NF*);

hilo.start(); // Ejecución del hilo

}

}

}

1. Clase **HiloServidor**

Las operaciones de cada cliente conectado al servidor se gestionarán mediante la clase HiloServidor. Se definen los stream de entrada y de salida y se obtienen en el constructor, a partir del socket enviado por el servidor. El intercambio de objetos entre el cliente y el servidor se llevará a cabo mediante el método *run()* del hilo.

En el método *run()* lo primero que se hace es enviar al cliente el objeto EstructuraFicheros. Las peticiones llegan como objetos. Se comprobará, mediante el operador instanceof, si el objeto es una instancia de PideFichero o de ObtieneFichero. El proceso terminará cuando el cliente cierre la conexión, produciéndose una IOException.

El código Java asociado es el siguiente:

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.net.Socket;

**public** **class** HiloServidor **extends** Thread{

Socket socket;

ObjectOutputStream outObjeto; //stream de salida

ObjectInputStream inObjeto; //stream de entrada

EstructuraFicheros NF;

// Constructor

**public** HiloServidor (Socket s, EstructuraFicheros nF) **throws** IOException {

socket = s;

NF = nF;

inObjeto = **new** ObjectInputStream (socket.getInputStream());

outObjeto= **new** ObjectOutputStream (socket.getOutputStream());

}

**public** **void** run() {

**try** {

// Enviar al cliente el objeto EstructuraFicheros

outObjeto.writeObject(NF);

**while** (**true**) {

System.***out***.println("EEEEEEEnviado");

Object peticion;

**try** {

peticion = inObjeto.readObject();

// Comprobar qué quiere el cliente

**if** (peticion **instanceof** PideFichero) {

// El cliente pide un fichero al servidor

PideFichero fichero = (PideFichero) peticion;

EnviaFichero(fichero);

}

**if** (peticion **instanceof** EnviaFichero) {

// El cliente envía un fichero al servidor para cargarlo

EnviaFichero fic = (EnviaFichero) peticion;

File d=**new** File (fic.getDirectorio());

File f1=**new** File (d, fic.getNombre());

// Creación del fichero en el directorio,

// con los bytes enviados en el objeto

FileOutputStream fos = **new** FileOutputStream(f1);

fos.write(fic.getContenidoFichero());

fos.close();

// Creación de la nueva estructura de directorios

EstructuraFicheros n = **new** EstructuraFicheros (fic.getDirectorio());

outObjeto.writeObject(n);// Se envía al cliente

}

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

inObjeto.close();

outObjeto.close();

socket.close();

System.***out***.println("Cerrando cliente");

}

} // while

} **catch** (IOException e) {

// Cuando un cliente cierra la conexión

}

}// Fin run

// Método que envía al cliente el fichero solicitado

**private** **void** EnviaFichero (PideFichero fich) {

**try** {

// Obtención del fichero

File fichero = **new** File (fich.getNombreFichero());

FileInputStream filein = **null**;

filein= **new** FileInputStream(fichero);

**long** bytes=fichero.length();

**byte**[] buff = **new** **byte** [(**int**) bytes];

**int** i,j = 0;

// Lectura del fichero y llenado del array

**try** {

**while** ((i = filein.read()) != -1 ) { // Lectura de bytes

buff[j]=(**byte**) i;

j++;

}

} **catch** (IOException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

**try** {

filein.close();

} **catch** (IOException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

Object ff=**new** ObtieneFichero(buff);

// Envío del objeto ObtieneFichero con los bytes del fichero

**try** {

outObjeto.writeObject(ff);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}// EnviaFichero

}

1. Clase **PideFichero**

Se pone en funcionamiento cuando el cliente le pide al servidor el nombre de un fichero que el primero quiere descargar.

Define un atributo con el nombre del fichero a descargar y el método para obtenerlo, además del constructor.

El código Java asociado es el siguiente:

**import** java.io.Serializable;

**public** **class** PideFichero **implements** Serializable {

// Declaración atributo con el nombre del fichero a descargar del servidor

String nombreFichero;

// Constructor

**public** PideFichero (String nombreFichero) {

**this**.nombreFichero=nombreFichero;

}

// Método para obtener el nombre del fichero a descargar

**public** String getNombreFichero() { **return** nombreFichero;}

}

1. Clase **ObtieneFichero**

Se pone en funcionamiento después de que el servidor haya recibido del cliente un objeto de tipo PideFichero.

Se va leyendo el fichero que el cliente quiere descargar y se va guardando en un array de bytes que después se enviará al cliente.

Define un atributo con el contenido en bytes del fichero y el método para obtenerlo, además del constructor.

El código Java asociado es el siguiente:

**import** java.io.Serializable;

**public** **class** ObtieneFichero **implements** Serializable {

// Declaración atributo con el contenido en bytes del fichero a descargar

**byte**[] contenidoFichero ;

// Constructor

**public** ObtieneFichero(**byte**[] contenidoFichero) {

**this**.contenidoFichero = contenidoFichero;

}

// Método para obtener el fichero a descargar

**public** **byte**[] getContenidoFichero() {**return** contenidoFichero;}

}

1. Clase **EnviaFichero**

Se pone en funcionamiento cuando el cliente quiere subir al servidor un determinado fichero.

Define tres atributos con el contenido del fichero, el nombre del fichero y el directorio donde cargarlo. Dispone de varios métodos para obtener los atributos anteriormente descritos así como el constructor.

El código Java asociado es el siguiente:

**import** java.io.Serializable;

**public** **class** EnviaFichero **implements** Serializable{

// Declaración de atributos del fichero a subir al servidor

**byte**[] contenidoFichero;

String nombre;

String directorio;

// Constructor

**public** EnviaFichero(**byte**[] contenidoFichero, String nombre, String directorio) {

**this**.contenidoFichero = contenidoFichero;

**this**.nombre = nombre;

**this**.directorio = directorio;

}

// Método para obtener el contenido del fichero

**public** **byte**[] getContenidoFichero() {

**return** contenidoFichero;

}

// Método para obtener el nombre del fichero

**public** String getNombre() {

**return** nombre;

}

// Método para obtener el directorio del fichero

**public** String getDirectorio() {

**return** directorio;

}

}

1. Clase **ClienteFicheros**

Para implementar el cliente del servidor de ficheros se va a heredar de la clase JFrame y se va a implementar la interface Runnable. En el método main se creará un socket que se conectará al servidor por el puerto pactado.

El diseño de la pantalla incluye una lista, varias cajas de texto y los botones Subir, Descargar y Salir.

El constructor define los flujos de entrada y de salida.

Se creará un hilo por cada uno de los clientes conectados al servidor. El método *run()* va a dibujar la pantalla y a obtener la **EstructuraFicheros** con información sobre el número de ficheros junto a la lista de ficheros. Luego los visualizará en el JList mediante el método *llenarLista().*

En el elemento lista solo se van a implementar los ficheros.

Para **Subir** un fichero se seleccionará el fichero mediante la ventana JfileChooser, desde donde el cliente elige el fichero. Una vez elegido, se leen los bytes del mismo y se van almacenando en un array de bytes para mandárselo al servidor en un fichero EnviaFichero. Se envía al servidor y el servidor responde con un objeto EstructuraFicheros con el listado de ficheros actualizado. Se llama al método *llenarLista()* para que el cliente actualice los ficheros de la lista.

Para **Descargar** un fichero se hace una petición al servidor mediante el objeto PideFichero, donde se solicita el fichero seleccionado. Antes hay que comprobar si se ha hecho click sobre algún fichero del JList. Esto se hace preguntando sobre la variable *ficherocompleto*. A continuación se abre un FileOutputStream con el nombre del fichero seleccionado para copiar lo que se reciba del ObjectInputStream del socket. Se lee el objeto recibido del socket (ObtieneFichero) y se pasan los bytes al fichero abierto mediante FileOutputStream. El fichero se creará en el directorio de trabajo actual y su nombre será el mismo que aparece en el JList.

El cierre del cliente mediante la pulsación del botón **Cerrar** implica el cierre del socket más el lanzamiento de la excepción que se pueda producir.

**CONSIDERACIONES DE DISEÑO**

Este programa sólo permite cargar y descargar ficheros del directorio seleccionado en el servidor. Si el directorio seleccionado tiene a su vez otros directorios, éstos se mostrarán en pantalla, pero la navegación a través de ellos no se podrá realizar. Se mostrará un mensaje indicando que la función no está implementada: FUNCIÓN NO IMPLEMENTADA.

Tal y como está diseñado el programa, si hubiera varios clientes conectados de forma simultánea y uno de ellos subiera ficheros al servidor, éstos no serán visibles para el resto de los clientes conectados hasta que no reciban por parte del servidor un objeto **EstructuraFicheros** con la lista actual de los ficheros del directorio, y esto se produce cuando el cliente carga un fichero al servidor.

Para ejecutar el programa, las clases *EnviaFichero, EstructuraFicheros, ObtieneFichero* y *PideFichero* tienen que estar tanto en la máquina cliente como en el servidor. Además, en la máquina cliente se necesita la clase *ClienteFicheros* y en la máquina servidor las clases *Servidor* e *HiloServidor*.

El fichero se envía en un solo objeto; es decir, se llena todo el fichero, se mete en un array y se envía. Si el fichero a enviar es muy grande, pueden surgir errores de memoria del tipo *java.lang.OutOfMemoryError: Java heap space*. Para evitarlo se puede enviar el fichero por trozos; por ejemplo, enviándolo en partes de 4.000 bytes.

**EVIDENCIAS**

Incluye evidencias gráficas y **explica** el correcto funcionamiento del programa para las siguientes funcionalidades del servidor de ficheros:

1. Elegir mediante la interface gráfica del servidor la carpeta “C:\Usuarios\dm2\PROS” (u otra equivalente) como la carpeta a la que van a acceder los clientes cuando se conecten al servidor.
2. Introducir dentro de esta carpeta 7 archivos cuyos nombres no coincidan con los que se van a subir más adelante desde el cliente y crear dentro de ella una carpeta llamada Descargas
3. Subir a la carpeta a la que van a acceder los clientes, 2 ficheros de nombre FICHERO1.docx y FICHERO2.docx, los cuales están disponibles en la plataforma Moodle
4. Descargar de la carpeta del servidor a la que acceden los clientes, el fichero de nombre FICHERO2.docx.

Recomendaciones a la hora de probar el servidor de ficheros:

* Empezar ejecutando la clase Servidor, eligiendo la carpeta a la que van a acceder los clientes cuando se conecten a él.
* Ejecutar la clase ClienteFicheros, realizando las subidas y descargas de ficheros que se necesiten.
* Salir de la clase ClienteFicheros pulsando en el botón “Salir”.

**RESOLUCIÓN DE PROBLEMAS**

**REQUISITOS**

1. Entregar la actividad en un archivo comprimido que contenga todos los ficheros \*.java empleados para el funcionamiento del servidor de ficheros, más un documento nuevo con el apartado de EVIDENCIAS rellenado con lo que se pide.
2. El nombre del documento que contenga el apartado EVIDENCIAS rellenado será el mismo que el del enunciado, añadiendo al final del mismo el primer apellido de la persona que realiza la actividad. Por ejemplo: 4\_ACTIVIDADES 5\_V3 (Servidor ficheros)\_Perez
3. El archivo comprimido seguirá la siguiente nomenclatura:

*UD4\_Actividadn\_apellido*

donde:

*n*: valdrá 5, representando el número de la actividad

*apellido*: será el primer apellido del alumno

**CRITERIOS DE CORRECCIÓN**

1. Nivel de cumplimiento de los Requisitos (15%).
2. Nivel de adecuación del código Java de las clases no suministradas en formato electrónico, al detalle incluido en el Enunciado (15%).
3. Nivel de adecuación de las Evidencias aportadas con respecto a lo que se pide en cada una de ellas (70%).
4. Explicación detallada de cómo se han resuelto problemas que hayan podido surgir en la realización de la actividad (hasta un 15% adicional sobre la nota obtenida). Dicha explicación se incluirá en el apartado de RESOLUCION DE PROBLEMAS de este documento.