

# UNIDAD 4

## Generación de Servicios en Red

---

|          |                                                         |           |
|----------|---------------------------------------------------------|-----------|
| <b>1</b> | <b>INTRODUCCIÓN .....</b>                               | <b>2</b>  |
| <b>2</b> | <b>PROTOCOLOS ESTÁNDAR DE COMUNICACIÓN EN RED .....</b> | <b>2</b>  |
| <b>3</b> | <b>COMUNICACIÓN CON UN SERVIDOR FTP .....</b>           | <b>3</b>  |
| 3.1      | COMUNICACIÓN DESDE JAVA CON UN SERVIDOR FTP .....       | 4         |
|          | PRÁCTICA 1 .....                                        | 8         |
| 3.2      | SUBIR FICHEROS AL SERVIDOR .....                        | 8         |
| 3.3      | DESCARGAR FICHEROS DEL SERVIDOR .....                   | 10        |
| 3.4      | CREACIÓN DE UN CLIENTE FTP .....                        | 10        |
|          | PRÁCTICA 2 .....                                        | 11        |
| <b>4</b> | <b>COMUNICACIÓN CON UN SERVIDOR TELNET .....</b>        | <b>11</b> |
| 4.1      | INSTALACIÓN Y USO DE UN SERVIDOR TELNET .....           | 11        |
| 4.2      | COMUNICACIÓN DESDE JAVA CON UN SERVIDOR TELNET .....    | 12        |
| <b>5</b> | <b>COMUNICACIÓN CON UN SERVIDOR SMTP .....</b>          | <b>15</b> |
| 5.1      | INSTALACIÓN DE UN SERVIDOR DE CORREO ELECTRÓNICO .....  | 16        |
| 5.2      | USO DE TELNET PARA COMUNICAR CON EL SERVIDOR SMTP ..... | 16        |
| 5.3      | COMUNICACIÓN DESDE JAVA CON UN SERVIDOR SMTP .....      | 18        |
|          | PRÁCTICA 3 .....                                        | 25        |
| 5.4      | ACCESO A LOS MENSAJES DE UN SERVIDOR POP3 .....         | 25        |
| 5.5      | USO DE TELNET PARA COMUNICAR CON EL SERVIDOR POP .....  | 26        |
| 5.6      | COMUNICACIÓN DESDE JAVA CON UN SERVIDOR POP3 .....      | 27        |
|          | PRÁCTICA 4 .....                                        | 30        |

## 1 INTRODUCCIÓN

Los servicios son programas auxiliares utilizados en un sistema de computadoras para gestionar una colección de recursos y prestar su funcionalidad a los usuarios y aplicaciones. Por ejemplo, cuando enviamos un documento a una impresora que está formando parte de una red estamos usando un servicio de impresión, este servicio permite gestionar y compartir la impresora en la red. El único acceso que tenemos 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 este capítulo estudiaremos estos servicios y usaremos Java para programar clientes de los servicios de Internet que se usan más frecuentemente.

## 2 PROTOCOLOS ESTÁNDAR DE COMUNICACIÓN EN RED

El modelo TCP/IP está compuesto por cuatro capas o niveles. La capa de aplicación maneja protocolos de alto nivel que implementan servicios 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

| Capas TCP/IP    | Protocolos TCP/IP       |                |
|-----------------|-------------------------|----------------|
| Aplicación      | SMTP, Telnet, FTP, HTTP | NFS, SNMP, DNS |
| Transporte      | TCP                     | UDP            |
| Internet        | IP                      |                |
| Interfaz de Red |                         |                |

Figura 1. Capas y protocolos de la arquitectura TCP/IP

Todas las aplicaciones que implementan TCP/IP se basan en el modelo cliente-servidor.

**TELNET** (*Telecommunication NetWork*). Emulación de terminal; permite a un usuario acceder a una máquina remota y manejarla como si hubiese iniciado una sesión local. Su principal problema es la seguridad ya que los nombres de usuario y contraseñas viajan por la red como texto plano.

**SMTP** (*Simple Mail Transfer Protocol*). Protocolo simple de transferencia de correo electrónico; es probablemente el servicio más popular entre los usuarios de la red. 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 Transfer Protocol*). Protocolo de transferencia de ficheros; es un servicio confiable orientado a conexión que se utiliza para transferir ficheros de una máquina a otra a través de Internet. Los sitios FTP son lugares desde los que podemos descargar o a los que podemos subir ficheros.

**TFTP** (*Trivial File Transfer Protocol*). Protocolo trivial de transferencia de ficheros; es un protocolo de transferencia muy simple semejante a una versión básica de FTP. Fue definido para aplicaciones que no necesitan tanta interacción entre cliente y servidor. A menudo se utiliza para transferir ficheros entre ordenadores en una red en los 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; utilizado por los navegadores web para realizar peticiones a los servidores web y para recibir las respuestas de ellos. Es un protocolo que especifica los mensajes involucrados en un intercambio petición-respuesta, los métodos, argumentos y resultados y las reglas para representar todo ello en los mensajes.

**NFS** (*Network File System*). Sistema de ficheros de red, ha sido desarrollado por Sun Microsystems y 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 este fuera un fichero local.

**SNMP** (*Simple Network Management Protocol*). Protocolo simple de administración de red, es un protocolo utilizado para intercambiar información de gestión entre los dispositivos de una red. Permite a los administradores monitorear, controlar y supervisar el funcionamiento de la red.

**DNS** (*Domain Name System*). Sistema de nombres de dominio, es un 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 manera no hay que recordar y usar su dirección IP.

En este capítulo aprenderemos a crear clientes para acceder a diferentes servicios TCP/IP.

### 3 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 habitual de publicación en Internet.

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

Hay dos tipos fundamentales de acceso a través de FTP:

- **Acceso anónimo:** cuando la conexión con la máquina servidora la realiza un usuario sin autenticar y sin ningún tipo de privilegio 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 autenticado, el usuario es recluido a su directorio personal donde puede subir y bajar ficheros; normalmente se le asigna una cuota de espacio.

FTP utiliza dos conexiones TCP distintas, una conexión de control y otra 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, esta existe únicamente cuando hay datos a transmitir.

Cuando un cliente se conecta a un servidor FTP, el cliente emplea un puerto aleatorio pero el servidor se conecta en el puerto 21. Para la transferencia de datos no se utilizan los mismos puertos, el cliente obtiene un nuevo puerto y el servidor en el proceso de transferencia de datos usa en el puerto 20.

### 3.1 Comunicación desde Java con un servidor FTP

Existen librerías Java que nos permiten crear programas cliente para comunicar con un servidor FTP. **Apache Commons Net™** proporciona una librería de componentes que nos permite implementar el lado cliente de muchos protocolos básicos de Internet. La librería incluye soporte para protocolos como FTP, SMTP, Telnet, TFTP, etc.

A continuación vamos a ver cómo acceder desde un programa cliente Java, a un servidor FTP, podremos conectarnos, listar los ficheros y directorios, subir ficheros, eliminarlos, etc. Necesitaremos la librería `commons-net-x.y.jar` que se puede descargar desde la URL de **Apache Commons** [http://commons.apache.org/proper/commons-net/download\\_net.cgi](http://commons.apache.org/proper/commons-net/download_net.cgi); para los ejemplos se ha utilizado la versión 3.6.

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 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 finalice el proceso, cerrar la conexión usando el método `disconnect`. En el siguiente ejemplo realizamos una conexión a un servidor FTP (<ftp.rediris.es>), comprobamos si se ha realizado correctamente o no y cerramos la conexión:

```
import java.io.IOException;
import java.net.SocketException;
import org.apache.commons.net.ftp.FTPClient;
import org.apache.commons.net.ftp.FTPReply;

public class ClienteFTP1 {
    public static void main(String[] args) throws SocketException, IOException {
        FTPClient cliente = new FTPClient();
        String servidor = "ftp.rediris.es";
        System.out.println("Nos conectamos a: " + servidor);
        cliente.connect(servidor);
        /* respuesta del servidor FTP */
        System.out.print(cliente.getReplyString());
        /* código de respuesta */
        int respuesta = cliente.getReplyCode();
        /* comprobación del código 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.println("Conexión finalizada.");
    }
}
```

La ejecución visualiza la siguiente información:

```
Nos conectamos a: ftp.rediris.es
220- Bienvenido al servicio de replicas de RedIRIS.
220- Welcome to the RedIRIS mirror service.
220 Only anonymous FTP is allowed here
Conexión finalizada.
```

La clase `FTPReply` almacena un conjunto de constantes para códigos de respuesta FTP. Para interpretar el significado de los códigos se puede consultar el RFC 959 (<http://www.ietf.org/rfc/rfc959.txt>). Los nombres nemónicos usados para las constantes son transcripciones de las descripciones de los códigos de la RFC 959. El método `isPositiveCompletion(int respuesta)` devuelve `true` si un código de respuesta ha terminado positivamente. El código 220 significa que el servicio está preparado.

Sabías que...

El protocolo FTP usa un esquema de códigos de respuesta donde cada uno de sus dígitos tiene un significado especial. Son números de tres dígitos en ASCII, el primer dígito indica si la respuesta es buena, mala o incompleta:

- 1yz Respuesta preliminar positiva, el servidor inició la acción solicitada.
- 2yz Respuesta de terminación positiva, el servidor terminó con éxito la acción solicitada.
- 3yz Respuesta intermedia positiva, el servidor aceptó el comando pero la acción solicitada necesita más información.
- 4yz Respuesta de terminación negativa transitoria, el servidor no aceptó el comando y la acción solicitada no ocurrió.
- 5yz Respuesta de terminación negativa permanente, el servidor no aceptó el comando y la acción solicitada no ocurrió.

En la siguiente tabla se muestran algunos métodos de la clase FTP usados anteriormente:

|                                        |                                                                              |
|----------------------------------------|------------------------------------------------------------------------------|
| <code>void connect(String host)</code> | Abre la conexión con el servidor FTP indicado en host.                       |
| <code>int getReplyCode()</code>        | Devuelve el valor entero del código de respuesta de la última respuesta FTP. |
| <code>String getReplyString()</code>   | Devuelve el texto completo de la respuesta del servidor FTP.                 |

En la siguiente tabla se muestran algunos métodos de la clase FTPClient (derivada de FTP) que utilizaremos para iniciar sesión en el servidor FTP, subir, bajar y eliminar ficheros, movemos de un directorio a otro, etc. Muchos de estos métodos devuelven un valor booleano, `true` si el método tuvo éxito y `false` en caso contrario:

|                                                         |                                                                                                                                                                                             |
|---------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>void disconnect()</code>                          | Cierra la conexión con el servidor FTP y restaura los parámetros de conexión a los valores predeterminados                                                                                  |
| <code>boolean login (String user, String passwd)</code> | Inicia sesión en el servidor FTP usando el nombre de usuario y la contraseña proporcionados. Devuelve <code>true</code> si se inicia la sesión con éxito, si no devuelve <code>false</code> |
| <code>boolean logout()</code>                           | Sale del servidor FTP.                                                                                                                                                                      |
| <code>String printWorkingDirectory()</code>             | Devuelve el nombre de ruta del directorio de trabajo actual.                                                                                                                                |
| <code>FTPFile[] listFiles()</code>                      | Obtiene una lista de ficheros del directorio actual como un array de objetos <code>FTPFile</code> .                                                                                         |
| <code>FTPFile[] listFiles(String path)</code>           | Obtiene una lista de ficheros del directorio indicado en <code>path</code> .                                                                                                                |
| <code>String[] listNamesO</code>                        | Obtiene una lista de ficheros del directorio actual como un array de cadenas.                                                                                                               |

|                                                                      |                                                                                                                                               |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <code>FTPFile[] listDirectories()</code>                             | Obtiene la lista de los directorios que se encuentran en el directorio de trabajo actual.                                                     |
| <code>FTPFile[] listDirectories(String parent)</code>                | Obtiene la lista de los directorios que se encuentran en el directorio especificado en <code>parent</code> .                                  |
| <code>boolean changeWorkingDirectory(String pathname)</code>         | Cambia el directorio de trabajo actual de la sesión FTP al indicado en <code>pathname</code> .                                                |
| <code>Boolean changeToParentDirectory()</code>                       | Cambia al directorio padre del directorio de trabajo actual.                                                                                  |
| <code>boolean setFileType(int fileType)</code>                       | Establece el tipo de fichero a transferir: <code>ASCII_FILE_TYPE</code> (fichero ASCII), <code>BINARY_FILE_TYPE</code> (imagen binaria), etc. |
| <code>boolean storeFile(String nombre, InputStream local)</code>     | Almacena un fichero en el servidor con el nombre indicado tomando como entrada el <code>InputStream</code> .                                  |
| <code>boolean retrieveFile(String nombre, OutputStream local)</code> | Recupera un fichero del servidor y lo escribe en el <code>OutputStream</code> dado.                                                           |
| <code>boolean deleteFile(String pathname)</code>                     | Elimina un fichero en el servidor FTP.                                                                                                        |
| <code>boolean rename(String antiguo, String nuevo)</code>            | Cambia el nombre de un fichero del servidor FTP.                                                                                              |
| <code>boolean removeDirectory(String pathname)</code>                | Elimina un directorio en el servidor FTP (si está vacío).                                                                                     |
| <code>boolean makeDirectory(String pathname)</code>                  | Crea 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 determinado (generalmente 900 segundos). La clase `FTPClient` detectará un cierre prematuro de la conexión con el servidor FTP y puede lanzar `FTPConnectionClosedException`.

Lo más normal es conectar a un servidor FTP con un nombre de usuario y su clave. Para identificarnos usaremos el método `login` que devuelve `true` si la conexión se realiza correctamente. Para desconectarnos usamos el método `logout`.

En el siguiente ejemplo nos conectamos al servidor FTP [ftp.rediris.es](http://ftp.rediris.es) utilizando un acceso anónimo. Nos conectamos como usuario `anonymous` para mostrar la lista de ficheros del directorio actual. Usamos el método `listFiles` que devuelve un array de la clase `FTPFile` con información de los ficheros y directorios encontrados; recorreremos 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 org.apache.commons.net.ftp.FTPClient;
import org.apache.commons.net.ftp.FTPFile;

public class ClienteFTP2 {
    public static void main(String[] args) {
        FTPClient cliente = new FTPClient();
        String servidor = "ftp.rediris.es";
        System.out.println("Nos conectamos a: " + servidor);
        String usuario = "anonymous";
        String clave = "anonymous";
        try {
```

```

        cliente.connect(servidor);
        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);
        String tipos[] = { "Fichero", "Directorio", "Enlace simbólico" };
        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("Sesión con el servidor FTP finalizada...");
        else
            System.out.println("Error finalizando la sesión...");
        cliente.disconnect();
        System.out.println("Desconectado ...");
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
}

```

La ejecución muestra la siguiente salida:

```

Nos conectamos a: ftp.rediris.es
Login correcto. ..
Directorio actual: /
Ficheros en el directorio actual:7
. => Directorio
.. => Directorio
debian => Enlace simbólico
debian-cd => Enlace simbólico
mirror => Directorio
sites => Directorio
welcome.msg => Fichero
Sesión con el servidor FTP finalizada...
Desconectado ...

```

Para movernos de un directorio a otro usamos el método `changeWorkingDirectory`. Devuelve `true` si el directorio existe, si no existe devuelve `false`. Por ejemplo, para ir al directorio `mirror/MySQL/Downloads` habrá que ejecutar las sentencias siguientes:

```

String directorio="/mirror/MySQL/Downloads/";
if(cliente.changeWorkingDirectory(directorio))
    System.out.println("Directorio actual:" + cliente.printWorkingDirectory());
else
    System.out.println("NO EXISTE EL DIRECTORIO: " + directorio);

```

El siguiente ejemplo crea un directorio en el directorio actual (tenemos que tener permiso para poder hacerlo) y hacemos que sea el directorio de trabajo actual:

```

String directorio="NuevoDirectorio";
if(cliente.makeDirectory (directorio)){
    System.out.println("Directorio creado ....");
    cliente.changeWorkingDirectory (directorio);
}
else
    System.out.println("No se ha podido crear el directorio: " + directorio);

```

La clase `FTPFile` se utiliza para representar información acerca de los ficheros almacenados en un servidor FTP. Algunos métodos importantes son:

|                                       |                                                                                                                                                                                           |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>String getName()</code>         | Devuelve el nombre del fichero.                                                                                                                                                           |
| <code>long getSize()</code>           | Devuelve el tamaño del fichero en bytes.                                                                                                                                                  |
| <code>int getType()</code>            | Devuelve el tipo del fichero, 0 si es un fichero ( <code>FILE_TYPE</code> ), 1 un directorio ( <code>DIRECTORY_TYPE</code> ) y 2 un enlace simbólico ( <code>SYMBOLIC_LINK_TYPE</code> ). |
| <code>String getUser()</code>         | Devuelve el nombre del usuario propietario del fichero.                                                                                                                                   |
| <code>boolean isDirectory()</code>    | Devuelve <code>true</code> si el fichero es un directorio.                                                                                                                                |
| <code>boolean isFile()</code>         | Devuelve <code>true</code> si es un fichero.                                                                                                                                              |
| <code>boolean isSymbolicLink()</code> | Devuelve <code>true</code> si es un enlace simbólico.                                                                                                                                     |

## PRÁCTICA 1

Conéctate a [ftp.rediris.es](http://ftp.rediris.es) y visualiza los directorios del directorio raíz, entra después en cada directorio del directorio raíz mostrando la lista de ficheros y directorios que hay. Prueba en otros servidores FTP que admiten usuarios anónimos como [ftp.mozilla.org](http://ftp.mozilla.org), [ftp.freebsd.org](http://ftp.freebsd.org), etc.

### 3.2 Subir ficheros al servidor

Para los siguientes ejemplos necesitamos tener acceso a un servidor FTP. Podemos crear un hosting web gratuito que ofrezca servicio de FTP (<http://www.x90x.net/>, <http://byethost.com/>, <http://www.lk6.com.ar/>, etc), o podemos instalar el servidor FTP Filezilla Server en nuestra máquina local.

Para subir ficheros al servidor necesitamos un usuario, su clave y un espacio en el servidor y tener privilegios para ello. En primer lugar necesitamos situarnos en el directorio donde vamos a subir los ficheros; por ejemplo, supongamos que es un subdirectorio que cuelga del directorio raíz y se llama `psp`:

```
String directorio = "/psp";
cliente.changeWorkingDirectory(directorio);
```

A continuación con el método `setFileType` se indica el tipo de fichero a subir. Este tipo es una constante entera definida en la clase FTP. Se suele utilizar `BINARY_FILE_TYPE` que permite enviar ficheros de cualquier tipo:

```
cliente.setFileType(FTP.BINARY_FILE_TYPE);
```

Creamos un stream de entrada con los datos del fichero que vamos a subir (en el ejemplo el fichero `texto1.txt` ubicado en la carpeta `d:\documentos`) y se lo pasamos al método `storeFile`. En el primer parámetro indicaremos el nombre que tendrá el fichero en el directorio FTP y en el segundo el `InputStream`:



```
BufferedInputStream in = new BufferedInputStream(  
    new FileInputStream("d:\\documentos\\textol.txt"));  
cliente.storeFile("textol.txt", in);
```

El ejemplo completo se muestra a continuación, se suben 2 ficheros al directorio /psp (que tiene que existir), uno de texto y el otro una imagen; se le da el mismo nombre en el servidor que el que tienen actualmente:

```
import java.io.BufferedInputStream;  
import java.io.FileInputStream;  
import java.io.IOException;  
  
import org.apache.commons.net.ftp.FTP;  
import org.apache.commons.net.ftp.FTPClient;  
  
public class SubirFichero {  
    public static void main(String[] args) {  
        FTPClient cliente = new FTPClient();  
        String servidor = "localhost";  
        String user = "usuario";  
        String pass = "clave";  
        BufferedInputStream in = null;  
        try {  
            System.out.println("Conectándose a " + servidor);  
            cliente.connect(servidor);  
            boolean login = cliente.login(user, pass);  
            if (login) {  
                cliente.changeWorkingDirectory("/psp");  
                cliente.setFileType(FTP.BINARY_FILE_TYPE);  
                in = new BufferedInputStream(  
                    new FileInputStream("d:\\documentos\\textol.txt"));  
                cliente.storeFile("textol.txt", in);  
                in.close();  
                in = new BufferedInputStream(  
                    new FileInputStream("d:\\documentos\\imagen1.jpg"));  
                cliente.storeFile("imagen.jpg", in);  
                cliente.logout();  
                cliente.disconnect();  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
        finally {  
            if (in != null)  
                try {  
                    in.close();  
                } catch (IOException e) {  
                    e.printStackTrace();  
                }  
        }  
    }  
}
```

Para renombrar un fichero se usa el método `rename`. Devuelve `true` si renombra el fichero con éxito, en caso contrario devuelve `false`. Por ejemplo, renombro el fichero de nombre `Lisa.png` que se encuentra en la carpeta `/htdocs/LosSimpson/personajes` a `LisaSimpson.png`:

```
cliente.changeWorkingDirectory("/htdocs/LosSimpson/personajes");  
if (cliente.rename("Lisa.png", "LisaSimpson.png"))  
    System.out.println("Fichero renombrado... ");  
else  
    System.out.println("No se ha podido renombrar el Fichero... ");
```

Para eliminar un fichero usamos el método `deleteFile`. Devuelve `true` si elimina el fichero y `false` si no lo elimina. Por ejemplo, para eliminar el fichero de nombre `Lisa.png`, ubicado en la carpeta `/htdocs/LosSimpson/personajes` escribo se ejecutarán las sentencias siguientes:

```

if(cliente.deleteFile("/htdocs/LosSimpson/personajes/Lisa.png"))
    System.out.println("Fichero eliminado... ");
else
    System.out.println("No se ha podido eliminar Fichero... ");

```

### 3.3 Descargar ficheros del servidor

Para descargar un fichero del servidor en nuestro disco duro usamos el método `retrieveFile(String remote, OutputStream local)`. Necesitamos saber el directorio desde el que descargaremos el fichero. El método devuelve `true` si el proceso se realiza satisfactoriamente, en caso contrario devuelve `false`. Necesitaremos crear un stream de salida para escribir el fichero en nuestro disco duro.

Por ejemplo para descargar el fichero de nombre `texto2.txt`, que se ubica en la carpeta del servidor FTP `/documentos` en nuestro disco duro en la carpeta `d:\documentos` y guardarlo con el nombre `texto2nuevo.txt`, se ejecutarán las sentencias siguientes:

```

cliente.changeWorkingDirectory("/documentos");
BufferedOutputStream out = new BufferedOutputStream(
    new FileOutputStream("d:\\documentos\\texto2nuevo.txt"));
if (cliente.retrieveFile("texto2.txt", out))
    System.out.println("Descargado correctamente... ");
else
    System.out.println("No se ha podido descargar... ");
out.close();

```

### 3.4 Creación de un cliente FTP

A continuación vamos a crear un cliente sencillo desde el que podremos subir, descargar y eliminar ficheros, crear y eliminar directorios o carpetas en un servidor FTP. Como servidor FTP usaremos *Filezilla Server* instalado en nuestra máquina local.

Para el ejemplo se crea un usuario con nombre "usuario1" y clave "usuario" y la carpeta `d:\ftp\usuario1` con algunos archivos dentro. Desde el cliente de administración de FileZilla Server se asigna al usuario la carpeta creada dándole todos los permisos:

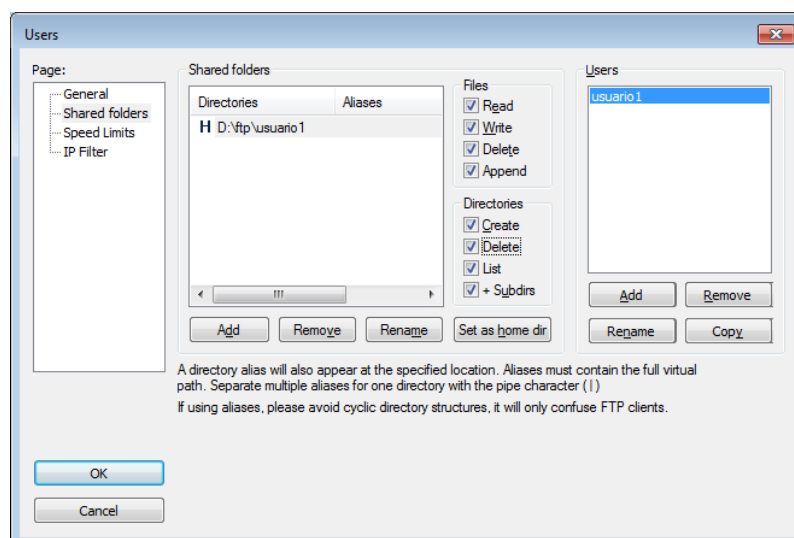


Figura 2. Asignación de directorio y permisos a usuario1.

Los datos que necesitaremos para la conexión al servidor FTP son el nombre del servidor, el nombre del usuario y su clave. En el programa se han usado las siguientes variables para almacenar estos datos y se han asignado los siguientes valores: `servidor = "127.0.0.1"`, `user = "usuario1"` y `password = "usuario"`.

**NOTA:** Para mostrar en la consola el contenido de los mensajes comando/respuesta que se van originando en la comunicación con el servidor FTP podemos usar el método `addProtocolCommandListener` de la clase `SocketClient`. Se escribe la siguiente expresión antes de realizar la conexión al servidor:

```
cliente.addProtocolCommandListener(  
    new PrintCommandListener(new  
        PrintWriter(System.out));
```

La interfaz `ProtocolCommandListener` junto con la interfaz `PrintCommandListener`

---

## PRÁCTICA 2

Crea un cliente FPT con la interfaz de usuario que se muestra a continuación:

---

## 4 COMUNICACIÓN CON UN SERVIDOR TELNET

Telnet es un protocolo de red que pertenece a la familia de protocolos de Internet. Permite a los usuarios acceder a un ordenador remoto y realizar tareas como si estuviesen trabajando directamente delante de él. El acceso al ordenador remoto se realiza en modo terminal, es decir, no se muestra una pantalla gráfica (como el escritorio de Windows) para realizar las tareas; se trabaja desde la línea de comandos. Es útil para arreglar fallos de forma remota y para consultar información. Se utiliza bastante en sistemas UNIX y en equipos de comunicaciones para la configuración de routers. Utiliza el puerto TCP 23.

El principal problema de Telnet es la seguridad, ya que los datos necesarios para conectarse a máquinas remotas (nombre de usuario y clave), no son cifrados y viajan por la red como texto plano, facilitando a cualquiera que espíe la red mediante un programa sniffer obtener estos datos.

Telnet sigue un modelo cliente-servidor, para poder utilizarlo necesitamos tener instalado en una máquina un servidor Telnet y en otra máquina el cliente Telnet para poder acceder a ella.

### 4.1 Instalación y uso de un servidor Telnet

Para instalar un servidor Telnet en Windows (Windows 7 Home Premium) pulsamos en *Inicio-> Panel de control-> Programas-> Programas y características-> Activar y desactivar las características de Windows*; se abre una ventana desde la que podremos activar o desactivar características de Windows. Marcamos las casillas *Servidor Telnet* para instalar el servidor y *Cliente Telnet* para instalar el cliente.

Después se pulsa *Aceptar*, un mensaje nos indica que hay que esperar unos minutos hasta que la instalación finalice, al finalizar puede que nos pida reiniciar el equipo.

El servidor Telnet se instala como un servicio de Windows, será necesario iniciarlo para poder trabajar con él. Accediendo a los servicios de Windows (*Inicio -> Panel de control -> Herramientas administrativas-> Servicios*) se puede indicar el tipo de inicio: manual, automático, etc.

Para permitir que los usuarios obtengan acceso al servidor Telnet, es necesario agregarlos al grupo *TelnetClients*, seguiremos estos pasos:

- Abrimos una ventana de terminal en el modo "Ejecutar como administrador".
- Ejecutamos el comando siguiente para añadir el usuario creado al grupo *TelnetClients*:

```
net localgroup TelnetClients /add usuario1
```

Se debe visualizar el mensaje *Se ha completado el comando correctamente*.

- Para comprobar si se ha añadido el usuario ejecutamos esta orden:

```
net localgroup TelnetClients
```

Se debe ver el usuario en la lista de miembros.

Para probar la conexión con el servidor Telnet ejecutamos desde la línea de comandos la orden `telnet localhost`. Una vez introducido el nombre de usuario y la contraseña e iniciada la sesión, se mostrará el prompt donde podremos escribir comandos e interactuar con el equipo remoto (en este caso el equipo local).

## 4.2 Comunicación desde Java con un servidor Telnet

La librería Apache Commons Net™ proporciona la clase `TelnetClient` (extiende `SocketClient`) que implementa el sencillo terminal virtual de red (NVT) para el protocolo Telnet según RFC 854. Presenta dos tipos de constructores:

|                                            |                                                                                                                   |
|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| <code>TelnetClient()</code>                | Constructor por defecto, establece como tipo de terminal VT100.                                                   |
| <code>TelnetClient(String termtype)</code> | Se establece como tipo de terminal el especificado en el String <code>termtype</code> (XTerm, VT100, VT200, etc.) |

La clase utiliza el método `connect` de la clase `SocketClient` para conectarse al servidor. Una vez conectado se obtienen un `InputStream` y un `OutputStream` para leer y enviar datos a través de la conexión que se pueden obtener mediante los métodos `getInputStream` y `getOutputStream`.

Algunos métodos de esta clase son:

|                                           |                                                                                                                    |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| <code>void disconnect()</code>            | Desconecta la sesión Telnet, cierra los input y output stream así como el socket.                                  |
| <code>InputStream getInputStream()</code> | Devuelve el stream de entrada de la conexión Telnet, se usa para leer las respuestas que envía el servidor Telnet. |

|                                             |                                                                                                         |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>OutputStream getOutputStream()</code> | Devuelve el stream de salida de la conexión Telnet, se usa para enviar los comandos al servidor Telnet. |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------|

En el siguiente ejemplo nos conectamos al servidor Telnet local con nombre de usuario `usuario1` y clave `usuario` y le enviamos el comando `DIR` para que nos muestre el contenido del directorio. Vamos a ver paso a paso las operaciones de lectura mostrando lo que nos envía el servidor y de escritura enviando lo que nos pide (login, password, comandos,...)

En primer lugar creamos el cliente Telnet mediante el segundo constructor estableciendo como tipo de terminal `XTerm`. A continuación se realiza la conexión con el servidor mediante el método `connect`. Una vez conectados se crean el `OutputStream` o flujo de salida para enviar datos al servidor y el `InputStream` o flujo de entrada para leer los datos que el servidor envía:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintStream;
import org.apache.commons.net.telnet.TelnetClient;

public class ClienteTelnet1 {
    static InputStream in;
    static PrintStream out;

    public static void main(String[] args) {
        TelnetClient telnet = new TelnetClient("xterm");
        String servidor = "localhost";
        String login = "usuario1";
        String password = "usuario";

        try {
            telnet.connect(servidor);
            out = new PrintStream(telnet.getOutputStream());
            in = telnet.getInputStream();
            // Recibe las primeras lineas y pide login:
            RecibirDatos();
            // Se envia el nombre de usuario
            EnviarDatos(login);
            // Ahora pide password:
            RecibirDatos();
            // Se envia el password
            EnviarDatos(password);
            // Se muestra la presentación y el prompt
            RecibirDatos();
            // se manda el comando DIR
            EnviarDatos("DIR");
            // lectura visualizacion de la salida del comando DIR
            RecibirDatos();
            // desconectar
            telnet.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Recibir y visualizar lo que envía el servidor Telnet
    static void RecibirDatos() throws IOException {
        byte[] buff = new byte[1024];
        int nbytes;
        nbytes = in.read(buff);
        System.out.print(new String(buff, 0, nbytes));
    }

    // Enviar datos al servidor Telnet
    static void EnviarDatos(String cad) {
        out.println(cad);
        out.flush();
    }
}
```

```
}
```

La ejecución del programa muestra la siguiente salida:

```
Welcome to Microsoft Telnet Service

login: usuario1

password:

*=====
Microsoft Telnet Server.
*=====
C:\Users\usuario1>DIR
El volumen de la unidad C es OS
El número de serie del volumen es: 2484-EBDD

Directorio de C:\Users\usuario1

05/04/2013  00:08    <DIR>          .
05/04/2013  00:08    <DIR>          ..
14/07/2009  04:34    <DIR>          Desktop
04/04/2013  23:58    <DIR>          Documents
14/07/2009  04:34    <DIR>          Downloads
14/07/2009  04:34    <DIR>          Favorites
14/07/2009  04:34    <DIR>          Links
14/07/2009  04:34    <DIR>          Music
14/07/2009  04:34    <DIR>          Pictures
11/10/2011  12:59    <DIR>          Roaming
14/07/2009  04:34    <DIR>          Saved Games
14/07/2009  04:34    <DIR>          Videos
                0 archivos                0 bytes
                13 dirs   125.738.921.984 bytes libres

C:\Users\usuario1>
```

En este ejemplo se hace una lectura (llamada al método `LecturaDatos`) cada vez que se van a recibir datos del servidor. Podemos hacer que el proceso de lectura se realice en un hilo, una vez conectados al servidor Telnet se inicia el hilo y se leen los datos o respuestas que el servidor envía desde el hilo. El hilo recibe en su constructor el objeto `TelnetClient`:

```
import java.io.IOException;
import java.io.InputStream;

import org.apache.commons.net.telnet.TelnetClient;

class LecturaRespuestas extends Thread {

    TelnetClient cliente = null;

    public LecturaRespuestas(TelnetClient cliente) {
        this.cliente = cliente;
    }

    public void run() {
        InputStream in = cliente.getInputStream();
        byte[] buff = new byte[1024];
        int nbytes = 0;
        try {
            nbytes = in.read(buff);
            while (nbytes > 0) {
                System.out.print(new String(buff, 0, nbytes));
                nbytes = in.read(buff);
            }
        } catch (IOException e) {
            System.err.println("Conexión con el servidor finalizada: " +
e.getMessage());
        }
    }
}
```

```
}
```

Lo más normal es que el usuario escriba por teclado las órdenes a enviar al servidor. Una vez conectados e iniciado el hilo de lectura de respuestas del servidor hacemos un bucle desde el que se introducen datos por teclado y se envían al servidor. El proceso termina cuando el usuario escriba `exit`. El código es el siguiente:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;

import org.apache.commons.net.telnet.TelnetClient;

public class ClienteTelnet2 {
    static TelnetClient telnet = new TelnetClient();

    public static void main(String[] args) {
        String server = "10.0.2.100";
        String cadena = null;
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        try {
            telnet.connect(server);
        } catch (IOException e) {
            System.err.println("Error de conexión: " + e.getMessage());
            System.exit(-1);
        }

        LecturaRespuestas hilo = new LecturaRespuestas(telnet);
        hilo.start();

        PrintStream out = new PrintStream(telnet.getOutputStream());
        do {
            try {
                if ((cadena = in.readLine()).length() > 0) {
                    out.println(cadena);
                    out.flush();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        } while (!cadena.trim().equalsIgnoreCase("exit"));

        try {
            telnet.disconnect();
            System.out.println("Fin de la conexión ....");
        } catch (IOException e) {
            System.err.println("Error de desconexión: " + e.getMessage());
        }


        while (true)
            try {
                hilo.join();
                break;
            } catch (InterruptedException e) {
            }
    }
}
```

## 5 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, al puerto 25). A cada comando que envía el cliente le sigue una respuesta del servidor compuesta por un número y un mensaje descriptivo. Las especificaciones de este protocolo se definen en la RFC 2821.

## 5.1 Instalación de un servidor de correo electrónico

Un servidor SMTP es un programa que permite enviar correo electrónico a otros servidores SMTP. Vamos a ver cómo se instala un simple servidor SMTP en nuestro equipo local en el sistema operativo Windows (para el ejemplo se ha usado Windows 7). Descargamos ArGoSoft Mail Server de la URL <http://argosoft-mail-server.uptodown.com/windows> y lo instalamos. Durante la instalación puede que se muestre la ventana de alerta de seguridad Windows pidiéndonos confirmación para permitir acceso al programa a través del firewall de Windows. Si es así, pulsamos en el botón *Permitir acceso* para continuar.

Una vez instalado, ejecutamos *ArGoSoft Mail Server* y procedemos a configurar el servidor. Para ello, en la pantalla principal se pulsa en el botón Options , o se selecciona en el menú principal la opción *Tools -> Options*. Marcamos las casillas que se muestran en la imagen y pulsamos OK:

- DNS Server dirección IP de un servidor de dominios, por ejemplo usamos 8.8.8.8, DNS de Google.
- Automatically Start the Server: marcamos esta casilla para que se inicie el servidor cuando abramos ArGoSoft Mail.
- Local Host: como vamos a usar el servidor local escribimos 127.0.0.1

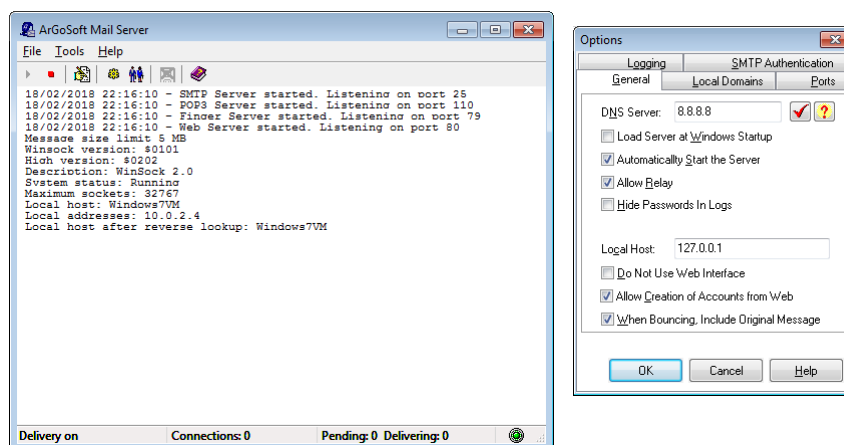


Figura 3. Pantalla principal de ArGoSoft Mail Server y configuración del servidor.

Desde la pestaña *Logging* podemos marcar las casillas *Log SMTP Commands*, *Log SMTP Conversations with Exchangers* y *Log to File* para si ocurre algún fallo consultar los registros de log con las conversaciones entre cliente y servidor. Desde el menú principal se pueden consultar con la opción *Tools -> View log file*.

Normalmente cuando creamos una cuenta de correo en un proveedor de servicios de internet, el proveedor nos 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, etc. El primero se utiliza para recibir los mensajes (es decir, para configurar el correo entrante) y el segundo para enviar nuestros mensajes (configurar el correo saliente). Podemos usar el servidor SMTP que hemos instalado para enviar correos, en lugar de utilizar el proporcionado por nuestro proveedor de correo.

## 5.2 Uso de Telnet para comunicar con el servidor SMTP

Vamos a ver a continuación como enviar un correo electrónico de forma manual usando Telnet al puerto 25 del servidor SMTP. Algunos de los comandos que usaremos se muestran en la siguiente tabla:



|                   |                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------|
| HELO o EHLO       | Se utiliza para abrir una sesión con el servidor.                                                               |
| MAIL FROM: origen | A la derecha se indica quien envía el mensaje, por ejemplo: remitente@servidor.com                              |
| RCPT TO: destino  | A la derecha se indica el destinatario del mensaje, por ejemplo: destinatario@servidor.com                      |
| DATA mensaje      | Se utiliza para indicar el comienzo del mensaje, éste finalizará cuando haya una línea únicamente con un punto. |
| QUIT              | Cierra la sesión.                                                                                               |
| HELP              | Muestra la lista de comandos SMTP que el servidor admite.                                                       |

Abrimos un terminal y escribimos telnet. A continuación escribimos: open localhost 25. El servidor nos responde con la siguiente línea:

```
220 127.0.0.1 ArGoSoft Mail Server Freeware, Versión 1.8 (1.8.9.1)
```

Normalmente responde con un número de 3 dígitos donde cada uno tiene un significado especial (como en FTP). Por ejemplo los números que empiezan en 2 (220, 250, ...) indican que la acción se ha completado con éxito; los que empiezan por 3 (354) indican que el comando ha sido aceptado, pero la acción solicitada está suspendida a la espera de recibir más información, se usa en grupo de secuencias de comandos (DATA). Los que empiezan por 4 indican que el comando no fue aceptado pero se puede volver a escribir de nuevo. Los que empiezan por 5 indican que el comando no ha sido aceptado y la acción no se ha realizado por ejemplo, 502 Unknown command.

Empezamos a escribir los comandos, primero se abre la sesión con HELO, a continuación escribimos el origen (MAIL FROM:) y el destino del mensaje (RCPT TO:), por cada línea que vamos escribiendo el servidor nos responde. Por último mediante el comando DATA enviamos el mensaje, para finalizar el mensaje escribimos una línea únicamente con un punto. Para finalizar escribimos QUIT. En el siguiente ejemplo se escribe un correo a la dirección *dampsp2018\_profesor@outlook.es* que se usará como origen y *dampsp2018\_alumnos@outlook.es* que se usará como destino. A continuación se muestra la secuencia completa:

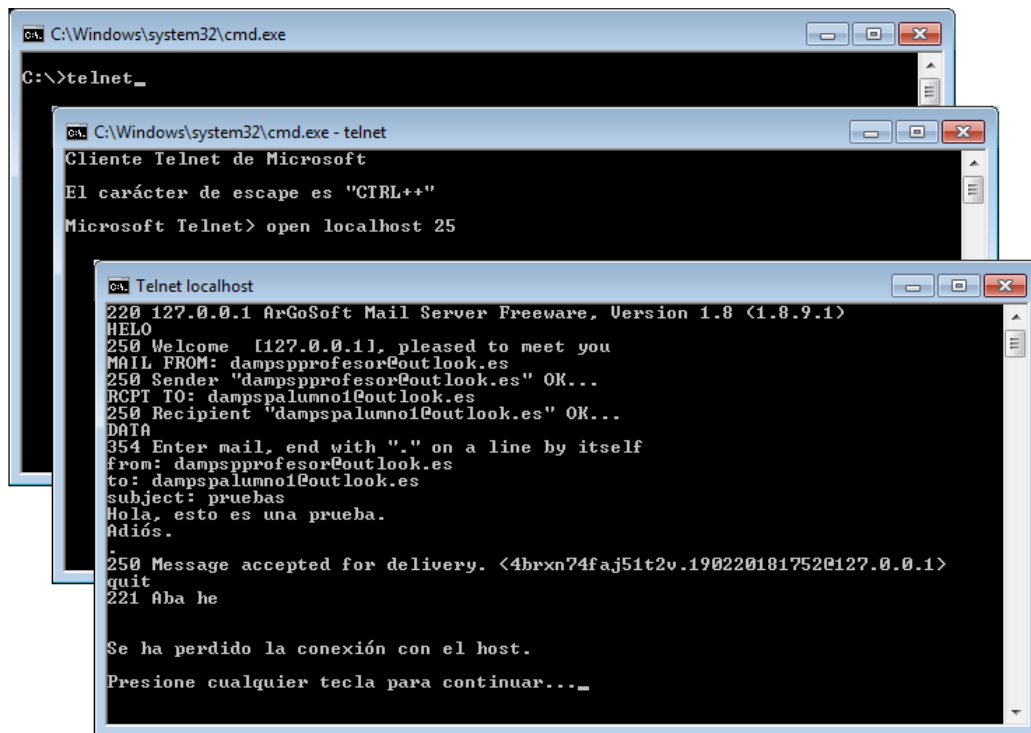


Figura 4. Envío de correo mediante Telnet

Con el comando DATA empieza el cuerpo del mensaje. Si se el servidor responde con el mensaje 354 Enter mail, end with "." on a line by itself, se puede empezar a escribir el cuerpo que puede contener las siguientes cabeceras: Date, Subject, To, Ce y From (se pueden escribir en mayúsculas o minúsculas).

### 5.3 Comunicación desde Java con un servidor SMTP

La librería Apache Commons Net™ proporciona la clase SMTPClient (extiende SMTP) que 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 SocketClicnt, 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 correctamente.

El siguiente ejemplo, copiado de la documentación de *Apache Commons Net* de la clase SMTPClient, constituye un ejemplo sencillo de como prodría ser la comunicación con un servidor SMTP:

```
import java.io.IOException;

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

public class ClienteSMTP1 {
    public static void main(String[] args) {
        SMTPClient client = new SMTPClient();
        try {
            int reply;
            client.connect("localhost");
            System.out.print(client.getReplyString());

            // After connection attempt, you should check the reply code to verify
            // success.
            reply = client.getReplyCode();

            if (!SMTPReply.isPositiveCompletion(reply)) {
```

```

        client.disconnect();
        System.err.println("SMTP server refused connection.");
        System.exit(1);
    }

    // Do useful stuff here.
    // ...

} catch (IOException e) {
    if (client.isConnected()) {
        try {
            client.disconnect();
        } catch (IOException f) {
            // do nothing
        }
    }
    System.err.println("Could not connect to server.");
    e.printStackTrace();
    System.exit(1);
}
}
}

```

La salida que se muestra al ejecutar el programa es:

```
220 127.0.0.1 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)
```

La clase `SMTPReply` almacena un conjunto de constantes para códigos de respuesta SMTP. Para interpretar el significado de los códigos se puede consultar la RFC 2821 (<http://tools.ietf.org/html/rfc2821>). El método `isPositiveCompletion(int respuesta)` devuelve `true` si un código de respuesta ha terminado positivamente. Los métodos `getReplyString` y `getReplyCode` son métodos de la clase `SMTP` y son similares a los vistos en la clase `FTP`.

`SMTPClient` presenta dos tipos de constructores:

|                                                 |                                                                              |
|-------------------------------------------------|------------------------------------------------------------------------------|
| <code>SMTPClient()</code>                       | Constructor por defecto.                                                     |
| <code>SMTPClient() (String codificación)</code> | Se establece una codificación en el constructor (BASE64, BINARY, 8BIT, etc.) |

La clase 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 conectarnos a un servidor SMTP cuyo puerto de escucha sea distinto, tendríamos que indicar en la conexión el número de puerto: `connect(host, puerto)`.

Algunos métodos de esta clase son:

|                                                   |                                                                                                                      |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| <code>boolean addRecipient(String address)</code> | Añade la dirección de correo de un destinatario usando el comando RCPT.                                              |
| <code>boolean completePendingCommand()</code>     | Este método se utiliza para finalizar la transacción y verificar el éxito o el fracaso de la respuesta del servidor. |
| <code>boolean login()</code>                      | Inicia sesión en el servidor SMTP enviando el comando HELO.                                                          |
| <code>boolean login(String hostname)</code>       | Igual que la anterior pero envía el nombre del host como argumento.                                                  |

|                                                                                                  |                                                                                                                                                                           |
|--------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>boolean logout()</code>                                                                    | Finaliza la sesión con el servidor enviando el comando QUIT.                                                                                                              |
| <code>Writer sendMessageData()</code>                                                            | Envía el comando DATA para después enviar el mensaje de correo. La clase Writer se usará para escribir secuencias de caracteres, como la cabecera y el cuerpo del mensaje |
| <code>boolean sendShortMessageData(String message)</code>                                        | Método útil para envío de mensajes cortos.                                                                                                                                |
| <code>boolean sendSimpleMessage(String remitente, String[] destinatarios, String message)</code> | Un método útil para el envío de un correo electrónico corto. Se especifica el remitente, los destinatarios y el mensaje.                                                  |
| <code>boolean sendSimpleMessage(String remitente, String destinatario, String message)</code>    | Igual que el anterior pero el mensaje sólo va dirigido a un destinatario.                                                                                                 |
| <code>boolean setSender(String address)</code>                                                   | Se especifica la dirección del remitente usando el comando MAIL.                                                                                                          |
| <code>boolean verify(String username)</code>                                                     | Compruebe que un nombre de usuario o dirección de correo electrónico es válida (envía el comando VRFY para comprobarlo, tiene que estar soportado por el servidor).       |

Para enviar un simple mensaje a un destinatario podemos escribir las siguientes líneas:

```
client.login();
String destinatario="dampspalumno1@outlook.es";
String mensaje = "Hola.";
mensaje += System.lineSeparator();
mensaje += "Enviando saludos.";
mensaje += System.lineSeparator();
mensaje += "Adios.";
String remitente="dampspprofesor@outlook.es";
client.sendSimpleMessage(remitente, destinatario, mensaje);
client.logout();
```

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 un mensaje de error: *"Our system has detected that this message is not RFC 2822 compliant. Tu reduce the amount of spam sent to Gmail, this message has been blocked. Please review RFC 2822 specifications for more Information. tc4si5253654pbc.4 – gsmtp"*. En otros puede que llegue 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 subject)
```

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

`from`: valor del campo de cabecera *from*, dirección de correo origen.

`to`: valor del campo de cabecera *to*, dirección de correo destino.

`subject`: valor del campo de cabecera *subject*, asunto del mensaje.

Proporciona los siguientes métodos:

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

Desde el siguiente código se envía un correo a dos destinatarios (almacenados en las variables `destino1` y `destino2`), el texto se encuentra en la variable `mensaje`. Mediante la clase `SimpleSMTPHeader` se construye la cabecera y se agrega al campo `CC` el segundo destinatario. Mediante el método `setSender` establecemos el remitente y mediante el método `addRecipient` añadimos los destinatarios del mensaje, que en este caso son 2:

```
client.login();
String origen = "dampspprofesor@outlook.es";
String destino1 = "dampspalumno1@outlook.es";
String destino2 = "dampspalumno2@outlook.es";
String asunto = "Prueba de SMTPClient";
String mensaje = "Hola.";
mensaje += System.lineSeparator();
mensaje += "Enviando saludos.";
mensaje += System.lineSeparator();
mensaje += "Adios.";
SimpleSMTPHeader cabecera = new SimpleSMTPHeader(origen, destino1, asunto);
cabecera.addCC(destino2);
client.setSender(origen);
client.addRecipient(destino1);
client.addRecipient(destino2);
```

Después se crea un objeto `Writer` para escribir el mensaje. Con el método `sendMessageData` se envía el comando `DATA`, después se escribe la cabecera del correo y a continuación el cuerpo. Luego se cierra el stream. Por último se comprueba si el correo se ha enviado correctamente mediante el método `completePendingCommand` y se cierra la sesión:

```
Writer writer = client.sendMessageData();
if (writer == null) {
    System.err.println("Error al enviar el comando 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);
}
client.logout();
```

Hasta ahora hemos utilizado el servidor SMTP sin necesidad de autenticarnos (por defecto al instalar Argosoft la autenticación no está activada). La autenticación SMTP se configura con el fin de elevar los niveles de seguridad y eficacia del servicio de correo electrónico y con el objetivo de prevenir que nuestra dirección de correo sea utilizada sin autorización, evitando el posible envío de correos no deseados

a otras personas con fines peijudiciales. La autenticación se realiza a través de la verificación de su nombre de usuario y su contraseña.

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

La clase `SMTPSClient` (extiende `SMTPClient`) proporciona varios constructores. Usar uno u otro dependerá de los datos que nos proporcione el servidor SMTP en el que tengamos nuestra cuenta de correo. El constructor sin parámetros y el constructor con un parámetro booleano `SMTPSClient(boolean implicit)`.

Si se selecciona el segundo constructor con el parámetro con valor `true` (modo implícito), la negociación SSL/TLS comienza justo después de que se haya establecido la conexión. En modo explícito (primer constructor), la negociación SSL/TLS se inicia cuando el usuario llama al método `execTLS` y el servidor acepta el comando.

Ejemplo de uso en modo implícito:

```
SMTPSClient c = new SMTPSClient(true);
c.connect("127.0.0.1", 465);
```

Ejemplo de uso en modo explícito:

```
SMTPSClient c = new SMTPSClient();
c.connect("127.0.0.1", 25);
if (c.execTLSQ) {
    /* resto de sentencias aquí */
}
```

Alguno de los métodos que usaremos después para realizar la autenticación SMTP son:

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

La clase `AuthenticatingSMTPClient` presenta varios constructores. Utilizaremos el constructor por defecto para crear un nuevo cliente de autenticación SMTP. Puede lanzar `NoSuchAlgorithmException`, esta excepción se produce cuando un algoritmo criptográfico, habiéndose solicitado, no está disponible en el entorno. Algunos de los métodos de la clase son los siguientes:

|                                                                                                        |                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <pre>boolean auth(AuthenticatingSMTPClient.AUTH_METHOD method, String username, String password)</pre> | Se envía el comando AUTH1 con el método seleccionado para autenticarse en el servidor SMTP, se envía el nombre del usuario y su clave. |
|--------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|

|                                        |                                                                                                                         |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>int ehlo(String hostname)</code> | 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*.

En el siguiente ejemplo vamos a usar el servidor SMTP de gmail. Los datos que proporciona gmail son los siguientes:

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

En el ejemplo se definen las variables con el nombre y la clave del usuario que se autentica en el servidor, el nombre del servidor y el puerto utilizado (en este caso el 587). Se crea una instancia de la clase `AuthenticatingSMTPClient` para crear el cliente SMTP seguro. Mediante la clase `KeyManagerFactory` creamos las `KeyManager` para establecer un canal de comunicación seguro (se incluyen los import necesarios para estas clases):

```
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.AuthenticatingSMTPClient;
import org.apache.commons.net.smtp.SMTPReply;
import org.apache.commons.net.smtp.SimpleSMTPHeader;

public class ClienteSMTP3 {
    public static void main(String[] args) throws NoSuchAlgorithmException,
        UnrecoverableKeyException, KeyStoreException,
        InvalidKeyException, InvalidKeySpecException {
        AuthenticatingSMTPClient client = new AuthenticatingSMTPClient();
        String server = "smtp.gmail.com";
        String username = "un_usuario@gmail.com";
        String password = "una_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];
```

A continuación se utiliza el método `connect` para realizar la conexión al servidor SMTP y se establece la clave para la comunicación segura. Antes de continuar se comprueba el código de respuesta generado:

```
// nos conectamos al servidor SMTP
client.connect(server, puerto);
System.out.println("1 - " + client.getReplyString());
// se establece 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);
        }
    }

```

Se envía el comando EHLO mediante el método `ehlo` y se ejecuta el método `execTLS`. Si tiene éxito se realiza todo el proceso empezando por la autenticación del usuario mediante el método `auth`. Como método de autenticación se usa `AuthenticatingSMTPClient.AUTH_METHOD.PLAIN`. Después se preparan las cabeceras y el texto del mensaje, se añaden el emisor y destinatario del mensaje y se escribe en el `Writer`:

```

// se envia el commando EHLO
client.ehlo(server);
System.out.println("2 - " + client.getReplyString());
// Se ejecuta el comando STARTTLS y se comprueba si es true
if (client.execTLS()) {
    System.out.println("3 - " + client.getReplyString());
    // se realiza la autenticación con el servidor
    if (client.auth(AuthenticatingSMTPClient.AUTH_METHOD.PLAIN,
        username, password)) {
        System.out.println("4 - " + client.getReplyString());
        String destino = "dampspalumnol@outlook.es";
        String asunto = "Prueba de SMTPClient con GMAIL";
        String mensaje = "Hola.";
        mensaje += System.lineSeparator();
        mensaje += "Enviando saludos usando GMail.";
        mensaje += System.lineSeparator();
        mensaje += "Adios.";
        // se crea la cabecera
        SimpleSMTPHeader cabecera = new SimpleSMTPHeader(username,
            destino, asunto);
        // el nombre de usuario y el email de origen coinciden
        client.setSender(username);
        client.addRecipient(destino);
        System.out.println("5 - " + client.getReplyString());
        // se envia DATA
        Writer writer = client.sendMessageData();
        if (writer == null) {
            System.err.println("Fallo al enviar DATA.");
            System.exit(1);
        }
        writer.write(cabecera.toString());
        writer.write(mensaje);
        writer.close();
        System.out.println("6 - " + client.getReplyString());
        boolean exito = client.completePendingCommand();
        System.out.println("7 - " + client.getReplyString());
        if (!exito) {
            System.err.println("Fallo al finalizar la transacción.");
            System.exit(1);
        }
    } else
        System.err.println("Usuario no autenticado.");
} else
    System.err.println("Fallo al ejecutar STARTTLS.");
} catch (IOException e) {
    System.err.println("Could not connect to server.");
    e.printStackTrace();
    System.exit(1);
}
try {
    client.disconnect();
} catch (IOException f) {
    f.printStackTrace();
}
System.out.println("Fin de envío.");
System.exit(0);
}

```



La salida generada al ejecutar el programa mostrando las respuestas que envía el servidor es la siguiente:

```
1 - 220 smtp.gmail.com ESMTP p60sm13902498wrc.88 - gsmtip
2 - 250-smtp.gmail.com at your service, [46.222.206.148]
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 p60sm13902498wrc.88 - gsmtip

6 - 354 Go ahead p60sm13902498wrc.88 - gsmtip

7 - 250 2.0.0 OK 1519042681 p60sm13902498wrc.88 - gsmtip

Fin de envío.
```

Algunos códigos de respuesta son: 220: indica que el servicio está preparado, 250: solicitud aceptada, 235: autenticación aceptada, 354: acepta la entrada de datos. Si el servidor SMTP no requiere negociar el uso de TLS hay que prescindir de la sentencia `if (client.execTLS())`.

### PRÁCTICA 3

Realiza un programa Java en el que utilices el servidor SMTP de tu cuenta de correo para enviar un mensaje a alguno de tus contactos. Los datos para conectarse al servidor SMTP (nombre servidor, usuario, clave y puerto) se introducen desde la línea de comandos.

## 5.4 Acceso a los mensajes de un servidor POP3



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

- **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 documentos (como imágenes, sonido y texto) en un mensaje. Su versión segura se denomina **S/MIME**.
- **POP** (*Post Office Protocol*): proporciona acceso a los mensajes de los servidores SMTP. En general, cuando hacemos referencia al término POP, nos estamos refiriendo a POP3 que es la última versión. Este es el que usaremos en este apartado.
- **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 sobre POP, por ejemplo, los mensajes continúan siempre almacenados en el servidor, cosa que no ocurre con POP o los usuarios pueden organizar los mensajes en carpetas. La versión actual es la 4, **IMAP4**.

A POP3 se le asigna el puerto 110. El Servidor POP3 es el servidor de correo electrónico entrante y utiliza en general el puerto 110. Al igual que SMTP, funciona con comandos de texto. Algunos de ellos se muestran en esta tabla:

|                      |                                                                      |
|----------------------|----------------------------------------------------------------------|
| USER login           | A la derecha se escribe el login de la cuenta de correo              |
| PASS contraseña      | A la derecha se escribe la contraseña de la cuenta de correo         |
| STAT                 | Muestra el número de mensajes de la cuenta                           |
| LIST                 | Listado de mensajes (numero - tamaño total del mensaje)              |
| RETR número-mensaje  | Obtiene el mensaje cuyo número coincida con el indicado a la derecha |
| DELE número-mensaje  | Borra el mensaje indicado                                            |
| QUIT                 | Cierra la conexión.                                                  |
| TOP número-mensaje n | Muestra las $n$ primeras líneas del mensaje indicado.                |

## 5.5 Uso de Telnet para comunicar con el servidor POP

Podemos hacer Telnet al puerto 110 para interactuar con el servidor POP. Para probarlo vamos a crear un usuario en nuestro servidor local de correo. Desde la pantalla principal de Argosoft hacemos clic en la opción de menú *Tools->Users* o bien clic en el botón *Users* . Se abre una ventana en la Pulsamos en el botón *Add new User* . Rellenamos los campos, pulsamos *OK* y después cerramos la ventana:

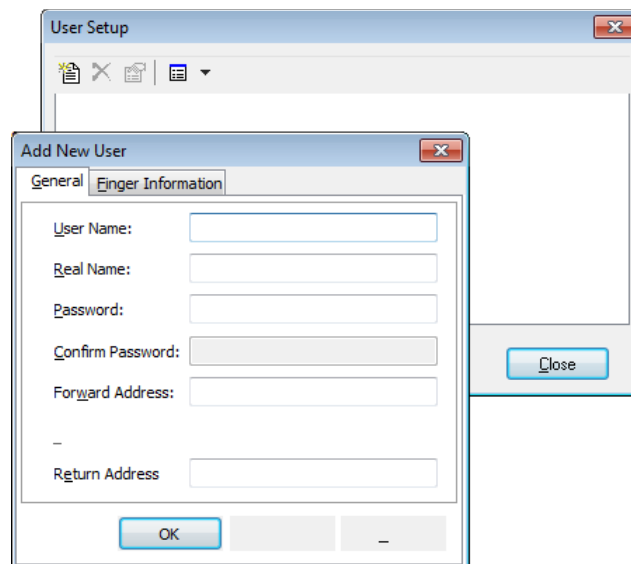


Figura 5. Creación de un usuario de correo.

Ahora, usando los comandos SMTP por medio de Telnet al puerto 25, enviamos varios mensajes al usuario creado, por ejemplo:

```

C:\> Telnet localhost
220 127.0.0.1 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)
HELO
250 Welcome [127.0.0.1], pleased to meet you
MAIL FROM: yo@localhost
250 Sender "yo@localhost" OK...
RCPT TO: usuario
250 Recipient "usuario" OK...
DATA
354 Enter mail, end with "." on a line by itself
Subject: Probando
Esto es una prueba
enviada desde Telnet.
Fin.
250 Message accepted for delivery. <n7wv45357djl8y.1902201813590127.0.0.1>
QUIT_

```

Por último, usando los comandos **POP3** por medio de Telnet al puerto **110**, recuperamos uno de los mensajes enviados al usuario. En primer lugar se envía el comando **USER** con el nombre de usuario, a continuación se envía **PASS** con la clave. El comando **STAT** nos muestra el número de mensajes del usuario y el tamaño. El comando **LIST** nos muestra la lista de mensajes (en este caso hay 1 mensaje), el comando **RETR 1** obtiene el mensaje con número 1:

```

C:\> Telnet localhost
+OK ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)
USER usuario
+OK Password required for usuario
PASS usuario
+OK Mailbox locked and ready
STAT
+OK 1 296
LIST
+OK
1 296
RETR 1
+OK 296 octets
Received: from [127.0.0.1] by Windows7UM
(ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)); Mon, 19 Feb 2018 13:59:03 +0100
Message-ID: <n7wv45357djl8y.1902201813590127.0.0.1>
Date: Mon, 19 Feb 2018 13:59:03 +0100

Subject: Probando
Esto es una prueba
enviada desde Telnet.
Fin.
QUIT
+OK Aba he

Se ha perdido la conexión con el host.
Presione cualquier tecla para continuar...

```

## 5.6 Comunicación desde Java con un servidor POP3

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

- **POP3Client**: implementa el lado 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.

La clase `POP3Client` presenta un único constructor. Algunos de los métodos que usaremos en los ejemplos son:

|                                                         |                                                                                                                                              |
|---------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <code>boolean deleteMessage(int messageId)</code>       | Elimina el mensaje con número <code>messageId</code> del servidor POP3. Devuelve <code>true</code> si la operación se realizó correctamente. |
| <code>POP3MessageInfo listMessage(int messageId)</code> | Lista el mensaje indicado en el parámetro <code>messageId</code> .                                                                           |

|                                                                     |                                                                                                                                                                                                      |
|---------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>POP3MessageInfo[] listMessages()</code>                       | Obtiene un array con información de todos los mensajes.                                                                                                                                              |
| <code>POP3MessageInfo listUniqueIdentifier(int message id)</code>   | Obtiene la lista de un único mensaje                                                                                                                                                                 |
| <code>boolean login(String userName, String password)</code>        | Inicia sesión en el servidor POP3 enviando el nombre de usuario y la clave. Devuelve <code>true</code> si la operación se realizó correctamente.                                                     |
| <code>boolean logout()</code>                                       | Finaliza la sesión con el servidor POP3. Devuelve <code>true</code> si la operación se realizó correctamente.                                                                                        |
| <code>Reader retrieveMessage(int messageId)</code>                  | Recupera el mensaje con número <code>messageId</code> del servidor POP3.                                                                                                                             |
| <code>Reader retrieveMessageTop(int messageId, int numLines)</code> | Igual que el anterior, pero sólo el número de líneas especificadas en el parámetro <code>numLines</code> . Para recuperar la cabecera del mensaje <code>numLines</code> debe de ser <code>0</code> . |

La clase `POP3SCient` presenta varios constructores, usar uno u otro dependerá de los datos que nos proporcione el servidor POP en el que tengamos nuestra cuenta de correo. En los ejemplos usaremos los más básicos, el constructor sin parámetros y el constructor con un parámetro booleano `POP3SCient(boolean implicit)`.

Si se selecciona el segundo constructor con el valor `true` para el parámetro (modo implícito), la negociación SSL/TLS comienza justo después de que se haya establecido la conexión. En modo explícito (constructor por defecto), la negociación SSL/TLS se inicia cuando el usuario llama al método `execTLS` y el servidor acepta el comando.

Ejemplo de uso en modo implícito:

```
POP3SCient c = new POP3SCient(true);
c.connect(server, 995);
```

Ejemplo de uso en modo explícito:

```
POP3SCient c = new POP3SCient();
c.connect(server, 110);
if (c.execTLS()) {
    /* resto de sentencias */
}
```

Los métodos de la clase `POP3SCient` son similares a los vistos para la clase `SMTPSCient`.

En el siguiente ejemplo nos conectamos al servidor POP3 local y visualizamos el número de mensajes de un usuario. Utilizamos el primer constructor (modo explícito) pero no se usa el método `execTLS` ya que no se necesita negociar el uso de TLS:

```
import java.io.IOException;
import org.apache.commons.net.pop3.POP3MessageInfo;
import org.apache.commons.net.pop3.POP3SCient;

public class Ejemplo1POP3 {
    public static void main(String[] args) {
        String server = "localhost", username = "usuario", password = "usuario";
        int puerto = 110;
        POP3SCient pop3 = new POP3SCient();
        try {
```

```

        pop3.connect(server, puerto);
        System.out.println("Conexión con el servidor POP3 establecida: " +
            server);
        if (!pop3.login(username, password))
            System.err.println("Error de inicio de sesión");
        else {
            POP3MessageInfo[] mensajes = pop3.listMessages();
            if (mensajes == null)
                System.err.println("No se han podido recuperar los mensajes");
            else
                System.out.println("Nº de mensajes: " + mensajes.length);
            pop3.logout();
        }
        pop3.disconnect();
    } catch (IOException e) {
        System.err.println(e.getMessage());
        e.printStackTrace();
        System.exit(1);
    }
    System.exit(0);
}
}

```

La ejecución muestra la siguiente información:

```

Conexión con el servidor POP3 establecida: localhost
Nº de mensajes: 2

```

`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 del mensaje.

En el siguiente ejemplo se recorre el array de mensajes y se visualiza información de los campos anteriores (`identifier`, `number` y `size`), se puede ver cómo varían sus valores al usar el array con la lista de mensajes y al usar el método `ListUniqueIdentifier`:

```

for (int i = 0; i < mensajes.length; i++) {
    System.out.println("Mensaje: " + (i + 1));
    POP3MessageInfo msginfo = mensajes[i];
    System.out.println("IDentificador: " + msginfo.identifier + ", Number: " +
        msginfo.number + ", Tamaño: " + msginfo.size);
    System.out.println("Prueba de listUniqueIdentifier: ");
    POP3MessageInfo pmi = pop3.listUniqueIdentifier(i + 1);
    System.out.println("\tIDentificador: " + pmi.identifier + ", Number: " +
        pmi.number + ", Tamaño: " + pmi.size);
}

```

Se visualiza:

```

Conexión con el servidor POP3 establecida: localhost
Mensaje: 1
IDentificador: null, Number: 1, Tamaño: 296

```

```
Prueba de listUniquelIdentifier:  
IDentificador: ekzc2lkb2so6hykg, Number: 1, Tamaño: -1
```

Para probarlo con una cuenta de GMail necesitaríamos el segundo constructor (modo implícito)

`POP3SClient c = new POP3SClient(true)`. Los datos que nos proporciona GMail son los siguientes:

- Servidor de correo entrante (POP3) - requiere SSL: pop.gmail.com
- Utilizar SSL: Sí
- Puerto: 995.

En el siguiente ejemplo se muestra solo la cabecera de un mensaje recogido del servidor POP de GMail, se realiza mediante el método `retrieveMessageTop(msginfo.number, 0)`, pasándole como primer parámetro el número de mensaje y como segundo el valor 0:

```
System.out.println("Cabecera del mensaje:");  
BufferedReader reader = (BufferedReader) pop3.retrieveMessageTop(msginfo.number, 0);  
String linea;  
while ((linea = reader.readLine()) != null)  
    System.out.println(linea.toString());  
reader.close();
```

#### PRÁCTICA 4

Realiza un programa Java en el que utilices el servidor POP3 de tu cuenta de correo para recoger los mensajes que tengas. Visualiza el número y el cuerpo de los mensajes. Los datos para conectarse al servidor POP3 (nombre servidor, usuario, clave y puerto) se introducen desde la línea de comandos.