Contenido

[Contenido 1](#_Toc109307655)

[1. INTRODUCCIÓN 2](#_Toc109307656)

[2. PROTOCOLOS ESTÁNDAR DE COMUNICACIONES EN RED 2](#_Toc109307657)

[2.1 El modelo TCP/IP 2](#_Toc109307658)

[2.2 La librería Apache Commons Net 3](#_Toc109307659)

[3. COMUNICACIÓN CON UN SERVIDOR FTP 4](#_Toc109307660)

[3.1 Java para comunicar con un servidor FTP 6](#_Toc109307661)

[3.2 Subir ficheros al servidor 11](#_Toc109307662)

[3.3 Renombrar y borrar ficheros 13](#_Toc109307663)

[3.4 Descargar ficheros del Servidor 14](#_Toc109307664)

[4. COMUNICACIÓN CON UN SERVIDOR SMTP 16](#_Toc109307665)

[4.1 Instalación de un servidor de correo electrónico 16](#_Toc109307666)

[4.2 Uso de Java para comunicar con un servidor SMTP 18](#_Toc109307667)

[4.3 Uso de Java para comunicar con un servidor SMTP con autenticación 24](#_Toc109307668)

[4.4 Uso de Java para comunicar con un servidor POP3 28](#_Toc109307669)

[5. PROGRAMACIÓN DE SERVIDORES CON JAVA 35](#_Toc109307670)

[5.1 Programa servidor 36](#_Toc109307671)

[5.2 Programa Cliente 37](#_Toc109307672)

[6. DISPONIBILIDAD DEL SERVICIO 38](#_Toc109307673)

[6.1 Requisitos 38](#_Toc109307674)

[6.2 Calidad del servicio 38](#_Toc109307675)

[6.3 Denegación de servicio 40](#_Toc109307676)

1. INTRODUCCIÓN

Los servicios son programas auxiliares utilizados en un sistema informático que permiten gestionar una colección de recursos y prestar su funcionalidad a los usuarios y aplicaciones. Por ejemplo, cuando se envía un documento a una impresora se está utilizando un servicio de impresión. Este servicio permite gestionar y compartir la impresora en red. El único acceso que se tiene al servicio está formado por el conjunto de operaciones que ofrece. Por ejemplo, un servicio de ficheros ofrece operaciones de lectura, escritura o borrado de ficheros.

Todos los servicios de Internet implementan una relación cliente-servidor. En esta unidad didáctica se estudiarán estos servicios y se usará Java para programar clientes de los servicios de Internet más frecuentemente usados.

1. PROTOCOLOS ESTÁNDAR DE COMUNICACIONES EN RED
   1. El modelo TCP/IP

El modelo TCP/IP está compuesto por 4 capas o niveles. La capa de aplicación maneja los protocolos de alto nivel que implementa servicios tales como:

* Conexión remota: Telnet.
* Correo electrónico: SMTP.
* Acceso a ficheros remotos: FTP, NFS, TFTP.
* Resolución de nombres de ordenadores: DNS, WINS.
* World Wide Web: HTTP.

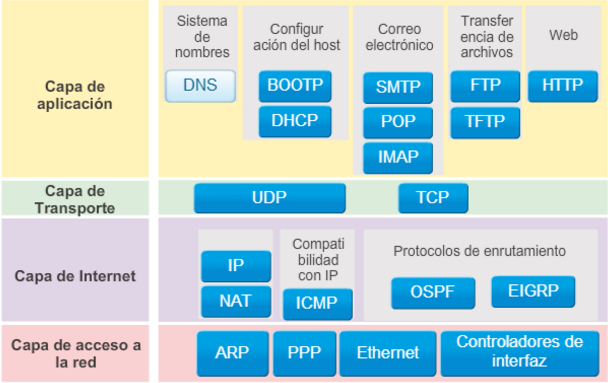


Figura 1.1 – Protocolos de las capas del modelo TCP/IP

Todas las aplicaciones que implementan TCP/IP se basan en un modelo cliente-sevidor:

* **Telnet** (Telecomunication Network): emulación de terminal. Permite a un usuario acceder a una máquina remota y manejarla como si estuviese conectado a ella. Este servicio ha sido sustituido por SSH, ya que envía la información entre cliente y el servidor en forma de texto plano y esto no es seguro. Usado frecuentemente para arreglar fallos de máquinas remotas o para realizar consultas a distancia.
* **SMTP** (Simple Mail Transfer Protocol): protocolo simple de transferencia de correo electrónico. Este estándar especifica el formato exacto de los mensajes que un cliente en una máquina debe enviar al servidor en otra. Administra la transmisión de correo electrónico a través de las redes informáticas.
* **FTP** (File Tranfer Protocol): protocolo de transferencia de ficheros. Permite la transferencia de ficheros entre un cliente y servidor a través de Internet. En los comienzos de Internet se utilizaban los servidores FTP públicos para la descarga de programas. Este servicio está orientado a conexión.
* **TFTP** (Trivial File Tranfer Protocol): protocolo trivial de transferencia de ficheros. A diferencia del protocolo FTP, no necesita autenticación para la transferencia de ficheros entre el cliente y el servidor. Se suele utilizar para transferir ficheros entre ordenadores en una red local en la que no es necesaria una autenticación. Es un servicio no orientado a conexión que utiliza el protocolo UDP.
* **HTTP** (HyperText Transference Protocol): protocolo de transferencia de Hipertexto. Lo usan los navegadores web para realizar peticiones a los servidores web y para recibir las respuestas de ellos. Se trata de un protocolo que especifica los mensajes involucrados en un intercambio petición-respuesta, los métodos, argumentos, resultados y las reglas para representar todo ello en los mensajes.
* **NFS** (Network File System): sistema de ficheros de red que permite a los usuarios el acceso en línea a ficheros que se encuentran en sistemas remotos. De esta forma, el usuario accede a un fichero como si éste fuera un fichero local. Fue desarrollado por Sun Microsystems.
* **SNMP** (Simple Network Management Protocol): protocolo simple de administración de red. Protocolo utilizado para intercambiar información entre los diferentes dispositivos de una red. Permite a los administradores monitorizar, controlar y supervisar el funcionamiento de la red.
* **DNS** (Domain Name System): sistema de nombres de dominio. Sistema que usa servidores distribuidos a lo largo de la red para resolver el nombre de un host IP (nombre de ordenador + nombre de subdominio + nombre de dominio) en una dirección IP. De esta forma, no hay que recordar el nombre del dominio.
  1. La librería Apache Commons Net

Esta librería implementa la parte cliente de muchos protocolos básicos de Internet (FTP, DNS, SMTP, Telnet, TFTP, etc). El objetivo de la librería es proporcionar acceso básico de protocolos y no abstracciones de alto nivel. Por tanto, parte de su diseño viola los principios del diseño de orientación a objetos.

La descarga de la última versión de esta librería suele estar disponible en <https://commons.apache.org/proper/commons-net/download_net.cgi> e incluye una serie de ejemplos.

Aunque de una versión a otra del IDE Eclipse puede haber alguna variación, para integrar esta librería en un proyecto de Eclipse se pueden seguir los siguientes pasos:

* Descargar la librería del enlace anteriormente descrito
* Extraer la información disponible en el archivo descargado con la herramienta adecuada
* Copiar el archivo [commons-net.jar](http://www.java2s.com/Code/JarDownload/commons-net.jar.zip) **a un directorio lib que se cree dentro de la carpeta donde se ubica el proyecto en el que se quiere usar la librería**
* Acceder a las propiedades del proyecto Eclipse
* Seleccionar “Java Build Path”
* Ir a la pestaña “Libraries”
* Pulsar en el botón “Add External JARs…”
* Seleccionar el archivo [commons-net.jar](http://www.java2s.com/Code/JarDownload/commons-net.jar.zip)del directorio lib y aceptar

1. COMUNICACIÓN CON UN SERVIDOR FTP

FTP es una de las herramientas más útiles para el intercambio de ficheros entre diferentes ordenadores y es la forma más habitual de publicación en Internet.

Para usar FTP para transferir ficheros entre dos ordenadores, cada uno debe tener un papel; es decir, uno debe de ser el cliente FTP y el otro el servidor FTP. El cliente envía comandos al servidor (subir, bajar, borrar ficheros, crear un directorio) y el servidor los lleva a cabo. Es posible imaginarse al servidor como un gran contenedor en el que se pueden encontrar gran cantidad de ficheros y directorios.

Existen básicamente 2 formas de acceder a través de FTP:

* **Acceso anónimo:** cuando la conexión con la máquina servidora la realiza un usuario sin autentificar y con escasos privilegios en el servidor. En este caso el usuario es recluido a un directorio público donde solo se le permite descargar ficheros.
* **Acceso autorizado:** el usuario que realiza la conexión con la máquina servidora está registrado y tiene ciertos privilegios en el servidor. En este caso, y una vez autentificado, el usuario es recluido a su directorio personal donde puede subir y bajar ficheros; normalmente se le asigna una cuota de espacio.

FTP utiliza 2 **conexiones** TCP distintas: una **conexión de control** y otra **conexión de transferencia de datos**. La primera se encarga de iniciar y mantener la comunicación entre el cliente y el servidor. La segunda se encarga de enviar datos entre cliente y servidor y solo existe cuando hay datos que transmitir.

Cuando un cliente se conecta a un servidor FTP, el cliente emplea un puerto aleatorio pero el servidor se conecta al puerto 21. Para la transferencia de datos no se utilizan los mismos puertos, por lo que el cliente obtiene un nuevo puerto y el servidor suele usar el puerto 20.

Existen dos **modos de conexión** de los clientes FTP a los servidores FTP:

* **Modo activo:** el servidor siempre crea la conexión de datos en su puerto 20, mientras que en el lado del cliente la conexión de datos se asocia a un puerto aleatorio mayor que el 1024. Para ello, el cliente manda un comando PORT al servidor por la conexión de control, indicándole ese número de puerto, de manera que el servidor pueda abrirle en el puerto especificado una conexión de datos por donde se transferirán los archivos. Esto tiene un problema de seguridad, ya que la máquina cliente debe estar dispuesta a aceptar cualquier conexión de entrada en un puerto superior al 1024, con los problemas que ello implica si se tiene el equipo conectado a una red insegura como Internet. De hecho, los [cortafuegos](https://es.wikipedia.org/wiki/Cortafuegos_(informática)) que se instalen en el equipo para evitar ataques seguramente rechazarán esas conexiones aleatorias. Para solucionar esto está el modo pasivo.
* **Modo pasivo:** cuando el cliente envía un comando PASV sobre la conexión de control, el servidor FTP le indica por la conexión de control el puerto (mayor a 1024 del servidor; por ejemplo, el puerto 2040) al que debe conectarse el cliente. El cliente inicia una conexión desde el puerto siguiente al puerto de control (ejemplo: 1036) hacia el puerto del servidor especificado anteriormente (ejemplo: 2040).

Antes de cada nueva transferencia, tanto en el modo Activo como en el Pasivo, el cliente debe enviar otra vez un comando de control (PORT o PASV, según el modo en el que haya conectado), y el servidor recibirá esa conexión de datos en un nuevo puerto (aleatorio si es en modo pasivo o por el puerto 20 si es en modo activo).

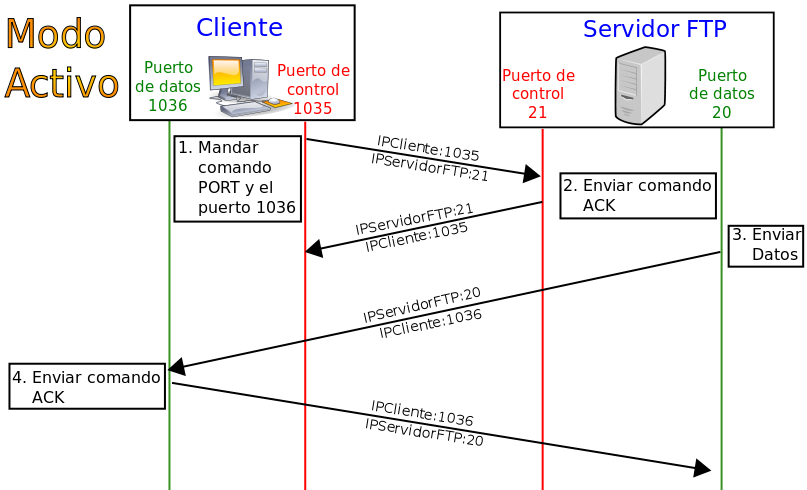


Figura 3.1 – Modo de conexión activo de un cliente FTP

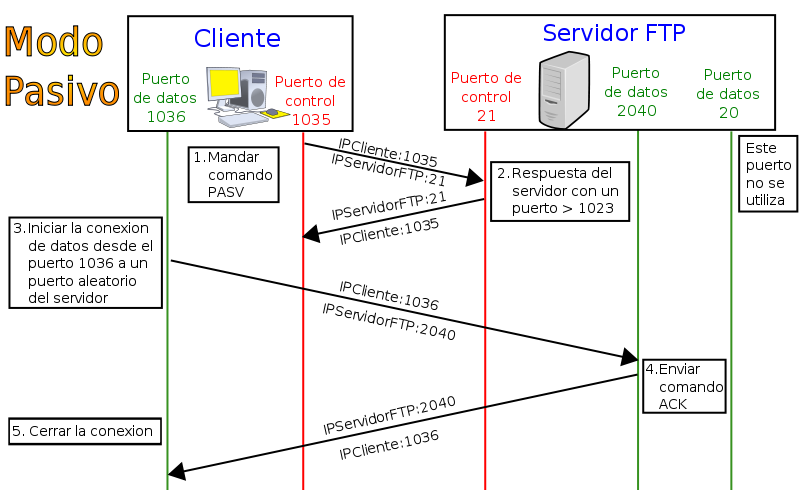


Figura 3.2 – Modo de conexión pasivo de un cliente FTP

* 1. Java para comunicar con un servidor FTP

**Apache Commons Net** proporciona una librería de componentes que permiten implementar el lado cliente de muchos protocolos básicos de Internet, entre ellos el protocolo FTP.

La clase **FTPClient** encapsula toda la funcionalidad necesaria para almacenar y recuperar ficheros de un servidor FTP. Esta clase se encarga de todos los detalles de bajo nivel de la interacción con un servidor FTP. Para poder utilizarla primero es necesario realizar la conexión al servidor con el método *connect()*, comprobar el código de respuesta para ver si ha ocurrido algún error, realizar las operaciones de transferencia y, cuando termine el proceso, cerrar la conexión usando el método *disconnect()*. Algunos de los métodos de esta clase aparecen en la siguiente tabla:

| **MÉTODO** | **MISIÓN** |
| --- | --- |
| void connect(String host) | Abre la conexión con el servidor FTP indicado en host |
| Int getReplyCode() | Devuelve el valor entero del código de respuesta de la última respuesta FTP |
| String getReplyString() | Devuelve el texto completo de la respuesta del servidor FTP |
| void disconnect() | Cierra la conexión con el servidor FTP y restaura los parámetros de conexión a los valores predeterminados |
| boolean login (String user, String passwd) | Inicia sesión en el servidor FTP usando el nombre de usuario y la contraseña proporcionados. Devuelve true si se inicia la sesión con éxito; en caso contrario, devuelve false |
| boolean logout() | Sale del servidor FTP |
| String pritWorkingDirectory() | Devuelve el nombre de la ruta del directorio de trabajo actual |
| FTPFile [] listFiles() | Obtiene una lista de ficheros del directorio actual como un array de objetos FTPFile |
| FTPFile [] listFiles(String path) | Obtiene una lista de ficheros del directorio indicado en el path |
| String[] listName() | Obtiene una lista de ficheros del directorio actual como un array de cadenas |
| FTPFile[] listDirectories() | Obtiene la lista de los directorios que se encuentran en el directorio de trabajo actual |
| FTPFile[] listDirectories(String parent) | Obtiene la lista de los directorios que se encuentran en el directorio especificado en parent |
| Boolean changeWorkingDirectory (String pathname) | Cambia el directorio de trabajo actual de la sesión de FTP al indicado en pathname |
| Boolean changeToParentDirectory() | Cambia al directorio padre del directorio de trabajo actual |
| Boolean setFileType (int filType) | Establece el tipo de fichero a transferir: ASCII\_FILE\_TYPE (fichero ASCII), BINARY\_FILE\_TYPE (imagen binaria), etc. |
| Boolean storeFile(String nombre, InputStream local) | Almacena un fichero en el servidor con el nombre indicado tomando como entrada el Input Stream |
| Boolean retrieveFile (String nombre, OutputStream local) | Recupera un fichero del servidor y lo escribe en el OutputStream dado |
| Boolean deleteFile(String pathname) | Elimina un fichero en el servidor FTP |
| Boolean rename (String antiguo, String nuevo) | Cambia el nombre de un fichero en el servidor FTP |
| Boolean removeDirectory (String pathname) | Elimina un diretorio en el servidor FTP (si está vacío) |
| Boolean makeDirectory(String pathname) | Cea un nuevo subdirectorio en el servidor FTP en el directorio actual |

Todos los métodos de comunicación con el servidor pueden lanzar IOException. El servidor FTP puede optar por cerrar antes de tiempo una conexión si el cliente ha estado inactivo durante más de un periodo de tiempo (generalmente 900 segundos). La clase FTPClient detectará un cierre prematuro de la conexión con el servidor FTP y se puede producir la excepción FTPConnectionClosedException.

Lo más normal es conectarse a un servidor FTP con un nombre de usuario y su clave. Para identificarse se usará el método *login()* que devuelve true si la conexión se realiza correctamente. Para la desconexión se usará el método *logout().*

La clase **FTPReply** almacena un conjunto de constantes para códigos de respuesta FTP, los cuales están disponibles en la RFC 959 del IETF (<https://www.ietf.org/rfc/rfc959.txt>). El protocolo FTP usa un esquema de códigos de respuesta donde cada uno de sus dígitos tiene un significado especial. Son números de 3 dígitos en ASCII y el primer dígito indica si la respuesta es buena, mala o incompleta.

**Ejemplo 1 – Conectarse al servidor FTP** [ftp.rediris.es](ftp://ftp.rediris.es/)**, comprobar si la conexión se ha realizado correctamente y cerrar la conexión.**

**import** java.io.IOException;

**import** org.apache.commons.net.ftp.FTPClient;

**import** org.apache.commons.net.ftp.FTPReply;

**public** **class** ClienteFTP1 {

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

FTPClient cliente = **new** FTPClient();

String servFTP = "ftp.rediris.es"; // servidor FTP

// Respuesta del servidor FTP

System.***out***.println("Nos conectamos a "+servFTP);

cliente.connect(servFTP);

System.***out***.println(cliente.getReplyString());

// Código de respuesta

**int** respuesta = cliente.getReplyCode();

// Comprobación del codigo de respuesta

**if** (!FTPReply.*isPositiveCompletion*(respuesta)) {

cliente.disconnect();

System.***out***.println("Conexión rechazada:" + respuesta);

System.*exit*(0);

}

// Desconexión del servidor FTP

cliente.disconnect();

System.***out***.print("Conexión finalizada");

}

}

**Resultado** (orientativo):

Nos conectamos a ftp.rediris.es

220- Bienvenido al FTP anónimo de RedIRIS.

220-Welcome to the RedIRIS anonymous FTP server.

220 Only anonymous FTP is allowed here

Conexión finalizada

**Ejemplo 2 – Conectarse al servidor FTP** [ftp.rediris.es](ftp://ftp.rediris.es/) **utilizando un acceso anónimo (usuario anonymous y clave anonymous) para mostrar la lista de ficheros del directorio actual utilizando el método *listFiles()*, que devuelve un array de la clase FTPFile con información de los ficheros y directorios encontrados; se recorrerá el array visualizando el nombre del fichero o directorio y el tipo que puede ser fichero, directorio o enlace simbólico.**

**import** java.io.IOException;

**import** java.net.SocketException;

**import** org.apache.commons.net.ftp.\*;

**public** **class** ClienteFTP2 {

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

FTPClient cliente = **new** FTPClient();

String servFTP = "ftp.rediris.es"; // servidor FTP

// Respuesta del servidor FTP

System.***out***.println("Nos conectamos a "+servFTP);

String usuario="anonymous";

String clave="anonymous";

**try** {

cliente.connect(servFTP);

**boolean** login = cliente.login(usuario, clave);

**if** (login)

System.***out***.println("Login correcto");

**else** {

System.***out***.println("Login incorrecto...");

cliente.disconnect();

System.*exit*(1);

}

System.***out***.println("Directorio actual:" + cliente.printWorkingDirectory());

FTPFile[] files = cliente.listFiles();

System.***out***.println("Ficheros en el directorio actual: "+ files.length);

// Array para visualizar el tipo de fichero

String tipos[] = {"Fichero", "Directorio", "Enlace simb." };

**for** (**int** i =0; i < files.length; i++) {

System.***out***.println("\t"+files[i].getName()+"=>"+tipos[files[i].getType()]);

}

**boolean** logout = cliente.logout();

**if** (logout)

System.***out***.println("Logout del servidor FTP.....");

**else**

System.***out***.println("Error al hacer un Logout....");

cliente.disconnect();

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

} **catch** (SocketException e) {

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

e.printStackTrace();

} **catch** (IOException e) {

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

e.printStackTrace();

}

}

}

**Resultado** (orientativo):

Nos conectamos a ftp.rediris.es

Login correcto

Directorio actual:/

Ficheros en el directorio actual: 12

.=>Directorio

..=>Directorio

debian=>Fichero

incoming=>Directorio

ls-lR=>Fichero

ls-lR.Z=>Fichero

ls-lR.gz=>Fichero

mirror=>Directorio

outgoing=>Directorio

pub=>Directorio

sites=>Directorio

welcome.msg=>Fichero

Logout del servidor FTP.....

Desconectado......

La clase **FTPFile** se utiliza para representar la información acerca de los ficheros almacenados en un servidor FTP. Algunos de sus métodos son los siguientes:

| **MÉTODOS** | **MISIÓN** |
| --- | --- |
| String getName() | Devuelve el nombre del fichero |
| long getSize() | Devuelve el tamaño del fichero en bytes |
| int getType | Devuelve el tipo del fichero, 0 si es un fichero (FYLE\_TYPE) y 2 un enlace simbólico (SYMBOLIC\_LINK\_TYPE) |
| String getUser() | Devuelve el nombre del usuario propietario del fichero |
| boolean isDirectory() | Devuelve true si el fichero es un directorio |
| booblean isFile() | Devuelve true si es un fichero |
| boolean isSymbolicLink() | Devuelve true si es un enlace simbólico |

* 1. Subir ficheros al servidor

En los siguientes ejemplos se asume que se dispone del **servidor proFTPd** debidamente instalado y configurado.

En general, y para subir ficheros a los servidores FTP suele ser necesario disponer de un usuario, su clave, un espacio en el servidor así como **los permisos necesarios** para dicho espacio. Además, dentro del servidor es necesario definir un directorio donde se van a subir los archivos y, en origen, la ubicación y nombre de los ficheros que se pretenden subir al servidor.

Con el método *setFileType()* de la clase FTPClient se define el tipo de fichero a subir. Aunque existen diversos tipos de ficheros, se suele usar la constante entera BINARY\_FILE\_TYPE de la **clase FTP**, ya que permite subir ficheros de cualquier tipo.

Mediante un stream de entrada con los datos a subir y la utilización del método *storeFile()* se podrá subir el archivo al servidor FTP.

Por último, se cierra el flujo de entrada.

**Ejemplo 3 – Subir los ficheros de nombre fichero.txt y fichero.jpg a la carpeta DIRECTORIO (que previamente tiene que existir) del directorio por defecto del servidor proFTPd. La ubicación origen del fichero de texto y de la imagen será un directorio de la unidad C: desde la que se va a realizar la subida de los ficheros. La dirección IP asociada al servidor FTP es la 192.168.1.135 y los datos de login en el servidor FTP serán dinux para el usuario y dinux para la contraseña**

**import** java.io.BufferedInputStream;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.net.SocketException;

**import** org.apache.commons.net.ftp.\*;

**public** **class** SubirFicheros {

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

FTPClient cliente = **new** FTPClient (); // cliente

String servidor = "192.168.1.135"; // Servidor

String user="dinux";

String pasw="dinux";

String direct ="DIRECTORIO";

String fichero ="C:/temp/fichero.txt";

String ficherodestino="fichero.txt";

String fichero2 ="C:/temp/fichero.jpg";

String ficherodestino2="fichero.jpg";

**try** {

System.***out***.println("Conectándose a "+ servidor);

cliente.connect(servidor);

**boolean** login=cliente.login(user, pasw);

**if** (login) {

cliente.changeWorkingDirectory(direct);

cliente.setFileType(FTP.***BINARY\_FILE\_TYPE***);

// Stream de entrda con el fichero a subir

BufferedInputStream in = **new** BufferedInputStream(**new** FileInputStream(fichero));

cliente.storeFile(ficherodestino, in);

// Stream de entrada con el fichero a subir

in = **new** BufferedInputStream(**new** FileInputStream(fichero2));

cliente.storeFile(ficherodestino2, in);

System.***out***.println("Ficheros subidos");

in.close();

cliente.logout();

cliente.disconnect();

}

} **catch** (SocketException e) {

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

e.printStackTrace();

} **catch** (IOException e) {

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

e.printStackTrace();

}

}

}

**Resultado**:

Conectándose a 192.168.1.135

Ficheros subidos

* 1. Renombrar y borrar ficheros

Para renombrar ficheros se puede usar el método *rename (antiguo, nuevo)* y para borrar un fichero el método *deleteFile(fichero)*. Devuelven true si se ha ejecutado el comando correctamente.

**Ejemplo 4 – Renombrar el fichero.txt como ficherorenombrado.txt y borrar el fichero.jpg dentro de la carpeta DIRECTORIO (que previamente tiene que existir) del directorio por defecto del servidor proFTPd. La dirección IP asociada al servidor FTP es la 192.168.1.135 y los datos de login en el servidor FTP serán dinux para el usuario y dinux para la contraseña**

**import** java.io.IOException;

**import** java.net.SocketException;

**import** org.apache.commons.net.ftp.\*;

**public** **class** RenombrarBorrarFicheros {

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

FTPClient cliente = **new** FTPClient (); // cliente

String servidor = "192.168.1.135"; // Servidor

String user="dinux";

String pasw="dinux";

String direct ="DIRECTORIO";

**try** {

System.***out***.println("Conectandose a "+ servidor);

cliente.connect(servidor);

**boolean** login=cliente.login(user, pasw);

**if** (login) {

cliente.changeWorkingDirectory(direct);

// Renombramos fichero1.txt por ficheromodificado.txt

**if** (cliente.rename("fichero.txt", "ficherorenombrado.txt")) {

System.***out***.println("Fichero renombrado..");

} **else** {

System.***out***.println("No se ha podido renombrar el fichero");

}

// Borramos el fichero fichero2.jpg

**if** (cliente.deleteFile("fichero.jpg")) {

System.***out***.println("Fichero borrado..");

} **else** {

System.***out***.println("No se ha podido borrar el fichero");

}

cliente.logout();

cliente.disconnect();

}

} **catch** (SocketException e) {

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

e.printStackTrace();

} **catch** (IOException e) {

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

e.printStackTrace();

}

}

}

**Resultado**:

Conectandose a 192.168.1.135

Fichero renombrado..

Fichero borrado..

* 1. Descargar ficheros del Servidor

Para descargar un fichero desde el servidor FTP hacia un disco duro se puede utilizar el método *retrieveFile(String remote, OutputStream local)*, que devuelve true si se ha descargado el fichero correctamente. Es necesario un stream de salida para escribir el fichero en el disco duro así como conocer el directorio origen desde el que descargar el fichero.

**Ejemplo 5 – Descargar el fichero de nombre ficherorenombrado.txt (que está dentro de la carpeta DIRECTORIO del directorio por defecto del servidor proFTPd) al directorio C:/temp del equipo al que se quiere descargar el fichero. La dirección IP asociada al servidor FTP es la 192.168.1.135 y los datos de login en el servidor son *dinux* para el usuario y *dinux* para la contraseña**

**import** java.io.BufferedOutputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.net.SocketException;

**import** org.apache.commons.net.ftp.\*;

**public** **class** DescargarFicheros {

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

FTPClient cliente = **new** FTPClient (); // cliente

String servidor = "192.168.1.135"; // Servidor

String user="dinux";

String pasw="dinux";

String direct ="DIRECTORIO";

String origen="ficherorenombrado.txt";

String destino="C:/temp/ficherorenombrado.txt";

**try** {

System.***out***.println("Conectandose a "+ servidor);

cliente.connect(servidor);

**boolean** login=cliente.login(user, pasw);

**if** (login) {

cliente.changeWorkingDirectory(direct);

// Stream de salida para recibir el fichero descargado

BufferedOutputStream out = **new** BufferedOutputStream (**new** FileOutputStream (destino));

**if** (cliente.retrieveFile(origen, out)) {

System.***out***.println("Fichero descargado..");

} **else** {

System.***out***.println("No se ha podido descargar el fichero");

}

out.close();

cliente.logout();

cliente.disconnect();

}

} **catch** (SocketException e) {

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

e.printStackTrace();

} **catch** (IOException e) {

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

e.printStackTrace();

}

}

}

**Resultado**:

Conectandose a 192.168.1.135

Fichero descargado..

1. COMUNICACIÓN CON UN SERVIDOR SMTP

**SMTP** (Simple Mail Transfer Protocol) es el protocolo estándar de Internet para el intercambio de correo electrónico. Funciona con comandos de texto que se envían al servidor SMTP (por defecto, usa el puerto 25). A cada comando que envía el cliente le sigue una respuesta del servidor, compuesto por un número y un mensaje descriptivo. Las especificaciones de este protocolo se definen en la RFC 2821.

* 1. Instalación de un servidor de correo electrónico

En el servicio de correo electrónico se distinguen **3 agentes diferentes** que intervienen en el proceso de envío, recepción y lectura de los mensajes:

* **MUA**: agente de usuario de correo.
* **MTA**: agente de transporte de correo.
* **MDA**: agente de entrega de correo.

Cuando un **usuario A** envía un correo electrónico a un **usuario B** a través de un cliente de correo electrónico o **MUA** (como puede ser Thunderbird, por ejemplo), éste enruta o redirige el correo hacia un **MTA** (como puede ser un servidor SMTP o un servidor de correo saliente como Sendmail). Este correo puede reenviarse a través de varios servidores de correo saliente.

MUA -----------→ MTA --------------→ MTA ---------------→ MDA -------→ MUA

usuario A Intranet Internet Intranet usuario B

El MTA entrega el correo al **MDA**, el cual almacena el correo en el buzón que el usuario tiene en el servidor. Los protocolos que usan los MDAs suelen ser POP3 o IMAP.

Finalmente, el usuario B accede mediante su cliente de correo electrónico o MUA a su buzón de correo electrónico y visualiza o descarga el correo electrónico enviado por el usuario A.

Los servidores SMTP son programas que permiten enviar correo electrónico a otros servidores SMTP. El servidor de correo **Argosoft** permite la instalación de un servidor de correo SMTP (MTA) y también de un servidor de correo POP3 (MDA). En la web <http://argosoft-mail-server.uptodown.com/> está disponible un archivo para poder realizar la instalación de este servidor de correo.

A continuación se va a realizar una instalación en local del servidor de correo Argosot, la cual se simplificará desactivando momentáneamente el Firewall del sistema operativo. Después de la instalación, se habrá instalado, entre otras cosas, un servidor SMTP y otro POP3.

Una vez instalado, se procederá a la configuración de ciertas opciones desde Tools/Options/General. Estas opciones pueden ser las siguientes:

* Servidor de DNS: 8.8.8.8 (por ejemplo)
* Automatically Start the Server
* Permitir Relay
* Local Host: 172.20.105.120

En la siguiente figura se muestra esta configuración:

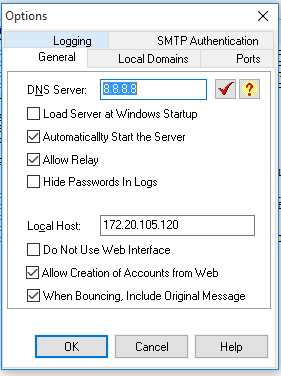


Figura 5.1 – Configuración básica del servidor de correo Argosoft

Una configuración adicional se consigue activando las siguientes opciones desde la opción Tools/Options/Logging del servidor Argosoft:

* Log SMTP commands
* Log SMTP Conversations with Exchangers
* Log to File

A continuación se dan de alta en el servidor dos usuarios: *pepe* y *juan*, desde la opción Tools/Users de Argosoft. La contraseña de cada uno de ellos coincide con el nombre del usuario.

Normalmente, cuando se crea una cuenta de correo en un proveedor de servicios de internet, el proveedor proporciona los datos del servidor POP3 (o IMAP) y del servidor SMTP. Estos son necesarios para configurar clientes de correo como Microsoft Outlook, Mozilla Thunderbird, Eudora, etc. El primero se usa para recibir los mensajes (configuración del correo entrante) y el segundo para enviar los mensajes (configuración del correo saliente). Se puede usar el servidor SMTP recién instalado de Argosoft para enviar correos, en vez de utilizar el proporcionado por el proveedor de internet.

* 1. Uso de Java para comunicar con un servidor SMTP

La librería *Apache Commons Net* proporciona la clase **SMTPClient**, que extiende de SMTP, la cual encapsula toda la funcionalidad necesaria para enviar ficheros a través de un servidor SMTP. Esta clase se encarga de todos los detalles de bajo nivel de interacción con un servidor SMTP.

Al igual que con todas las clases derivadas de la clase SocketClient, antes de hacer cualquier operación es necesario conectarse al servidor y, una vez finalizada la interacción con el servidor, es necesario desconectarse. Una vez conectados es necesario comprobar el código de respuesta SMTP para ver si la conexión se ha realizado o no correctamente.

La clase **SMTPReply** (similar a la clase FTPReply) almacena un conjunto de constantes para códigos respuesta SMTP. Para interpretar el significado de los códigos se puede consultar la RFC 2821 (<https://www.ietf.org/rfc/rfc2821.txt>).

La clase **SMTPClient** presenta dos tipos de constructores:

|  |  |
| --- | --- |
| **CONSTRUCTOR** | **Función** |
| SMTPClient() | Constructor por defecto |
| SMTPClient() (String codificacion) | Se establece una codificación en el constructor (BASE64, Binary, 8BIT, etc.) |

La clase SMTPClient utiliza el método *connect()* de la clase SocketClient para conectarse al servidor y el método *disconnect()*de la clase **SMTP** para desconectarse del servidor SMTP. Para conectarse a un servidor SMTP cuyo puerto de escucha sea distinto del 25, habría que indicar en la conexión el número de puerto: *connect (host, puerto).*

Alguno de los métodos de la clase SMTPClient son los siguientes:

|  |  |
| --- | --- |
| **Método** | **Función** |
| boolean addRecipient(String address) | Añade la dirección de correo de un destinatario usando el comando **RCPT** |
| boolean completePendingCommand() | Este método se utiliza para finalizar la transacción y verificar éxito o el fracaso de la respuesta del servidor |
| boolean login() | Inicia sesión en el servidor SMTP enviando el comando **HELO** |
| boolean login(String, hostname) | Igual que la anterior pero envía el nombre del host como argumento |
| boolean logout() | Finaliza la sesión con el servidor enviando el comando **QUIT** |
| Writer sendMessageData() | Envía el comando **DATA** para después enviar el mensaje de correo. La clase Writer se usará para escribir secuencias de caracteres como cabeceras y el cuerpo del mensaje |
| boolean sendShortMessageData (String message) | Método útil para el envío de mensajes cortos |
| boolean sendSimpleMessage(String remitente, String[] destinatarios, String message) | Un método útil para el envío de un correo electrónico corto. Se especifica el remitente, los destinatarios y el mensaje |
| boolean sendSimpleMessage(String remitente, String destinatario, String message) | Igual que el anterior pero el mensaje solo va dirigido a un usuario |
| boolean setSender(String address) | Se especifica la dirección del remitente usando el comando **MAIL** |
| boolean verify(String username) | Comprueba qu un nombre de usuario o dirección de correo es válida (comando **VRFY**) |

**NOTA importante:** para que algunos de los siguientes ejemplos relacionados con el servicio de correo electrónico funcionen en los equipos del aula, podría ser necesario agregar reglas en el Firewall, ya que el tráfico a través de los puertos 587 y 995 suele estar bloqueado.

**Ejemplo 6 – Conectarse desde Java a un servidor SMTP para enviar un mensaje de correo a una cuenta de correo electrónico de GMAIL**

**import** java.io.IOException;

**import** java.net.SocketException;

**import** org.apache.commons.net.smtp.SMTPClient;

**import** org.apache.commons.net.smtp.SMTPReply;

**public** **class** ClienteSMTP {

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

SMTPClient client = **new** SMTPClient ();

**try** {

**int** respuesta;

// Nos conectamos

client.connect("172.20.105.120");

System.***out***.println(client.getReplyString());

respuesta = client.getReplyCode();

**if** (!SMTPReply.*isPositiveCompletion*(respuesta)){

client.disconnect();

System.***err***.println("Conexión rechazada");

System.*exit*(1);

}

} **catch** (SocketException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

**if** (client.isConnected())

//Envío de un mensaje

**if** (client.isConnected())

**try** {

client.login(); // Inicio de sesión comando -HELO

String destinatario="dm2prosciudadjardin@gmail.com";

String mensaje="Hola. \nSaludos. \n Adios.";

String remitente="yo@localhost.es";

client.sendSimpleMessage(remitente, destinatario, mensaje);

System.***out***.println("TERMINADO EL ENVÍO DEL CORREO");

client.disconnect(); // Final de sesión comando -UIT

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**Resultado en la consola del IDE Eclipse:**

220 172.20.105.120 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)

TERMINADO EL ENVÍO DEL CORREO

**Resultado en el log del servidor SMTP (solo desde equipos del aula):**

El mensaje es aceptado para su envío, pero la conexión con el exterior es rechazada:

25/11/2016 18:12:10 - Requested SMTP connection from 172.20.105.120

25/11/2016 18:12:10 - ( 8) 220 172.20.105.120 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)

25/11/2016 18:12:10 - ( 8) HELO DM2-PROF

25/11/2016 18:12:10 - ( 8) 250 Welcome [172.20.105.120], pleased to meet you

25/11/2016 18:12:10 - ( 8) MAIL FROM:<yo@localhost.es>

25/11/2016 18:12:10 - ( 8) 250 Sender "yo@localhost.es" OK...

25/11/2016 18:12:10 - ( 8) RCPT TO:<dm2prosciudadjardin@gmail.com>

25/11/2016 18:12:10 - ( 8) 250 Recipient "dm2prosciudadjardin@gmail.com" OK...

25/11/2016 18:12:10 - ( 8) DATA

25/11/2016 18:12:10 - ( 8) 354 Enter mail, end with "." on a line by itself

25/11/2016 18:12:10 - Received 31 bytes

25/11/2016 18:12:10 - ( 8) 250 Message accepted for delivery. <brshwjrz8za36rb.251120161812@172.20.105.120>

25/11/2016 18:12:10 - ( 8)

25/11/2016 18:12:10 - **Error: [10054] Connection reset by peer**

25/11/2016 18:12:10 - SMTP connection with 172.20.105.120 ended. ID=8

**Resultado en servidores SMTP fuera del aula:**

* En algunos servidores de correo, por ejemplo en el servidor SMTP de GMAIL, este mensaje no sería admitido y no llegaría a su destino porque no está bien construido. Nuestro servidor SMTP mostraría en su log el siguiente mensaje de error: “*Our system has detected that this message is not RFC 2822 compliant,....*”. Con otros servidores SMTP diferentes, puede que llegue a destino como correo spam.

La clase **SimpleSMTPHeader** se utiliza para la construcción de una cabecera mínima aceptable para el envío de un mensaje de correo electrónico. El constructor es el siguiente:

SimpleSMTPHeader (string from, String to, String subjet)

Crea una nueva instancia de SimpleSMTPHeader inicializándola con los valores dados en los siguientes campos de cabecera:

* *from*: dirección de correo origen.
* *to*: dirección de correo destino.
* *subject*: asunto del mensaje.

Proporciona los siguientes métodos:

|  |  |
| --- | --- |
| **MÉTODO** | **Función** |
| void addCC(String address) | Agrega una dirección de correo electrónico a la lista CC |
| void addHeaderField(String headerField, String value) | Agrega un campo de encabezado arbitrario con el valor dado a la cabecera del artículo. |
| String toString() | Convierte el SimpleSMTPHeader a una cadena que contiene el encabezado con el formato correcto, incluyendo una línea en blanco para separar cabecera y cuerpo del mensaje. |

**Ejemplo 7 – Conectarse desde Java a un servidor SMTP para enviar un mensaje de correo a dos destinatarios diferentes de correo electrónico de GMAIL, usando la clase SimpleSMTPHeader para la gestión de la cabecera del mensaje**

**import** java.io.IOException;

**import** java.io.Writer;

**import** java.net.SocketException;

**import** org.apache.commons.net.smtp.SMTPClient;

**import** org.apache.commons.net.smtp.SMTPReply;

**import** org.apache.commons.net.smtp.SimpleSMTPHeader;

**public** **class** ClienteSMTP2 {

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

SMTPClient client = **new** SMTPClient ();

**try** {

**int** respuesta;

// Nos conectamos

client.connect("172.20.105.120");

System.***out***.println(client.getReplyString());

respuesta = client.getReplyCode();

**if** (!SMTPReply.*isPositiveCompletion*(respuesta)){

client.disconnect();

System.***err***.println("Conexión rechazada");

System.*exit*(1);

}

} **catch** (SocketException e) {

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

e.printStackTrace();

} **catch** (IOException e) {

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

e.printStackTrace();

}

**if** (client.isConnected())

**try** {

client.login();

String origen="pepe@localhost.es";

String destino1="dm2prosciudadjardin@gmail.com";

String destino2="ernestois.icjardin@gmail.com";

String asunto="Prueba desde el programa ClienteSMTP2.java";

String mensaje="Hola: \nEste mensaje ha salido desde un programa Java que se conecta con un servidor SMTP. \nSaludos cordiales.";

SimpleSMTPHeader cabecera = **new** SimpleSMTPHeader (origen, destino1, asunto);

cabecera.addCC(destino2);

// Establecer el correo de origen

client.setSender(origen);

// Añadir correos de destino

client.addRecipient(destino1);

client.addRecipient(destino2);

// Enviar el DATA

Writer writer= client.sendMessageData();

**if** (writer == **null**) {

System.***out***.println("Fallo al enviar DATA");

System.*exit*(1);

}

writer.write(cabecera.toString());

writer.write(mensaje);

writer.close();

**if** (!client.completePendingCommand()) {

System.***out***.println("FALLO AL FINALIZAR LA TRANSACCIÓN");

System.*exit*(1);

}

System.***out***.println("Mensajes de correo enviados desde Java hacia un servidor SMTP.");

client.logout();

client.disconnect();

} **catch** (IOException e) {

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

e.printStackTrace();

}

}

}

**Resultado en la consola del IDE Eclipse:**

220 172.20.105.120 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)

Mensajes de correo enviados desde Java hacia un servidor SMTP.

**Resultado (fuera del aula):**

El mensaje que recibiría uno de los destinatarios de GMAIL sería el que se muestra en la siguiente figura:



Figura 5.2 – Recepción del mensaje como spam en una de las cuentas de destino

**NOTA importante:**

* El mensaje enviado podría llegar a la carpeta de Spam de las cuentas de correo de destino o bien ser rechazado, ya que el servidor SMTP local podría no tener resolución inversa del servicio DNS.
  1. Uso de Java para comunicar con un servidor SMTP con autenticación

Hasta ahora se ha usado el servidor SMTP Argosoft sin necesidad de autenticación, ya que por defecto, su instalación no la activa. La autenticación SMTP se configura con el fin de elevar los niveles de seguridad y eficacia del servicio de correo electrónico. Normalmente los servidores obligan a autenticarse porque si no, cualquiera podría enviar un correo electrónico desde ellos. La autenticación se realiza a través de la verificación de un nombre y una contraseña.

La librería Apache Commons Net proporciona la clase **AuthenticatingSMTPClient** (extiende SMTPClient) con soporte de autenticación SMTP. La clase SMTPSClient proporciona soporte SMTP sobre el protocolo **SSL** (Secure Socket Layer-capa de conexión segura). SSL es un protocolo criptográfico empleado para realizar conexiones seguras entre un cliente y un servidor. **TLS** (Transport Layer Security- seguridad de la capa de transporte) es el protocolo sucesor de **SSL**.

Algunos de los **métodos** usados para realizar la autenticación SMTP son los siguientes:

|  |  |
| --- | --- |
| **MÉTODO** | **Función** |
| void setKeyManager(KeyManager newKeyManager) | Para obtener el certificado para la autenticación se usa la interface KeyManager. Con este método se establece la clave para llevar a cabo la autenticación. Las KeyManager se pueden crear usando un KeyManagerFactory. |
| boolean execTLS() | Ejecuta el comando STARTTLS. La palabra clave STARTTLS se usa para indicarle al cliente SMTP que el servidor SMTP está en disposición de negoriar el uso de TLS. Devuelve true si el comando y la negociación han tenido éxito. |

La clase AuthenticatingSMTPClient presenta varios **constructores**. Para crear un nuevo cliente de autenticación SMTP se usará el constructor por defecto:

*public AuthenticatingSMTPClien() throws NoSuchAlgorithmException*

Este constructor puede lanzar la excepción *NoSuchAlgorthmException*, la cual se produce cuando un algoritmo criptográfico, habiéndose solicitado, no está disponible en el entorno. Algunos de los métodos de la clase son los siguientes:

|  |  |
| --- | --- |
| **MÉTODO** | **Función** |
| auth (AuthenticatingSMTPClient.AUTH\_METHOD method, String username, String password) | Se envía el comando AUTH con el método seleccionado para autenticarse en el servidor SMTP, se envía el nombre del usuario y su clave. |
| int ehlo(String hostname) | Método para enviar el comando ESMTP (Extended SMTP-SMTP extendido) EHLO al servidor, devuelve el código de respuesta. |

Los valores para el método de autenticación son: CRAM\_MD5 la contraseña se envía encriptada, LOGIN y PLAIN donde la contraseña se envía sin encriptar, como texto plano (no importa ya que la autenticación se va a realizar sobre un canal cifrado) y XOAUTH es un mecanismo de autenticación SASL que se basa en firmas OAuth.

**Ejemplo 8 – Enviar un mensaje, con autenticación previa, usando el servidor SMTP de Gmail. Los datos proporcionados por Gmail son:**

* **Servidor de correo saliente (SMTP) – requiere TLS o SSL: *smtp.gmail.com***
* **Puerto para TLS/STARTTLS: *587***
* **Puerto para SSL: *465***

**NOTA importante:** es posible que el programa Java que da solución a este ejercicio no valide la contraseña de la cuenta de correo *dm2prosciudadjardin@gmail.com* por alguna de las razones explicadas en el enlace <https://support.google.com/mail/answer/14257?rd=1> Una solución para resolver este problema pasa por cambiar la contraseña a citada cuenta o bien por crear una nueva cuenta de correo de Gmail.

**Código en Java:**

**import** java.io.IOException;

**import** java.io.Writer;

**import** java.security.InvalidKeyException;

**import** java.security.KeyStoreException;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.UnrecoverableKeyException;

**import** java.security.spec.InvalidKeySpecException;

**import** javax.net.ssl.KeyManager;

**import** javax.net.ssl.KeyManagerFactory;

**import** org.apache.commons.net.smtp.\*;

**public** **class** ClienteSMTP3 {

**public** **static** **void** main (String[] args) **throws** IOException, UnrecoverableKeyException, KeyStoreException, InvalidKeyException, InvalidKeySpecException {

// Creación de un cliente SMTP seguro

AuthenticatingSMTPClient client = **new** AuthenticatingSMTPClient();

// Datos del usuario y del servidor

String server = "smtp.gmail.com";

String usuario="ernestois.icjardin@gmail.com";

String password="PONER\_AQUI\_LA\_CONTRASEÑA";

**int** puerto = 587;

**try** {

**int** respuesta;

// Creación de la clave para establecer un canal seguro

KeyManagerFactory kmf = KeyManagerFactory.*getInstance*(KeyManagerFactory.*getDefaultAlgorithm*());

kmf.init(**null**, **null**);

KeyManager km=kmf.getKeyManagers()[0];

// Conexión al servidor SMTP de Gmail

System.***out***.println (server + puerto);

client.connect(server, puerto);

System.***out***.println("1 - "+ client.getReplyString());

// Establecimiento de la clave para la comunicación segura

client.setKeyManager(km);

respuesta = client.getReplyCode();

**if** (!SMTPReply.*isPositiveCompletion*(respuesta)) {

client.disconnect();

System.***err***.println("Conexión rechazada");

System.*exit*(1);

}

// Envío del comando EHLO

client.ehlo(server); // es necesario

System.***out***.println("2 - " + client.getReplyString());

**if** (client.execTLS()) {

System.***out***.println("3 - " + client.getReplyString());

// Autenticación con el servidor

**if** (client.auth(AuthenticatingSMTPClient.AUTH\_METHOD.***PLAIN***, usuario, password)) {

System.***out***.println("4 - "+ client.getReplyString());

String destino1 = "hobetuzoutlook00@gmail.com";

String asunto = "Prueba de SMTPClient3 con GMAIL";

String mensaje = "Hola! \n Envío de saludos usando GMAIL.\n Adiós.\n.";

// Creación de la cabecera

SimpleSMTPHeader cabecera = **new** SimpleSMTPHeader (usuario, destino1, asunto);

// El nombre de usuario y el email de origen coinciden

client.setSender(usuario);

client.addRecipient(destino1);

System.***out***.println("5 - "+client.getReplyString());

// Envío del DATA

Writer writer = client.sendMessageData();

**if** (writer == **null**) {

System.***out***.println("Fallo al enviar DATA");

System.*exit*(1);

}

writer.write(cabecera.toString()); // cabecera

writer.write(mensaje);

writer.close();

System.***out***.println("6 - "+client.getReplyString());

**boolean** exito = client.completePendingCommand();

System.***out***.println("7 - "+client.getReplyString());

**if** (!exito) {

System.***out***.println("FALLO AL FINALIZAR TRANSACCIÓN");

System.*exit*(1);

}

}**else**

System.***out***.println("USUARIO NO AUTENTIFICADO");

} **else**

System.***out***.println("FALLO AL EJECUTAR STARTTLS");

} **catch** (NoSuchAlgorithmException e) {

System.***err***.println("Could not connect to server.");

e.printStackTrace();

System.*exit*(1);

}

client.disconnect();

System.***out***.println("Fin envio");

System.*exit*(1);

} // Fin del main

}

**Resultado en Java:**

Estas son las respuestas que envía el servidor SMTP de Gmail, las cuales se muestran en la consola de Java:

smtp.gmail.com587

1 - 220 smtp.gmail.com ESMTP pd2sm11031445wjb.31 - gsmtp

2 - 250-smtp.gmail.com at your service, [83.43.203.148]

250-SIZE 35882577

250-8BITMIME

250-STARTTLS

250-ENHANCEDSTATUSCODES

250-PIPELINING

250-CHUNKING

250 SMTPUTF8

3 - 220 2.0.0 Ready to start TLS

4 - 235 2.7.0 Accepted

5 - 250 2.1.5 OK pd2sm11031445wjb.31 - gsmtp

6 - 354 Go ahead pd2sm11031445wjb.31 - gsmtp

7 - 250 2.0.0 OK 1479508198 pd2sm11031445wjb.31 - gsmtp

Fin envio

**Resultado en la cuenta de correo de destino:**

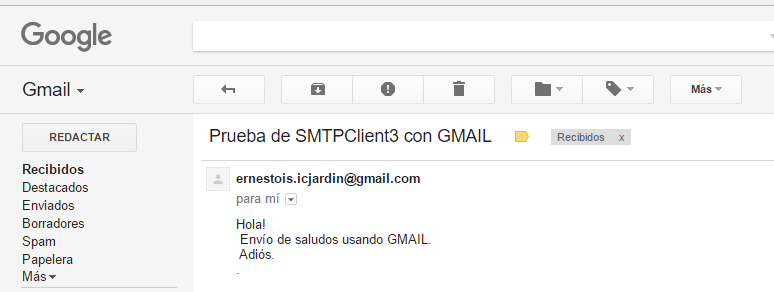


Figura 5.3 – Mensaje recibido en la cuenta de correo de destino

* 1. Uso de Java para comunicar con un servidor POP3

En el correo electrónico se usan otros protocolos, además de SMTP, para funciones adicionales. Entre los más utilizados están los siguientes:

* **MIME** (Multipurpose Internet Mail Extensions): define una serie de especificaciones para expandir las capacidades limitadas del correo electrónico y, en particular, para permitir la inserción de archivos (imágenes, sonido y texto) en un mensaje.
* **IMAP** (Internet Message Access Protocol): permite acceder a los mensajes de correo electrónico almacenados en los servidores SMTP. Permite que los usuarios accedan a su correo desde cualquier equipo que tenga una conexión a Internet. Tiene alguna ventaja con respecto al protocolo POP; por ejemplo: los mensajes continúan siempre almacenados en su servidor (cosa que no ocurre con POP) y los usuarios pueden organizar los mensajes en carpetas.
* **POP** (Post Office Protocol): proporciona acceso a los mensajes de los servidores SMTP. En general, cuando se hace referencia al término POP, se hace referencia a POP3, que es la última versión.

El servidor POP3 es el **servidor de correo entrante** y en general usa el puerto 110. Al igual que SMTP funciona con comandos de texto.

Apache Commons Net proporciona varias clases para acceder a los servidores POP3:

* **POP3Client**: implementa el lado del cliente del protocolo POP3 de internet definido en el RFC 1939.
* **POP3SClient**: POP3 con soporte SSL, extiende POP3Client.
* **POP3MessageInfo**: se utiliza para devolver información acerca de los mensajes almacenados en el servidor POP3. Sus campos (identifier, number y size) se utilizan para referirse a cosas ligeramente diferentes, dependiendo de la información que se devuelve:
  + En respuesta a un comando de estado, *number* contiene el número de mensajes en el buzón de correo, *size* contiene el tamaño del buzón de correo en bytes y el campo *identifier* es nulo
  + En respuesta a una lista de mensajes, *number* contiene el número de mensaje, *size* contiene el tamaño del mensaje en bytes, e *identifier* es nulo
  + En respuesta a una lista de un único mensaje, *number* contiene el número de mensaje, *size* no está definido, e *identifier* contiene el identificador único de mensaje

**Constructores de la clase POP3Client:** sólo tiene uno y se llama POP3Client();

Algunos **constructores de la clase POP3SClient** son los que se muestran a continuación:

|  |  |
| --- | --- |
| **CONSTRUCTOR** | **Función** |
| POP3SClient() | Conexión en modo **explícito**. La negociación SSL/TLS comienza cuando el usuario llama al método execTLS, y el servidor acepta el comando. |
| POP3SClient(boolean implicit) | Conexión en modo **implícito**. La negociación SSL/TLS comienza justo después de que se haya establecido la conexión. |

Los **métodos** de las clases POP3Client y POP3SClient son idénticos. Estos son algunos de ellos:

|  |  |
| --- | --- |
| **MÉTODO** | **Función** |
| boolean deleteMessage(int messageId) | Elimina el mensaje con un número messageId del servidor POP3. Devuelve true si la operación se reañozó correctamente. |
| POP3MessageInfo listMessage(int messageId) | Lista el mensaje indicado en el parámetro messageId |
| POP3MessageInfo[] listMessages) | Obtiene un array con información de todos los mensajes |
| POP3MessageInfo listUniqueIdentifier(int messageId) | Obtiene la lista de un único mensaje |
| boolean login(String username, String password) | Inicia la sesión en el servidor POP3 enviando el nombre de usuario y la clave. Devuelve true si la operación se realizó correctamente. |
| boolean logout(); | Finaliza la sesión con el servidor POP3. Devuelve true si la operación se realizó correctamente. |
| Reader retrieveMessageTop (int messageId) | Recupera el mensaje con número messageId del servidor POP3. |
| Reader retrieveMessageTop (int messageId, int numLines) | Igual que el anterior, pero sólo el número de líneas especificado en el parámetro segundo. Si el segundo parámetro es 0, se recupera la cabecera del mensaje. |

**Ejemplo 9 – Conectarse al servidor POP3 local a través del puerto 110, para visualizar el número de mensajes que tenga el usuario *pepe* con password *pepe* (para que el usuario pepe del servidor de correo Argosoft tenga algún mensaje que visualizar, se puede adaptar el programa indicado en el Ejemplo 6).**

**import** java.io.IOException;

**import** java.net.SocketException;

**import** org.apache.commons.net.pop3.POP3Client;

**import** org.apache.commons.net.pop3.POP3MessageInfo;

**public** **class** ClientePOP3 {

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

String server = "172.20.105.120", username = "pepe", password="pepe";

**int** puerto = 110;

POP3Client pop3 = **new** POP3Client ();

**try** {

// Conexión con el servidor

pop3.connect(server,puerto);

System.***out***.print("Conexión realizada al servidor POP3 " + server);

// Inicio de la sesión

**if** (!pop3.login(username, password))

System.***err***.println("Error al hacer el login");

**else** {

// Obtención de todos los mensajes en un array

POP3MessageInfo[] mensajes = pop3.listMessages();

**if** (mensajes == **null**)

System.***out***.println("Imposible recuperar mensajes");

**else**

System.***out***.println("\nNumero de mensajes "+mensajes.length);

**for** (**int** i=0; i < mensajes.length; i++) {

System.***out***.println("Mensaje: "+(i+1));

POP3MessageInfo msginfo = mensajes[i]; // lista de mensajes

System.***out***.println("IDentificador:"+msginfo.identifier+", Numero:" + msginfo.number + ",Tamaño: " + msginfo.size);

System.***out***.println("Prueba de listUniqueIdenifier:");

POP3MessageInfo pmi = pop3.listUniqueIdentifier(i+1); // Un mensaje

System.***out***.println("IDentificador:"+pmi.identifier+", Numero:" + pmi.number + ",Tamaño: " + pmi.size);

}

// Finalizar la sesión

pop3.logout();

}

} **catch** (SocketException e) {

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

e.printStackTrace();

System.*exit*(1);

} **catch** (IOException e) {

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

e.printStackTrace();

System.*exit*(1);

}

System.*exit*(0);

}

}

**Resultado (el servidor POP3 tiene 3 mensajes del usuario *pepe*):**

Conexión realizada al servidor POP3 172.20.105.120

Numero de mensajes 3

Mensaje: 1

IDentificador:null, Numero:1,Tamaño: 259

Prueba de listUniqueIdenifier:

IDentificador:jy558boe8a4kx7ap, Numero:1,Tamaño: -1

Mensaje: 2

IDentificador:null, Numero:2,Tamaño: 259

Prueba de listUniqueIdenifier:

IDentificador:q1k0jvw5t1e8lk0q, Numero:2,Tamaño: -1

Mensaje: 3

IDentificador:null, Numero:3,Tamaño: 259

Prueba de listUniqueIdenifier:

IDentificador:5e6jwp32ba80hz4b, Numero:3,Tamaño: -1

Para acceder a los **servidores POP3 seguros** (Gmail, por ejemplo) es preciso utilizar el segundo de los constructores (modo implícito):

POP3SClient pop3 = **new** POP3SClient (**true**);

**Ejemplo 10 – Conectarse al servidor POP3 de Gmail (pop.gmail.com) a través del puerto 995, para visualizar el número de mensajes existentes, la cabecera del primer mensaje y su contenido**

El método *RetrieveMessageTop()* obtiene la información de la cabecera, mientras que *RetrieveMessage()* obtiene la información de la cabecera y del mensaje.

**NOTA importante:** cada vez que se lee un mensaje del servidor POP3 con el siguiente programa, en la siguiente lectura el mensaje aparece como si ya no existiese en el servidor.

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.net.SocketException;

**import** org.apache.commons.net.pop3.POP3MessageInfo;

**import** org.apache.commons.net.pop3.POP3SClient;

**public** **class** ClientePOP3 {

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

String server = "pop.gmail.com", username = "ernestois.icjardin@gmail.com", password="PONER\_AQUI\_CONTRASEÑA";

**int** puerto = 995;

POP3SClient pop3 = **new** POP3SClient (**true**);

**try** {

// Conexión con el servidor

pop3.connect(server,puerto);

System.***out***.print("Conexión realizada al servidor POP3 " + server);

// Inicio de la sesión

**if** (!pop3.login(username, password))

System.***err***.println("Error al hacer el login");

**else** {

// Obtención de todos los mensajes en un array

POP3MessageInfo[] mensajes = pop3.listMessages();

**if** (mensajes == **null**)

System.***out***.println("Imposible recuperar mensajes");

**else**

System.***out***.println("\nNumero de mensajes "+mensajes.length);

/// Solo se muestra la información del mensaje indicada en la variable

**int** Mensaje=0;

System.***out***.println("Mensaje: "+(Mensaje));

POP3MessageInfo msginfo = mensajes[Mensaje]; // lista de mensajes

System.***out***.println("IDentificador:"+msginfo.identifier+", Numero:" + msginfo.number + ",Tamaño: " + msginfo.size);

System.***out***.println("Prueba de listUniqueIdenifier:");

POP3MessageInfo pmi = pop3.listUniqueIdentifier(Mensaje+1); // Un mensaje

System.***out***.println("IDentificador:"+pmi.identifier+", Numero:" + pmi.number + ",Tamaño: " + pmi.size);

// Visualizar la cabecera

System.***out***.println("\nCabecera del mensaje:");

BufferedReader reader = (BufferedReader) pop3.retrieveMessageTop(msginfo.number, Mensaje);

String linea;

**while** ((linea=reader.readLine()) !=**null**)

System.***out***.println(linea.toString());

reader.close();

// Obtención del contenido del mensaje

System.***out***.println("\nContenido del mensaje:");

BufferedReader readermensaje = (BufferedReader) pop3.retrieveMessage(msginfo.number);

String lineamensaje;

**while** ((lineamensaje=readermensaje.readLine()) !=**null**)

System.***out***.println(lineamensaje.toString());

readermensaje.close();

}

/// Finalizar la sesión

pop3.logout();

} **catch** (SocketException e) {

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

e.printStackTrace();

System.*exit*(1);

} **catch** (IOException e) {

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

e.printStackTrace();

System.*exit*(1);

}

System.*exit*(0);

}

}

**Resultado:**

Conexión realizada al servidor POP3 pop.gmail.com

Numero de mensajes 2

Mensaje: 0

IDentificador:null, Numero:1,Tamaño: 4743

Prueba de listUniqueIdenifier:

IDentificador:GmailId158915acc81219f2, Numero:1,Tamaño: -1

**Cabecera del mensaje:**

Delivered-To: ernestois.icjardin@gmail.com

Received: by 10.31.242.5 with SMTP id q5csp2786578vkh;

Wed, 23 Nov 2016 05:23:16 -0800 (PST)

X-Received: by 10.176.6.233 with SMTP id g96mr2082234uag.97.1479907396731;

Wed, 23 Nov 2016 05:23:16 -0800 (PST)

Return-Path: <dm2prosciudadjardin@gmail.com>

Received: from mail-ua0-x231.google.com (mail-ua0-x231.google.com. [2607:f8b0:400c:c08::231])

by mx.google.com with ESMTPS id w5si6709870uaw.74.2016.11.23.05.23.15

for <ernestois.icjardin@gmail.com>

(version=TLS1\_2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);

Wed, 23 Nov 2016 05:23:16 -0800 (PST)

Received-SPF: pass (google.com: domain of dm2prosciudadjardin@gmail.com designates 2607:f8b0:400c:c08::231 as permitted sender) client-ip=2607:f8b0:400c:c08::231;

Authentication-Results: mx.google.com;

dkim=pass header.i=@gmail.com;

spf=pass (google.com: domain of dm2prosciudadjardin@gmail.com designates 2607:f8b0:400c:c08::231 as permitted sender) smtp.mailfrom=dm2prosciudadjardin@gmail.com;

dmarc=pass (p=NONE dis=NONE) header.from=gmail.com

Received: by mail-ua0-x231.google.com with SMTP id b35so3975uaa.3

for <ernestois.icjardin@gmail.com>; Wed, 23 Nov 2016 05:23:15 -0800 (PST)

DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;

d=gmail.com; s=20120113;

h=mime-version:from:date:message-id:subject:to;

bh=iP+PTypu7QnXvQq6nH0cL3ZGKUY+2CcUavaHlJGtQIw=;

b=y4cEBlLI8qm6oNPzcWSphQgHDY3MQVtQ4n1j6A8Fjxiact8anDsM48jTy37YDRBiwj

3mz3R+TiRxQgmvgUlTPMExJXjcq1K4Xeg0meUNtylP6Fe7sITAPL7hjSWtod89YBMmq7

a/NZ5Ap3B4OlNGa/7+sAHKHaaVjZSE/0lxU0Ggz8R9nbUt3AUb59D9D9TeA4bshD9+6g

mupzGtEMlpjLokGldeTAvix/F47OG/yS+spxZBWHipjv+2GBw1G5h38wcf0ecdmrLBsk

da7Eln8JvuoDPeINsqTptkqelWQiorabELM7hBxUtSpj32XjsOUTvLRf+wo+nlrXH5OX

VCUA==

X-Google-DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;

d=1e100.net; s=20130820;

h=x-gm-message-state:mime-version:from:date:message-id:subject:to;

bh=iP+PTypu7QnXvQq6nH0cL3ZGKUY+2CcUavaHlJGtQIw=;

b=WcpR59nLqj++jICtZLI9vCGgqZ1T41/pZhGk9sqXkPh+u4MCJTa4FQI8HjNFB75rtb

dQsKzJalz7KQtsK46/DC5uaIfdyHlHzMsyeRT0e/6rvDj1XWY+RxNWugl/rhT2j/kiC2

z16IFLyaGRpnlSN3e4E6iaSxBd/nv2XPzt5svVrhdkaT9uM+Y/hoYZl9mNi4lC6/LZtI

fys41oLvMXeaWNbvMJpSp9s7VEnsiiYEi47HovQ1q7d8cgOrB075TI3/Zo/R7UBFghR8

t6P3RU1am+xiM8r3B4hbyNaMA2+H5ottcqRgqc6ikEdfUd6OCTG+mYVX5xZ1mn0Mvs2p

8egQ==

X-Gm-Message-State: AKaTC02OYs0vcELnxs4BrpEM+ePlFcslstoDgU8wIU4/UsoRytlMdKX3XoJDB8+P2McZCrHvycDn460twFVl7w==

X-Received: by 10.159.38.41 with SMTP id 38mr1679386uag.26.1479907395725; Wed,

23 Nov 2016 05:23:15 -0800 (PST)

MIME-Version: 1.0

Received: by 10.31.6.208 with HTTP; Wed, 23 Nov 2016 05:23:15 -0800 (PST)

From: DM2 PROS <dm2prosciudadjardin@gmail.com>

Date: Wed, 23 Nov 2016 14:23:15 +0100

Message-ID: <CA+Pitd0PNmpfJUii1Yq09XHD6PFCf4hduxzPrXEOss5hChHFTQ@mail.gmail.com>

Subject: =?UTF-8?Q?PROS\_=2D\_Unidad\_Did=C3=A1ctica\_4?=

To: ernestois.icjardin@gmail.com

Content-Type: multipart/alternative; boundary=001a113e23dc9f6c480541f7ca55

**Contenido del mensaje:**

Delivered-To: ernestois.icjardin@gmail.com

Received: by 10.31.242.5 with SMTP id q5csp2786578vkh;

Wed, 23 Nov 2016 05:23:16 -0800 (PST)

X-Received: by 10.176.6.233 with SMTP id g96mr2082234uag.97.1479907396731;

Wed, 23 Nov 2016 05:23:16 -0800 (PST)

Return-Path: <dm2prosciudadjardin@gmail.com>

Received: from mail-ua0-x231.google.com (mail-ua0-x231.google.com. [2607:f8b0:400c:c08::231])

by mx.google.com with ESMTPS id w5si6709870uaw.74.2016.11.23.05.23.15

for <ernestois.icjardin@gmail.com>

(version=TLS1\_2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);

Wed, 23 Nov 2016 05:23:16 -0800 (PST)

Received-SPF: pass (google.com: domain of dm2prosciudadjardin@gmail.com designates 2607:f8b0:400c:c08::231 as permitted sender) client-ip=2607:f8b0:400c:c08::231;

Authentication-Results: mx.google.com;

dkim=pass header.i=@gmail.com;

spf=pass (google.com: domain of dm2prosciudadjardin@gmail.com designates 2607:f8b0:400c:c08::231 as permitted sender) smtp.mailfrom=dm2prosciudadjardin@gmail.com;

dmarc=pass (p=NONE dis=NONE) header.from=gmail.com

Received: by mail-ua0-x231.google.com with SMTP id b35so3975uaa.3

for <ernestois.icjardin@gmail.com>; Wed, 23 Nov 2016 05:23:15 -0800 (PST)

DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;

d=gmail.com; s=20120113;

h=mime-version:from:date:message-id:subject:to;

bh=iP+PTypu7QnXvQq6nH0cL3ZGKUY+2CcUavaHlJGtQIw=;

b=y4cEBlLI8qm6oNPzcWSphQgHDY3MQVtQ4n1j6A8Fjxiact8anDsM48jTy37YDRBiwj

3mz3R+TiRxQgmvgUlTPMExJXjcq1K4Xeg0meUNtylP6Fe7sITAPL7hjSWtod89YBMmq7

a/NZ5Ap3B4OlNGa/7+sAHKHaaVjZSE/0lxU0Ggz8R9nbUt3AUb59D9D9TeA4bshD9+6g

mupzGtEMlpjLokGldeTAvix/F47OG/yS+spxZBWHipjv+2GBw1G5h38wcf0ecdmrLBsk

da7Eln8JvuoDPeINsqTptkqelWQiorabELM7hBxUtSpj32XjsOUTvLRf+wo+nlrXH5OX

VCUA==

X-Google-DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;

d=1e100.net; s=20130820;

h=x-gm-message-state:mime-version:from:date:message-id:subject:to;

bh=iP+PTypu7QnXvQq6nH0cL3ZGKUY+2CcUavaHlJGtQIw=;

b=WcpR59nLqj++jICtZLI9vCGgqZ1T41/pZhGk9sqXkPh+u4MCJTa4FQI8HjNFB75rtb

dQsKzJalz7KQtsK46/DC5uaIfdyHlHzMsyeRT0e/6rvDj1XWY+RxNWugl/rhT2j/kiC2

z16IFLyaGRpnlSN3e4E6iaSxBd/nv2XPzt5svVrhdkaT9uM+Y/hoYZl9mNi4lC6/LZtI

fys41oLvMXeaWNbvMJpSp9s7VEnsiiYEi47HovQ1q7d8cgOrB075TI3/Zo/R7UBFghR8

t6P3RU1am+xiM8r3B4hbyNaMA2+H5ottcqRgqc6ikEdfUd6OCTG+mYVX5xZ1mn0Mvs2p

8egQ==

X-Gm-Message-State: AKaTC02OYs0vcELnxs4BrpEM+ePlFcslstoDgU8wIU4/UsoRytlMdKX3XoJDB8+P2McZCrHvycDn460twFVl7w==

X-Received: by 10.159.38.41 with SMTP id 38mr1679386uag.26.1479907395725; Wed,

23 Nov 2016 05:23:15 -0800 (PST)

MIME-Version: 1.0

Received: by 10.31.6.208 with HTTP; Wed, 23 Nov 2016 05:23:15 -0800 (PST)

From: DM2 PROS <dm2prosciudadjardin@gmail.com>

Date: Wed, 23 Nov 2016 14:23:15 +0100

Message-ID: <CA+Pitd0PNmpfJUii1Yq09XHD6PFCf4hduxzPrXEOss5hChHFTQ@mail.gmail.com>

Subject: =?UTF-8?Q?PROS\_=2D\_Unidad\_Did=C3=A1ctica\_4?=

To: ernestois.icjardin@gmail.com

Content-Type: multipart/alternative; boundary=001a113e23dc9f6c480541f7ca55

--001a113e23dc9f6c480541f7ca55

Content-Type: text/plain; charset=UTF-8

Content-Transfer-Encoding: quoted-printable

Hola Ernesto:

=C2=BFHas recibido este mensaje correspondiente a la UD4 del m=C3=B3dulo PR=

OS?

Ya me dir=C3=A1s.

Saludos.

1. PROGRAMACIÓN DE SERVIDORES CON JAVA

En este punto se explica de forma resumida una forma de construir un servidor de ficheros en Java.

Su misión será proporcionar el acceso a ficheros y carpetas. Cuando se inicia el programa 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 objetos. El funcionamiento general se muestra en la siguiente figura:

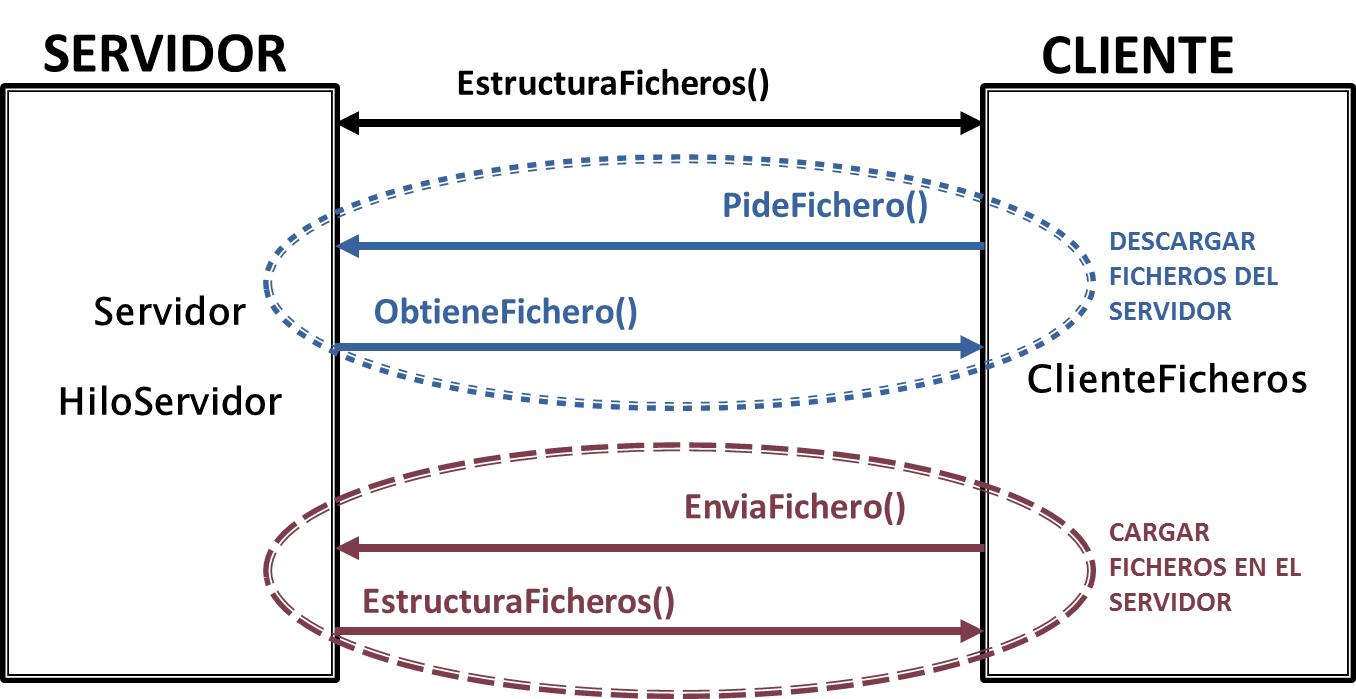


Figura 5.1 – Servidor de ficheros: comunicación entre el cliente y el servidor

El funcionamiento general 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.

La clase **EstucturaFicheros** se utiliza para definir la estructura del directorio seleccionado con sus ficheros y directorios. Tiene varios atributos y dos constructores con varios métodos de tipo get.

* 1. Programa servidor

El programa servidor, clase **Servidor**, es muy básico:

* Se define el puerto por el que va a quedar en escucha.
* Se elige el directorio con JFileChooser.
* Se inicia el servidor.
* Se entra en un proceso repetitivo donde se esperan las conexiones de los clientes.
* A cada cliente conectado se le envía el socket creado y un objeto *EstructuraFicheros(Directorio)*.

La siguiente figura muestra un momento de la ejecución del servidor en el que se elige el directorio:

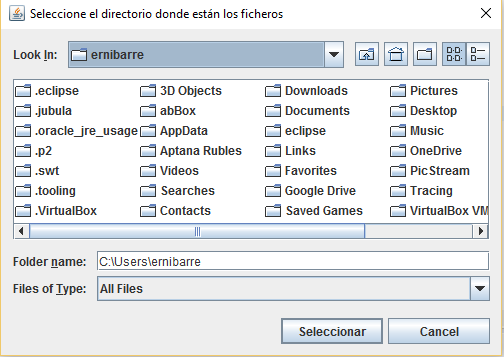


Figura 5.2 – Ejecución del servidor en un sistema Windows

Las operaciones por cada cliente conectado se realizarán en 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 cliente y servidor se realizarán en el método *run()* del hilo.

* 1. Programa Cliente

El programa cliente, clase **ClienteFicheros**, hereda de JFrame e implementa Runnable. En el método *main* se crea un socket que se va a conectar con el servidor a través del puerto pactado.

Al ejecutarse el cliente se muestra una pantalla con un JList donde aparecerá el contenido de los ficheros y directorios del directorio seleccionado en el servidor. Se muestran los botones Subir Fichero, Descargar Fichero y Salir:

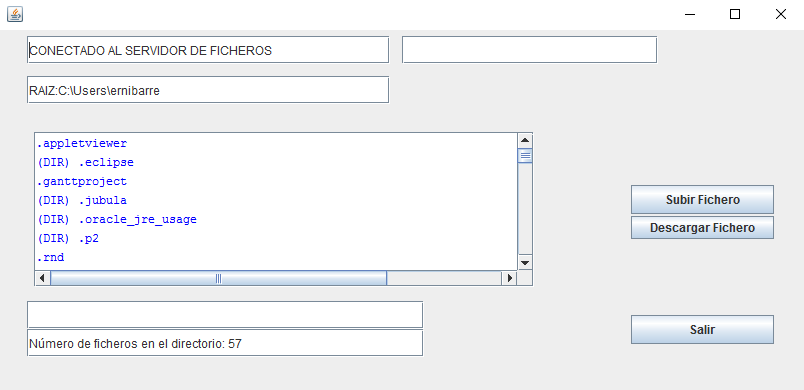


Figura 5.3 – Ejecución del cliente en un sistema Windows

1. DISPONIBILIDAD DEL SERVICIO
   1. Requisitos

Es importante que las organizaciones cuantifiquen los requisitos de disponibilidad de los servicios en red para la correcta elaboración de los Acuerdos de Nivel de Servicio (SLA) con aquellos proveedores (internos o externos) que se los proporcionen.

La disponibilidad propuesta debe encontrase en línea tanto con los necesidades reales de la organización como con las posibilidades que la organización tenga en materia de Tecnologías de la Información.

Aunque en principio todas las organizaciones estarán de acuerdo con unas elevadas cotas de disponibilidad, es importante hacerles ver que una alta disponibilidad puede generar unos costes injustificados dadas sus necesidades reales. Quizás unas pocas horas sin un determinado servicio de red pueden representar poco más allá de una pequeña inconveniencia mientras que la certeza de un servicio prácticamente continuo y sin interrupciones puede requerir la replicación de sistemas u otras medidas igualmente costosas que no van a tener una repercusión real en la rentabilidad de la organización.

Para llevar a cabo eficientemente esta tarea es necesario tener en cuenta lo siguiente:

* Identificar las actividades clave del negocio de la organización.
* Cuantificar los intervalos razonables de interrupción de los diferentes servicios en red dependiendo de sus respectivos impactos.
* Determinar las franjas horarias de disponibilidad de los servicios en red (24/7, 12/5, ...).
  1. Calidad del servicio

La monitorización de la disponibilidad del servicio y la elaboración de los informes correspondientes son dos actividades orientadas a medir la Calidad del servicio.

Desde el momento de la interrupción del servicio en red hasta su restitución o "tiempo de parada", el incidente pasa por distintas fases que deben ser individualizadamente analizadas:

* **Tiempo de detección**: es el tiempo que transcurre desde que ocurre el fallo hasta que la organización TI tiene constancia del mismo.
* **Tiempo de respuesta**: es el tiempo que transcurre desde la detección del problema hasta que se realiza un registro y diagnóstico del incidente.
* **Tiempo de reparación/recuperación**: periodo de tiempo utilizado para reparar el fallo o encontrar un "workaround" o solución temporal al mismo y devolver el sistema a la situación anterior a la interrupción del servicio.

Es importante determinar **métricas** que permitan medir con precisión las diferentes fases del ciclo de vida de la interrupción del servicio. La organización debe conocer estas métricas y dar su conformidad a las mismas para evitar malentendidos. En algunos casos es difícil determinar si el sistema está "caído o en funcionamiento" y la interpretación puede diferir entre proveedores y clientes, por lo tanto, estas métricas deben de poder expresarse en términos que el cliente pueda entender.

Algunos de los parámetros que se suelen utilizar para medir la disponibilidad del servicio son los siguientes:

* **Uptime:** periodo total durante el cual un determinado servicio está operativo dentro de los tiempos de servicio acordados.
* **Downtime:** periodo total durante el cual un determinado servicio no está operativo dentro de los tiempos de servicio acordados.
* **Mean Time Between Service Incidents (MTBSI):** es el cociente entre el tiempo en que el servicio está disponible y el número de caídas del servicio.
* **Mean Time to Restore Services (MTRS):** es el cociente entre el tiempo en que el servicio no está disponible y el número de caídas del servicio.
* **Mean Time Between Failures (MTBF):** es el cociente entre la diferencia entre el tiempo en que el servicio está disponible y el tiempo en que el servicio no está disponible, y el número de caídas del servicio.

En la siguiente figura se muestra la evolución de un servicio entre caída y caída:

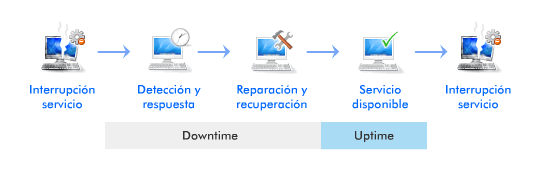


Figura 6.1 – Evolución de un servicio entre dos interrupciones del mismo

* 1. Denegación de servicio

En [seguridad informática](https://es.wikipedia.org/wiki/Seguridad_informática), un **ataque de denegación de servicio**, también llamado ataque **DoS** (por sus siglas en inglés), es un ataque a un sistema de ordenadores o [red](https://es.wikipedia.org/wiki/Red_de_computadoras) que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. Normalmente provoca la pérdida de la conectividad con la red por el consumo del [ancho de banda](https://es.wikipedia.org/wiki/Ancho_de_banda) de la red de la víctima o sobrecarga de los recursos computacionales del sistema atacado.

Los ataques DoS se generan mediante la saturación de los puertos con múltiples flujos de información, haciendo que el [servidor](https://es.wikipedia.org/wiki/Servidor) se sobrecargue y no pueda seguir prestando su servicio. Por eso se le denomina denegación, pues hace que el servidor no pueda atender a la cantidad enorme de solicitudes. Esta técnica es usada por los [*crackers*](https://es.wikipedia.org/wiki/Cracker) o hackers para dejar fuera de servicio servidores objetivo.

Una ampliación del ataque DoS es el llamado **ataque de denegación de servicio distribuido** (**DDoS** por sus siglas en inglés) el cual se lleva a cabo generando un gran flujo de información desde varios puntos de conexión. La forma más común de realizar un DDoS es a través de una [red de bots](https://es.wikipedia.org/wiki/Botnet), siendo esta técnica el [ciberataque](https://es.wikipedia.org/wiki/Ciberataque" \t "Ciberataque) más usual y eficaz por su sencillez tecnológica.

En ocasiones, esta herramienta ha sido utilizada como un buen método para comprobar la capacidad de tráfico que un ordenador puede soportar sin volverse inestable y afectar a los servicios que presta. Un administrador de redes puede así conocer la capacidad real de cada máquina.