

# COMUNICACIÓN CON UN SERVIDOR SMTP.

## Índice de contenido

1.COMUNICACIÓN CON UN SERVIDOR SMTP.....	2
1.1.INSTALACIÓN DE UN SERVIDOR DE CORREO ELECTRÓNICO. ....	2
1.2.USO DE TELNET PARA COMUNICAR CON EL SERVIDOR SMTP.....	4
1.3.COMUNICAR CON UN SERVIDOR SMTP CON JAVA.....	7
1.4.ACCESO A LOS MENSAJES DE UN SERVIDOR SMTP.....	15
1.5.USO DE TELNET PARA COMUNICAR CON EL SERVIDOR POP .....	16
1.6.COMUNICACIÓN CON UN SERVIDOR POP3 CON JAVA .....	18

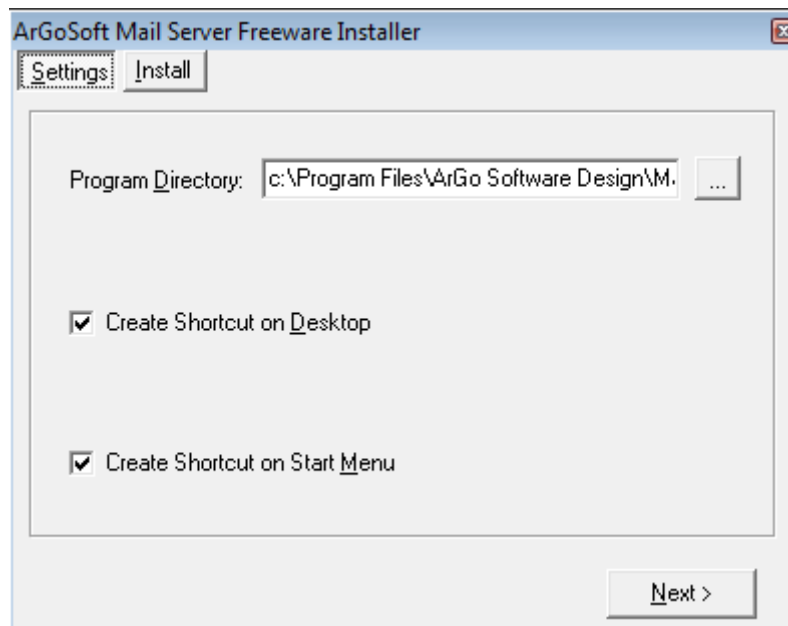
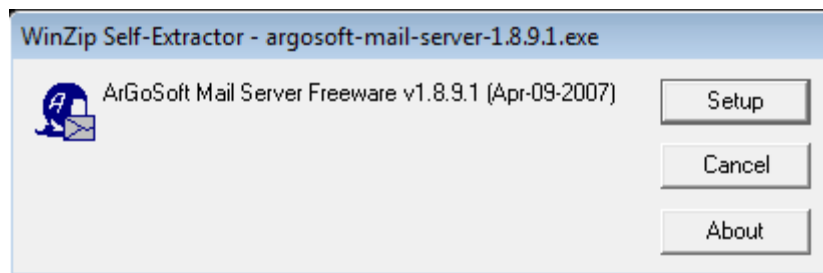
## 1. COMUNICACIÓN CON UN SERVIDOR SMTP

SMTP (*SimpleMail 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.

### 1.1. INSTALACIÓN DE UN SERVIDOR DE CORREO ELECTRÓNICO.

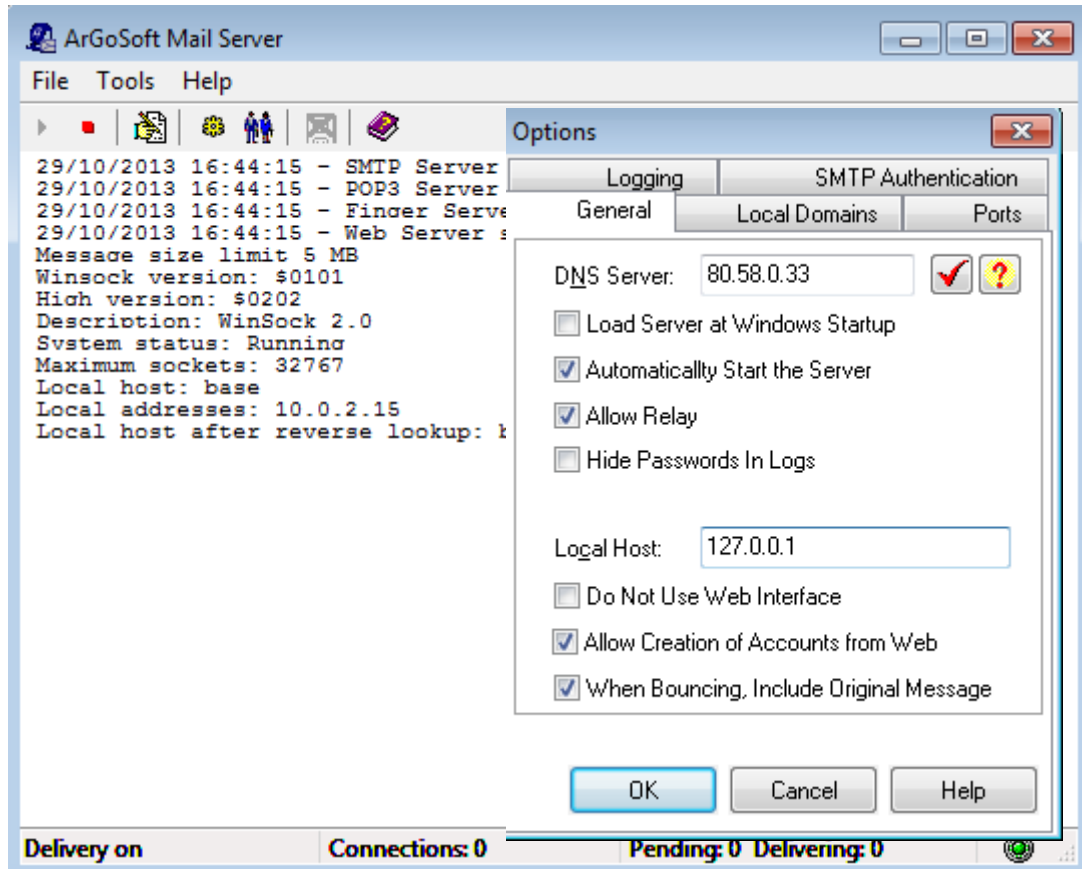
Un servidor SMTP es un programa que permite enviar correo electrónico a otros servidores SMTP. Vamos a ver cómo se instala un simple servidor SMTP en nuestro equipo local en el sistema operativo Windows 7. Descargamos el fichero *argosoft-mail-server-1.8.9.1.exe* (*ArGoSoft Mail Sever*) de la URL <http://argosoft-mailserver.uptodown.com/> y lo instalamos. Al ejecutar el fichero se muestra una ventana, pulsamos en el botón *Setup*. A continuación se elige la carpeta donde se instalará y se pulsa el botón *Next*, y a continuación al botón *Start Installation*.

Puede que se muestre la ventana de *Alerta de seguridad Windows* pidiéndonos confirmación para permitir acceso al programa a través del firewall de Windows, pulsamos en el botón *Permitir acceso* para continuar.



La siguiente pantalla muestra el progreso de la instalación. Al finalizar se muestra un mensaje indicando que la instalación se ha completado, se pulsa *Ok* y a continuación el botón *Finish*. El siguiente paso será configurar el servidor.

Hacemos doble clic en el icono creado en el escritorio (*ArGoSoft Mail Server*) para abrir el servidor, o seleccionamos el botón *Inicio* -> *Todos los programas* -> *ArGoSoft Mail Server* -> *ArGoSoft Mail Server*, se muestra la pantalla principal.



Desde esta pantalla se pulsa en el botón *Options*, o en la opción de menú *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 80.58.0.33, DNS de telefónica.
- **Automatically Start the Server:** marcamos esta casilla para que se inicie el servidor cuando abramos *ArGoSoft Mail*.
- **LocalHost:** como vamos a usar el servidor local escribimos 127.0.0.1

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 la opción de menú *Tools > View log file* se pueden consultar. **Aconsejable para ver todo lo que vamos haciendo.**

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, Eudora, 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.

## 1.2. 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:

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

Nos vamos a la línea de comandos del DOS 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, Version 1.8 (1.8.9.1)

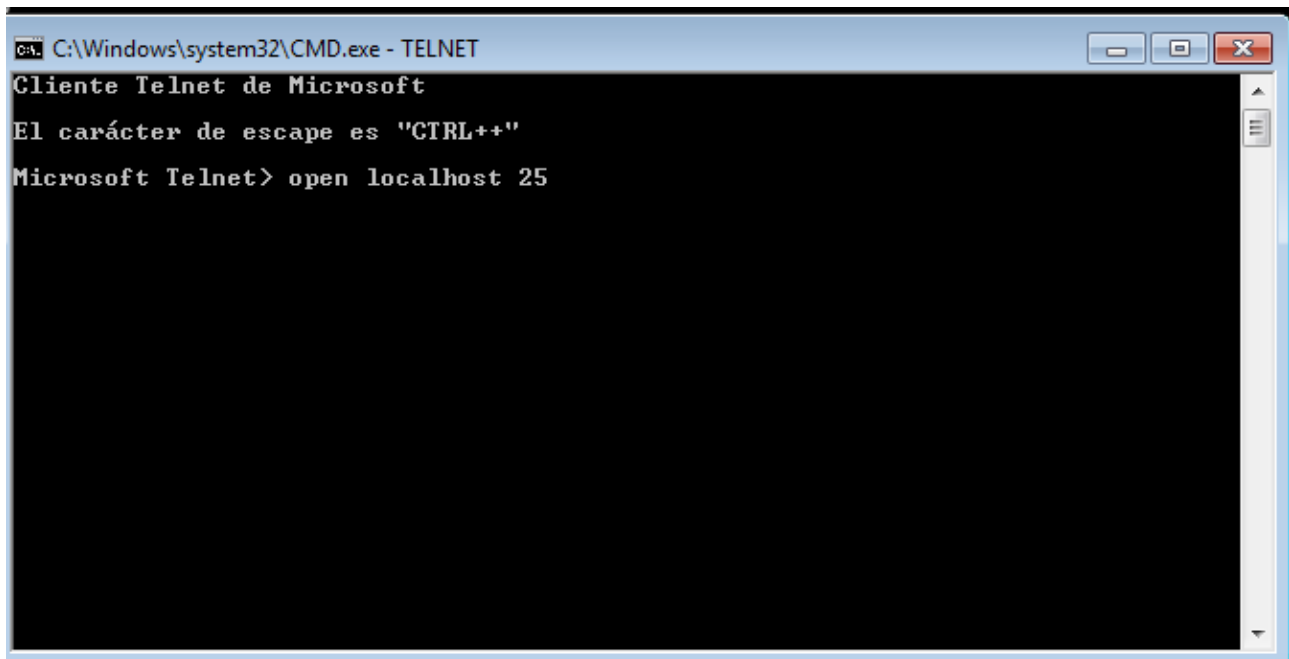
Normalmente responde con un número de 3 dígitos donde cada uno tiene un significado especial (como en FTP). Por ejemplo los números que empiezan en 2 (220,250, ...) indican que la acción se ha completado con éxito; los que empiezan por 3 (354) indican que el comando ha sido aceptado, pero la acción solicitada 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 *professmr2@gmail.com* que se usará como origen y destino:

```
HELO
250 Welcome [127.0.0.1], pleased to meet you
MAIL FROM: professmr2@gmail.com
250 Sender "professmr2@gmail.com" OK ...
RCPT TO: professmr2@gmail.com
250 Recipient "professmr2@gmail.com" OK ...
DATA
354 Enter mail, end with "." on a line by itself
from: professmr2@gmail.com
to: professmr2@gmail.com
subject: pruebas
Hola, esto es una prueba.
Adios.
250 Message accepted for delivery.
<s1012qpdOonfsjm.120420131025@127.0.0.1>
QUIT
```

**NOTA:** Si nos equivocamos y retrocedemos no va a funcionar, captura los caracteres de vuelta atrás, podemos ver lo que hace en los log del servidor ArGoSoft.

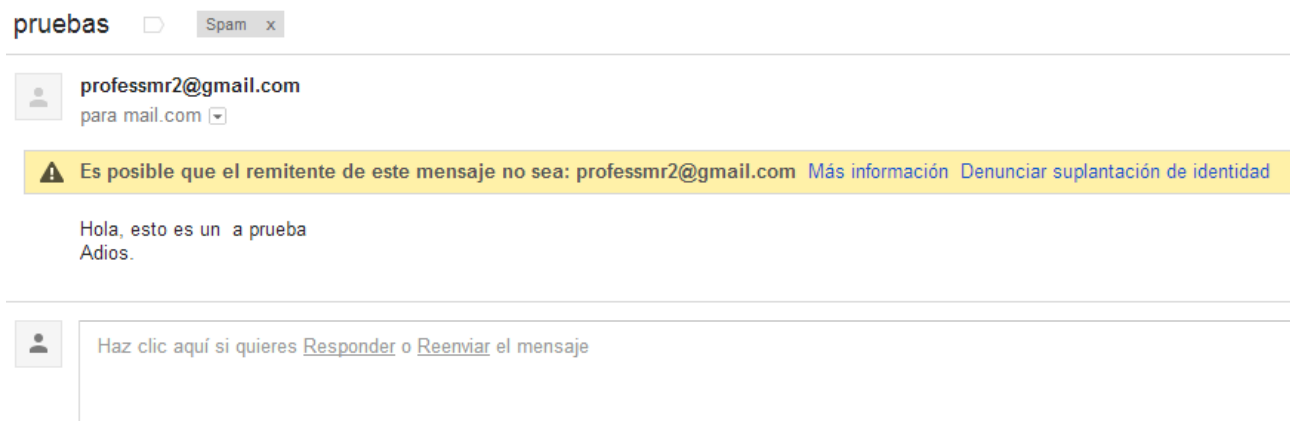


```
C:\ Telnet LOCALHOST
HELO
250 Welcome [127.0.0.1], pleased to meet you
MAIL FROM: professmr2@gmail.com
250 Sender "professmr2@gmail.com" OK...
RCPT TO: professmr2@gmail.com
250 Recipient "professmr2@gmail.com" OK...
DATA
354 Enter mail, end with "." on a line by itself
from: professmr2@gmail.com
to: professmr2@gmail.com
subject: pruebas

Hola, esto es una prueba
Adios
.
250 Message accepted for delivery. <c76y8nt2qnfvfmb.291020131746@127.0.0.1>
QUIT
```

Una vez realizados los pasos anteriores comprobamos si se ha recibido el correo.

Si no se recibió ni lo envió es porque el servidor de correo electrónico rechaza este tipo de envíos por seguridad, podemos ver que realmente lo envía en el log de nuestro servidor ArGoSoft Mail Server. Esta técnica era utilizada para enviar spam. [Mirar en la carpeta de spam, a ver si llegó.](#)



Con el comando DATA empieza el cuerpo del mensaje. Si en 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*, *Cc* y *From* (se pueden escribir en mayúsculas o minúsculas).

### 1.3. COMUNICAR CON UN SERVIDOR SMTP CON JAVA

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 **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 correctamente.

El siguiente ejemplo realiza una conexión al servidor SMTP local (por defecto se conecta al puerto 25, en el método *connect()* no es necesario indicarlo) y después se desconecta:

```
import java.io.IOException;
import org.apache.commons.net.smtp.*;

public class ClienteSMTP1 {
    public static void main(String[] args) {
        SMTPClient client =new SMTPClient();
        try {
            int respuesta;
            //NOS CONECTAMOS
            client.connect("localhost");//se conecta al puerto 25
            System.out.print(client.getReplyString());
            respuesta = client.getReplyCode();
            if (!SMTPReply.isPositiveCompletion(respuesta)){
                client.disconnect();
                System.err.println("CONEXION RECHAZADA. ");
                System.exit(1);
            }
            //REALIZAR ACCIONES
        }catch (IOException e) {
            if (client.isConnected()){
                try {
                    client.disconnect();
                }catch(IOException f) {
                    //no hago nada
                }
            }
            System.err.println("NO SE PUEDE CONECTAR AL SERVIDOR. ");
            e.printStackTrace();
            System.exit(1);
        }
        //NOS DESCONECTAMOS
        try {
            client.disconnect();
        }catch (IOException e) {
            System.err.println("ERROR AL DESCONECTAR DEL SERVIDOR.");
            e.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }
}
```

Para que funcione es necesario tener nuestro servidor de correo ArGoSoft Mail Server arrancado. La salida que se muestra al ejecutar el programa es:

```

C:\Windows\system32\cmd.exe

C:\Java\UD4>javac ClienteSMTP1.java
C:\Java\UD4>java ClienteSMTP1
220 127.0.0.1 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)
C:\Java\UD4>_

```

La clase **SMTPReply** (similar a **FTPReply**) 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:

CONSTRUCTOR	MISIÓN
<b>SMTPClient()</b>	Constructor por defecto
<b>SMTPClient(String codificacion)</b>	Se establece una codificación en el constructor ( <i>BASE64</i> , <i>BINARY</i> , <i>8BIT</i> , 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:

MÉTODOS	MISIÓN
<b>boolean addRecipient (String address)</b>	Añade la dirección de correo de un destinatario usando el comando RCPT
<b>boolean completePendingCommand()</b>	Este método se utiliza para finalizar la transacción y verificar el éxito o el fracaso de la respuesta del servidor
<b>boolean login()</b>	Inicia sesión en el servidor SMTP enviando el comando HELO
<b>boolean login(String hostName)</b>	Igual que la anterior pero envía el nombre del host como argumento
<b>boolean logout()</b>	Finaliza la sesión con el servidor enviando el comando QUIT
<b>Writer sendMessageData()</b>	Envía el comando DATA para después enviar el mensaje de correo. La clase <b>Writer</b> se usará para escribir secuencias de caracteres, como la cabecera y el cuerpo del mensaje
<b>boolean sendShortMessageData (String message)</b>	Método útil para envío de mensajes cortos
<b>boolean sendSimpleMessage (String remitente, String[] destinatarios, String message)</b>	Un método útil para el envío de un correo electrónico corto. Se especifica el remitente, los destinatarios y el mensaje



MÉTODOS	MISIÓN
<b>boolean sendSimpleMessage</b> (String remitente, String destinatarios, String message)	Igual que el anterior pero el mensaje sólo va dirigido a un destinatario
<b>boolean setSender</b> (String address)	Se especifica la dirección del remitente usando el comando MAIL
<b>boolean verify</b> (String username)	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(); /inicio de sesión -HELO
String destinatario="usuario@dominio.net";
String mensaje = "Hola. \nEnviando saludos.\nChao.";
String remitente="yo@localhost.es";
client.sendSimpleMessage(remitente, destinatario, mensaje);
client.logout() ;//final de sesión -QUIT

```

Si conseguimos enviar el correo y consultamos el correo recibido probablemente podremos ver que en *From* aparece *Invalid Address* y en *Subject* *[No Subject]* (aunque esto dependerá del servidor de correo que estemos usando).

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. To reduce the amount of spam sent to Gmail, this message has been blocked. Please review RFC 2822 specifications for more information. tc4si5253654pbc.4 - gsmtip"*. 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:

MÉTODOS	MISIÓN
<b>void addCC</b> (String address)	Agrega una dirección de correo electrónico a la lista CC
<b>void addHeaderField</b> (String headerField, String value)	Agrega un campo de encabezado arbitrario con el valor dado a la cabecera del artículo
<b>String toString</b> ()	Convierte el SimpleSMTPHeader a una cadena que contiene el encabezado con el formato

MÉTODOS	MISIÓN
	correcto, incluyendo la línea en blanco para separar la cabecera del cuerpo del correo

Vamos a crear nuestro ejemplo ClienteSMTP2.java a partir de ClienteSMTP1.java, incluyendo a continuación de la línea `//REALIZAR ACCIONES` el código que mostramos a continuación.

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, en este caso son 2:

```

client.login() ;//inicia sesión
String origen="professmr2@gmail.com";
String destino1="professmr2@gmail.com";
String destino2="tucuenta@gmail.com";
String asunto="Prueba de SMTPClient";
String mensaje = "Hola. \nEnviando saludos.\nChao.";

//se crea la cabecera
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);

```

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:

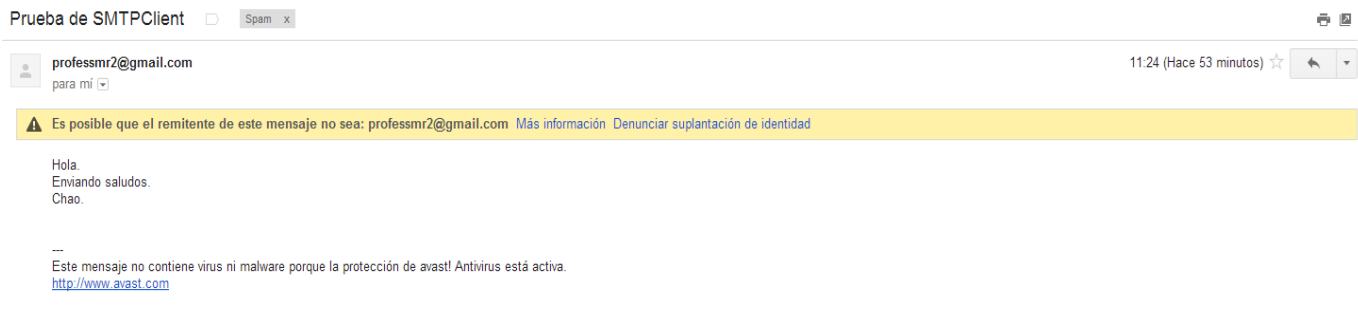
```

//se envia DATA
Writer writer = client.sendMessageData() ;
if(writer == null) {//fallo
    System.out.println("FALLO AL ENVIAR DATA.");
    System.exit(1);
}
writer.write(cabecera.toString()); //primero escribo cabecera
writer.write(mensaje); //luego mensaje
writer.close(); //se cierra stream
if(!client.completePendingCommand()) {//fallo
    System.out.println("FALLO AL FINALIZAR LA TRANSACCIÓN.");
    System.exit (1);
}
client.logout() ;//Finaliza sesión

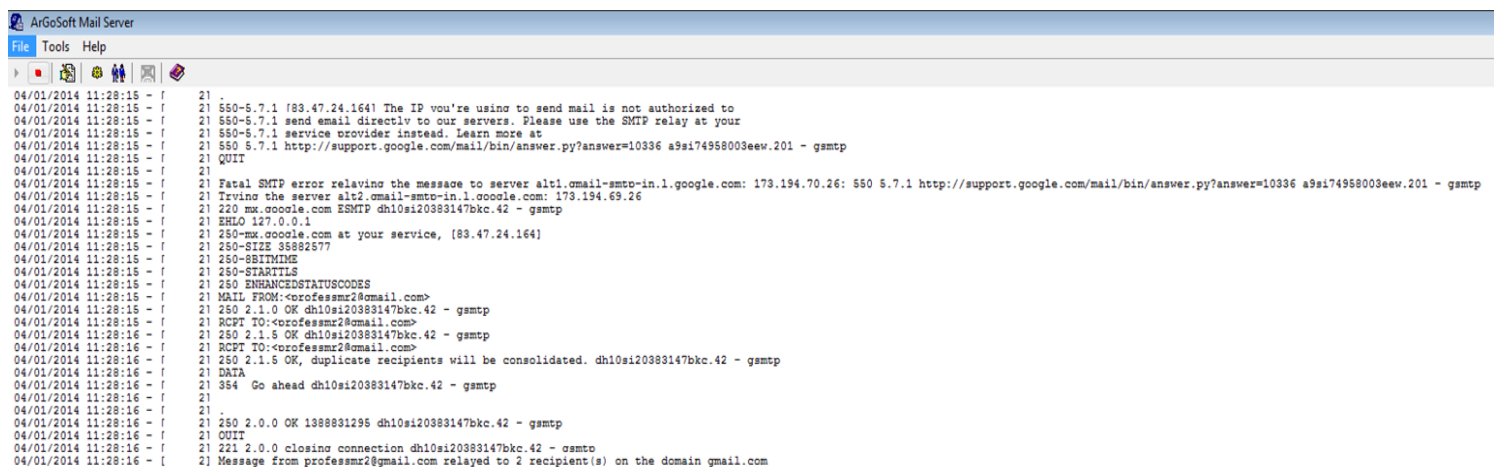
```

Debemos de comprobar en las dos cuentas de correo de los destinatarios que ha llegado el correo

(probablemente a la carpeta spam).



Si por alguna razón gmail no nos ha permitido enviar el correo veremos en la pantalla de nuestro servidor ArGoSoft Mail Server que realmente se envió y si ha fallado veremos los fallos que ha dado.



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 perjudiciales. 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 **SMTPClient**) con soporte de autenticación SMTP. La clase **SMTPClient** 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**.

La clase **SMTPClient** (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 **SMTPClient()** y el constructor con un parámetro booleano **SMTPClient(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. Por ejemplo: uso en modo implícito: *SMTPSClient e = new SMTPSClient(true); c.connect("127.0.0.1", 465);* Uso en modo explícito: *SMTPSClient e = new SMTPSClient(); c.connect("127.0.0.1", 25); if (c.execTLS()) { /resto de comandos aqui/}*.

Alguno de los métodos que se usan para realizar la autenticación SMTP son:

MÉTODOS	MISIÓN
<b>void setKeyManager (KeyManager newKeyManager)</b>	Para obtener el certificado para la autenticación se usa la interface <b>KeyManager</b> . Con este método se establece la clave para llevar a cabo la autenticación. Las <b>KeyManager</b> se pueden crear usando un <b>KeyManagerFactory</b>
<b>boolean execTLS()</b>	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 <i>true</i> si el comando y la negociación han tenido éxito

La clase **AuthenticatingSMTPClient** presenta varios constructores. Utilizaremos el constructor por defecto: *public AuthenticatingSMTPClient() throws NoSuchAlgorithmException*, 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:

MÉTODOS	MISIÓN
<b>boolean auth (AuthenticatingSMTPClient.AUTH_METHOD method, String username, String password)</b>	Se envía el comando AUTH con el método seleccionado para autenticarse en el servidor SMTP, se envía el nombre del usuario y su clave
<b>int ehlo(String hostname)</b>	Método para enviar el comando ESMTP ( <i>Extended SMTP</i> – 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 y 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.*;

public class ClienteSMTP3 {
    public static void main(String[] args) throws NoSuchAlgorithmException,
UnrecoverableKeyException, KeyStoreException, InvalidKeyException, InvalidKeySpecException{
    //se crea cliente SMTP seguro
    AuthenticatingSMTPClient client = new AuthenticatingSMTPClient();
    //datos del usuario y del servidor
    String server = "smtp.gmail.com";
    String username = "correo@gmail.com";
    String password = "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()); //Escribe por
//pantalla lo que recibe del servidor
//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 PLAIN (*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); // necesario
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 destino1 = "professmr2@gmail.com";
        String asunto = "Prueba de SMTPClient con GMAIL";
        String mensaje = "Hola. \nEnviando saludos.\nUsando GMAIL.\nChao.";
        //se crea la cabecera
        SimpleSMTPHeader cabecera = new SimpleSMTPHeader(username, destino1, asunto);
        //el nombre de usuario y el email de origen coinciden
        client.setSender(username);
        client.addRecipient(destino1);
        System.out.println("5 - " + client.getReplyString());
        //se envia DATA
        Writer writer = client.sendMessageData();
        if (writer == null) { // fallo
            System.out.println("FALLO AL ENVIAR DATA.");
            System.exit(1);
        }
        writer.write(cabecera.toString()); //cabecera
        writer.write(mensaje); //luego mensaje
        writer.close();
        System.out.println("6 - " + client.getReplyString());

        boolean exito = client.completePendingCommand();
        System.out.println("7 - " + client.getReplyString());
        if (!exito) { // fallo
            System.out.println("FALLO AL FINALIZAR TRANSACCIÓN.");
            System.exit(1);
        }
    } else
        System.out.println("USUARIO NO AUTENTICADO.");
} else
    System.out.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 envio.");
System.exit(0);
} //main
} // .. ClienteSMTP3
```

La salida que se debería generar al ejecutar el programa mostrando las respuestas que envía el servidor, si todo ha sido correcto es la siguiente:

1 - 220 mx.google.com ESMTP q13sm1822250wie.8 – gsmtip

2 - 250-mx.google.com at your service, [95.61.94.146]  
250-SIZE 35882577  
250-8BITMIME  
250-STARTTLS  
250 ENHANCEDSTATUSCODES

3 - 220 2.0.0 Ready to start TLS

4 - 235 2.7.0 Accepted

5 - 250 2.1.5 OK q13sm1822250wie.8 – gsmtip

6 - 354 Go ahead q13sm1822250wie.8 - gsmtip

7 - 250 2.0.0 OK 1366127119 q13sm1822250wie.8 – 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 quitar el *if (client.execTLS())*.

Otra forma de realizar un cliente SMTP es utilizando sockets

<http://tabasco.torreingenieria.unam.mx/gch/Curso%20de%20Java%20CD/Documentos/froufe/parte20/cap20-11.html>

o utilizando clases y métodos de la librería **javax.mail**. Si la utilizamos hay que acordarse de añadir su ruta en la variable de entorno CLASSPATH.

<http://translate.google.com/translate?hl=es&sl=en&u=http://www.mkymong.com/java/javamail-api-sending-email-via-gmail-smtp-example/&prev=/search%3Fq%3Dcliente%2Bsmtp%2Bcon%2Bjavamail%26sa%3DX%26hl%3Des%26biw%3D1745%26bih%3D855>

## 1.4. ACCESO A LOS MENSAJES DE UN SERVIDOR SMTP

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

- **MIME** (*Multipurpose Internet Mail Extensions* - extensiones multipropósito de correo de internet): 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* - protocolo de oficina de correo): 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.
- **IMAP** (*Internet Message Access Protocol* - protocolo de acceso a mensajes de Internet): 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:

COMANDOS	MISIÓN
<b>USER login</b>	A la derecha se escribe el login de la cuenta de correo
<b>PASS contraseña</b>	A la derecha se escribe la contraseña de la cuenta de correo
<b>STAT</b>	Muestra el número de mensajes de la cuenta
<b>LIST</b>	Listado de mensajes (numero - tamaño total del mensaje)
<b>RETR número-mensaje</b>	Obtiene el mensaje cuyo número coincida con el indicado a la derecha
<b>DELE número-mensaje</b>	Borra el mensaje indicado
<b>TOP número-mensaje n</b>	Muestra las 'n' primeras líneas del mensaje indicado

### 1.5. 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*.

Pulsamos en el botón *Add new*  *User*.

Rellenamos los campos (User Name: usu1, Real Name: Usuario1), pulsamos *OK* y después cerramos la ventana.



The image shows a 'Add New User' dialog box with two tabs: 'General' and 'Finger Information'. The 'General' tab is active. It contains the following fields and controls:

- User Name:** A text box containing 'usu1'.
- Real Name:** A text box containing 'Usuario1'.
- Password:** A text box containing '\*\*\*\*'.
- Confirm Password:** A text box containing '\*\*\*\*'.
- Forward Address:** An empty text box.
- Keep Copies:** A checkbox that is currently unchecked.
- Return Address:** An empty text box.

At the bottom of the dialog are three buttons: 'OK' (highlighted in blue), 'Cancel', and 'Help'.

Ahora usando los comandos SMTP por medio de Telnet al puerto 25, enviamos varios mensajes al usuario creado *usu1*, por ejemplo:

```
telnet localhost 25
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: professmr2@gmail.com
250 Sender "professmr2@gmail.com" OK ...
RCPT TO: usul
250 Recipient "usul" OK ...
DATA
354 Enter mail, end with " " on a line by itself
Subject: Probando
Esto es una prueba de un correo
de varias lineas
enviado desde Telnet.
Fin.
250 Message accepted for delivery.
<n8kewi5m24vtowr.17ü42ü13ü121@127.0.0.1>
QUIT
```

Ahora usando los comandos **POP3** por medio de Telnet al puerto 110, recuperamos uno de los mensajes enviados a *usu1*. 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 mensajes), el comando **RETR 1** obtiene el mensaje con número 1:

```
telnet localhost 110
+OK ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)
USER usu1
+OK Password required for usu1
PASS usu1
+OK Mailbox locked and ready
STAT
```

```

+OK
1 328
LIST
+OK
1 328
RETR 1
+OK 328 octets
Received: from [127.0.0.1] by base (ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)); Thu, 31 Oct 2013
11:04:23 +0100
Subject: Probando
Message-ID: <n8kewi5m24vtowr.170420130121@127.0.0.1>

Date: Thu, 31 Oct 2013 11:04:23 +0100
Esto es una prueba de un correo
de varias lineas
enviado desde Telnet.
Fin.
QUIT
+OK

```

## 1.6. COMUNICACIÓN CON UN SERVIDOR POP3 CON JAVA

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

- La clase **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 sus métodos son:

MÉTODOS	MISIÓN
<b>boolean deleteMessage (int messageId)</b>	Elimina el mensaje con número <i>messageId</i> del servidor POP3. Devuelve <i>true</i> si la operación se realizó correctamente
<b>POP3MessageInfo listMessage (int messageId)</b>	Lista el mensaje indicado en el parámetro <i>messageId</i>
<b>POP3MessageInfo[] listMessages ()</b>	Obtiene un array con información de todos los mensajes
<b>POP3MessageInfo listUniqueIdentifier (int messageId)</b>	Obtiene la lista de un único mensaje
<b>boolean login(String username, String password)</b>	Inicia sesión en el servidor POP3 enviando el nombre de usuario y la clave. Devuelve <i>true</i> si la operación se realizó correctamente
<b>boolean logout()</b>	Finaliza la sesión con el servidor POP3. Devuelve <i>true</i> si la operación se realizó correctamente
<b>Reader retrieveMessage(int messageId)</b>	Recupera el mensaje con número <i>messageId</i> del servidor POP3
<b>Reader retrieveMessageTOP(int messageId, int numLines)</b>	Igual que el anterior pero sólo el número de líneas especificadas en el parámetro <i>numLines</i> . Para recuperar la cabecera del mensaje <i>numLines</i> debe ser 0

La clase **POP3SClient** 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 se usan los más básicos, el constructor sin parámetros *POP3SClient()* y el constructor con un parámetro booleano *POP3SClient(boolean implicit)*.

Si se selecciona el segundo constructor con el parámetro a *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. Por ejemplo:

Uso en modo implícito: *POP3SClient e = new POP3SClient(true); c.connect(server, 995);*  
Uso en modo explícito: *POP3SClient c = new POP3SClient(); c.connect(server, 110); if (c.execTLS()) { //resto de comandos// } (como en **SMTPSClient**).*

Los métodos de la clase **POP3SClient** son similares a los vistos para la clase **SMTPSClient**.

En el siguiente ejemplo nos conectamos al servidor POP3 local y visualizamos el número de mensajes del usuario *usu1*, 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.POP3SCClient;

public class Ejemplo1POP3 {
    public static void main(String[] args) {
        String server = "localhost";
        String username="usul";
        String password = "usul";
        int puerto = 110;

        POP3SCClient pop3 = new POP3SCClient();

        try {
            //nos conectamos al servidor
            pop3.connect(server, puerto);
            System.out.println("Conexion realizada al servidor POP3 " + server);
            //iniciamos sesión
            if (!pop3.login(username, password))
                System.err.println("Error al hacer login");
            else{
                //obtenemos todos los mensajes en un array
                POP3MessageInfo[] men = pop3.listMessages();
                if (men == null)
                    System.out.println("Imposible recuperar mensajes.");
                else
                    System.out.println("Numero de mensajes: " + men.length) ;
                //finalizar sesión
                pop3.logout () ;
            }
            //nos desconectamos
            pop3.disconnect();
        }catch (IOException e) {
            System.err.println(e.getMessage());
            e.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }
}

```

La ejecución muestra siguiente información:

```

C:\Java\UD4>javac Ejemplo1POP3.java

C:\Java\UD4>java Ejemplo1POP3
Conexion realizada al servidor POP3 localhost
Numero de mensajes: 1

C:\Java\UD4>_

```

la

**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< men.length; i++) {
    System.out.println("Mensaje: " + (i+1));
    POP3MessageInfo msginfo =men[i]; //lista de mensajes
    System.out.println("IDentificador: " + msginfo.identifier + ", Number: " + msginfo.number +
        ",Tamaño: " + msginfo.size);

    System.out.println("Prueba de listUniqueIdentifier: ");
    POP3MessageInfo pmi = pop3.listUniqueIdentifier(i+1) //un mensaje
    System.out.println("\tIDentificador: " + pmi.identifier +
        ", Number: " + pmi.number + ", Tamaño: " + pmi.size);
}
```

Se visualiza:

```
Mensaje: 1
IDentificador: null, Number: 1, Tamaño: 328
Prueba de listUniqueIdentifier:
IDentificador: gtf2ud3gh74gwbmi, Number: 1, Tamaño: -1
```

En nuestro caso sólo tenemos un mensaje en el buzón, pero podemos probar a enviar más.

Para probarlo con una cuenta de gmail necesitaríamos el segundo constructor (modo implícito) *POP3SClient e = 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 fragmento de código 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:

```
//solo recupera cabecera
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();
```

Si queremos recuperar todo el mensaje habrá que escribir:

```
BufferedReader reader =(BufferedReader) pop3.retrieveMessage(msginfo.number);
```

Un ejemplo completo para nuestro servidor **ArGoSoft Mail Server** que permite conectarnos al servidor POP3 (puerto 110) y descargarnos los mensajes del usuario usu1, visualizando información acerca de los mensajes almacenados (POP3MessageInfo) y mostrando tanto su cabecera como el mensaje completo sería el siguiente:

```
import java.io.IOException;
import org.apache.commons.net.pop3.POP3MessageInfo;
import org.apache.commons.net.pop3.POP3SClient;
import java.io.*;

public class Ejemplo2POP3 {
    public static void main(String[] args) {
        String server = "localhost";
        String username="usu1";
        String password = "usu1";
        int puerto = 110;

        POP3SClient pop3 = new POP3SClient();

        try {
            //nos conectamos al servidor
            pop3.connect(server, puerto);
            System.out.println("Conexion realizada al servidor POP3 " + server);
            //iniciamos sesi6n
            if (!pop3.login(username, password))
                System.err.println("Error al hacer login");
            else{
                //obtenemos todos los mensajes en un array
                POP3MessageInfo[] men = pop3.listMessages();
                if (men == null)
                    System.out.println("Imposible recuperar mensajes.");
                else
                    System.out.println("Numero de mensajes: " + men.length) ;
            }
        }
    }
}
```

```

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

    //Muestro la cabecera del mensaje
    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());
    //Muestro todo el mensaje
    System.out.println("Mensaje completo:");
    reader = (BufferedReader) pop3.retrieveMessage(msginfo.number);
    while ((linea = reader.readLine()) != null)
        System.out.println(linea.toString());
    reader.close();
}

pop3.logout ();
}
//nos desconectamos
pop3.disconnect();
}catch (IOException e) {
    System.err.println(e.getMessage());
    e.printStackTrace();
    System.exit(1);
}
System.exit(0);
}
} // main
} // .. Ejemplo2POP3

```