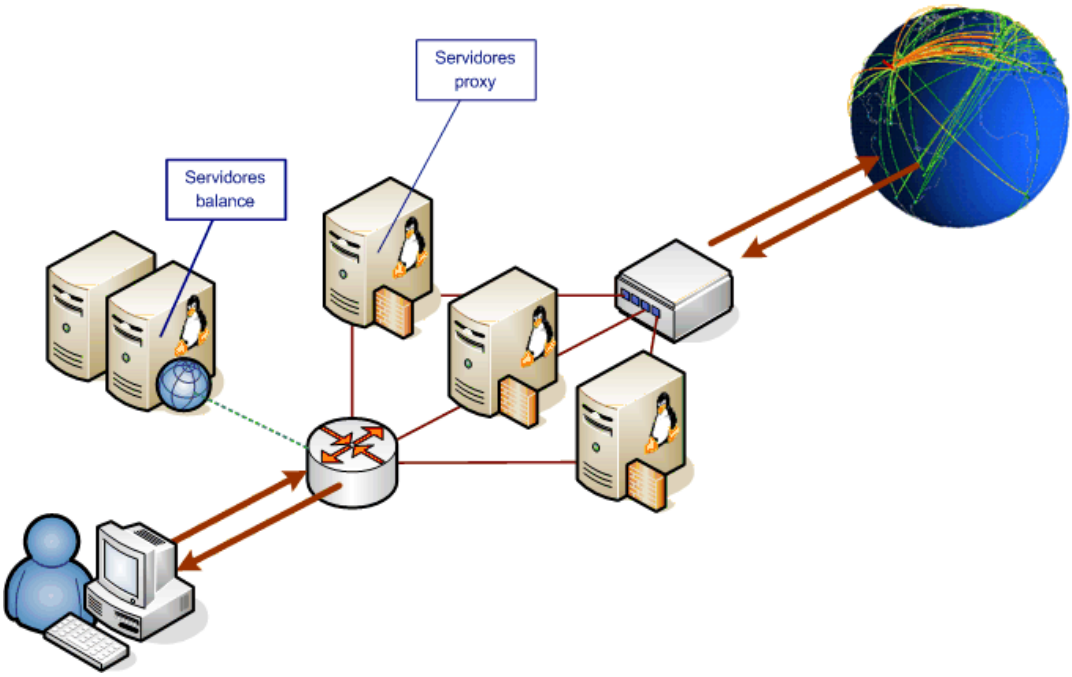
UD04: Programación de servicios de red



1. Introducción
2.
Protocolos estándar de comunicación en red a nivel de aplicación
2. 1. HTTP y HTTPS
2. 2. FTP
2. 3. SMTP
2. 4. IMAP
2. 5. POP3
2. 6. DNS
2. 7. TELNET
2. 8. SSH
2. 9. LDAP
2. 10. NFS
2. 11. SNMP
2. 12. DHCP
2. 13. SMB yCIFS
3. Clases y librerías para la creación de servicios en red
3. 1. Clases Java estándar
3. 1. 1. `java.net.URL`
3. 1. 2. `java.net.URLConnection`
3. 1. 3. `java.net.HttpURLConnection`
3. 1. 4. `java.net.http.HttpClient`
3. 1. 5. `java.net.http.HttpRequest`
3. 1. 6. `java.net.http.HttpResponse`
3. 1. 7. `JavaMail`
3. 2. Librerías `Apache Commons`
4. Comunicación mediante HTTP
4. 1. Peticiones HTTP basadas en `HttpUrlConnection`
4. 2. Peticiones HTTP basadas en `java.net.http`
5. Transferencia de ficheros mediante FTP

6. **Programación de envios y recepción de correos electrónicos**
7. **Programación distribuida**
8. **Ejemplos de la Unidad 04**
    8. 1. Ejemplo1
    8. 2. Ejemplo2
    8. 3. Ejemplo3
    8. 4. Ejemplo4
    8. 5. Ejemplo5
    8. 6. Ejemplo6
    8. 7. Ejemplo7
9. **Fuentes de información**

1. Introducción

Las redes de ordenadores (y de otros dispositivos) son una fuente increíble de oportunidades para desarrollar aplicaciones y servicios. Crear programas que se ejecuten de manera distribuida entre varios ordenadores, obtener información de páginas y servicios web, transferir ficheros o enviar correos electrónicos son algunos de los más interesantes ejemplos.

Las tecnologías y protocolos en los que se basa internet son de uso libre, lo que implica que se pueden utilizar desde cualquier lenguaje de programación. Además, mediante el uso de las clases y librerías adecuadas, la programación de aplicaciones que haga uso de dichas tecnologías es relativamente sencillo y rápido. En consecuencia, el desarrollo de aplicaciones que aprovechen el potencial de las redes es una tarea altamente atractiva.

En esta unidad se explican las técnicas de programación de los servicios más populares fundamentados en el uso de las redes basadas en las tecnologías de internet, así como las librerías y clases más convenientes para llevarlas a cabo.

2. Protocolos estándar de comunicación en red a nivel de aplicación

Los modelos de red OSI y TCP/IP se estructuran como una sucesión de capas responsables de toda la actividad relacionada con la comunicación entre dispositivos a través de una red. Cada capa tiene un nivel de abstracción distinta, dependiendo de la tarea que tenga que realizar.

Las capas inferiores son mucho más técnicas y áridas de programar que las superiores, ya que se encargan de detalles mucho más específicos relativos a la comunicación. Los programas de las capas inferiores se encuentran en los controladores de las tarjetas de red y en los sistemas operativos.

Las capas superiores son las que se utilizan en la creación de las aplicaciones que utilizan los usuarios. De hecho, la capa superior se denomina «de aplicación» tanto en el modelo teórico OSI como en el modelo TCP/IP utilizado en internet.

Por lo tanto, la capa de aplicación del modelo de red TCP/IP contiene las aplicaciones y los servicios que puede utilizar el usuario. Es la capa más próxima a las personas y sobre la que se van a desarrollar las aplicaciones de uso cotidiano.

Dentro de la capa de aplicación se ubican varios protocolos, cada uno de los cuales tiene un cometido específico. El número de protocolos es extenso por lo que solo se presentan en esta unidad aquellos más relevantes.

2.1. HTTP y HTTPS

El Protocolo de Transferencia de Hipertexto (HTTP o Hypertext Transfer Protocol) es el utilizado para la transmisión de documentos a través de internet. HTTP es el protocolo fundamental de la web, ya que permite la transferencia de las peticiones y de los recursos que intercambian clientes y servidores. Se basa en el protocolo TCP para su funcionamiento. Por su parte, el Protocolo Seguro de Transferencia de Hipertexto (HTTPS o Hypertext Transfer Protocol Secure) es la versión segura del protocolo HTTP, ya que cifra toda la información intercambiada entre cliente y servidor.

No es necesario conocer en profundidad el protocolo HTTP para desarrollar aplicaciones que realicen comunicaciones en red, pero sí hay algunos datos técnicos que conviene tener presentes.

Uno de los aspectos técnicos que resulta necesario conocer de HTTP es el referente a los tipos de peticiones que se pueden realizar. Se conoce como método y son los que se muestran en la siguiente tabla:

Método	Descripción
`GET`	Solicita la recuperación de un recurso. Una petición HTTP que utiliza este método recupera una entidad alojada en el servidor.
`POST`	Este método se utiliza para crear una entidad alojada en el servidor.
`PUT`	Este método se utiliza para crear o actualizar una entidad alojada en el servidor si esta ya existía previamente.
`DELETE`	Elimina una entidad alojada en el servidor.

Otro de los elementos de HTTP que resulta fundamental conocer es el referente a los códigos de respuesta. Cuando se realiza una petición HTTP a un servidor este genera una respuesta que va identificada con un código. Este código proporciona información sobre el resultado del tratamiento de la petición, permitiendo saber si esta se ha procesado correctamente o ha surgido algún inconveniente. Estos códigos se forman por tres dígitos, siendo el primero el que determina a nivel global el tipo de respuesta. En la siguiente tabla se muestran dichos grupos.

Códigos de retorno	Descripción
100-199	Respuesta informativa.
200-299	Éxito.
300-399	Redirección.
400-499	Error del cliente.
500-599	Error del servidor.

De los códigos contenidos en estos grupos, algunos de los más frecuentes son 200 (la solicitud ha tenido éxito) y 404 (el recurso solicitado no ha sido encontrado). En las aplicaciones que realizan peticiones HTTP hay que evaluar el código devuelto por estas para saber si se han procesado correctamente o ha habido algún problema.

Por defecto HTTP utiliza el puerto 80 mientras que HTTPS utiliza el 443.

2.2. FTP

El Protocolo de Transferencia de Ficheros (FTPo File Transfer Protocol) permite la transferencia entre dispositivos de ficheros de todo tipo. Se basa en una arquitectura cliente-servidor y utiliza el protocolo TCP.

FTP utiliza el puerto 21 por defecto.

2.3. SMTP

El Protocolo Simple de Transferencia de Correo (SMTP o Simple Mail Transfer Protocol) es el usado para el envío de correos electrónicos. Utiliza la especificación MIME (Multipurpose Internet Mail Extensions o extensiones multipropósito de correo de internet) que permite el intercambio de todo tipo de archivos a través de internet.

Por defecto utiliza el puerto 25.

2.4. IMAP

El Protocolo de Acceso a Mensajes de Internet (IMAP o Internet Message Access Protocol) es el utilizado para la recepción de correos electrónicos. IMAP almacena los correos en el servidor por lo que se pueden leer desde diferentes dispositivos.

Utiliza el puerto 143 para las conexiones sin cifrar y el puerto 993 para las cifradas.

2.5. POP3

El Protocolo de Oficina de Correo (POP3 o Post Office Protocol) es, al igual que IMAP, el utilizado para la recepción de correos electrónicos. POP3 descarga los correos y los elimina del servidor, por lo que una vez consultados no podrán ser accedidos desde otros dispositivos distintos al del primer acceso.

Utiliza el puerto 110 para las conexiones sin cifrar y el puerto 995 para las cifradas.

2.6. DNS

El Sistema de Nombres de Dominio (DNS o Domain Name System) permite la asociación de los nombres de los dispositivos conectados a la red con sus direcciones IP. Es el protocolo que permite que una URL o una dirección de correo se vincule con la dirección IP en la que se encuentra ubicado el servicio relacionado.

El puerto por defecto utilizado por este protocolo es el 53.

2.7. TELNET

Este protocolo permite acceder a través de un terminal (sin gráficos) a un ordenador remoto. No cifra la información enviada y recibida, por lo que se considera inseguro.

Telnet utiliza el puerto 23.

2.8. SSH

SSH (Secure Shell) es un protocolo con un objetivo similar al de Telnet, ya que permite acceder de manera remota a un ordenador a través de una terminal. En este caso, la comunicación es cifrada por lo que se considera un protocolo seguro.

Por defecto utiliza el puerto 22.

2.9. LDAP

El Protocolo Ligero de Acceso a Directorios (LDAP o Lightweight DirectoryAccess Protocol) proporciona una estructura jerárquica, ordenada y distribuida de información, así como las herramientas de acceso a la misma. Se suele utilizar para almacenar la información referente al acceso a los sistemas (credenciales y permisos).

El puerto utilizado por defecto es el 389.

2.10. NFS

El Sistema de Archivos en Red (NFS o Network File System) permite distribuir ficheros en una red y acceder a ellos desde cualquier nodo de esta.
Utiliza el puerto 2049.

2.11. SNMP

El Protocolo Simple de Administración de Red (SNMP o Simple Network Management Protocol) proporciona la capacidad de intercambiar información entre los dispositivos de una red que forman parte de su infraestructura (enrutadores, conmutadores, etc.), teniendo una vocación claramente orientada a la administración de las redes.

Utiliza el puerto 161.

2.12. DHCP

El Protocolo de Configuración Dinámica de Host (DHCP o Dynamic Host Configuration Protocol) es el encargado de la asignación dinámica de las direcciones IP y de los parámetros de configuración a los dispositivos que forman parte de una red.

Cuando en una red las direcciones no son fijas, el servicio DHCP se encarga de asignar a cada dispositivo una dirección IP de una lista de direcciones disponibles, permitiendo una gestión dinámica de dichas direcciones.

Este protocolo utiliza el puerto 67 por defecto.

2.13. SMB yCIFS

Estos dos protocolos permiten compartir recursos en una red, tales como ficheros o impresoras. SMB (Server Message Block) y CIFS (Common Internet File System) están muy relacionados entre sí, ya que CIFS es una implementación de SMB creada por Microsoft.

En la actualidad, la recomendación es utilizar SMB 2 y SMB 3 como implementaciones de este protocolo.
SMB yCIFS utilizan por defecto los puertos 139 y 445.

3. Clases y librerías para la creación de servicios en red

Prácticamente todos los lenguajes de programación modernos disponen de librerías para la programación de aplicaciones que hacen uso de la red. Hay que tener de en cuenta que, en ausencia de dichas librerías, la programación seria a un nivel bastante bajo, requiriendo la utilización de sockets, transferencia de bytes y conocimiento exhaustivo de protocolos.

Java dispone de manera nativa de un buen conjunto de librerías y de clases que dan soporte al desarrollo de este tipo de aplicaciones. Además, otros desarrolladores proporcionan librerías y clases que mejoran o complementan las propias del lenguaje.

3.1. Clases Java estándar

Las clases disponibles de manera nativa en Java proporcionan toda la funcionalidad necesaria para realizar aplicaciones en red. En este apartado se presentan las más relevantes.

3.1.1. `java.net.URL`

Representa una URL (Uniform Resource Locator), una referencia a un recurso de la web.

En el siguiente código se muestra la construcción de un objeto de esta clase.

```
1 URL url = new URL("http://www.martinezpenya.es");
```

Si la dirección web indicada en el constructor no está correctamente formada, lanzará la excepción `MalformedURLException`.
El método más importante se expone en la siguiente tabla:

Método	Descripción
`openConnection`	Proporciona una conexión (objeto `URLConnection`) a partir del recurso representando en la URL.

3.1.2. `java.net.URLConnection`

Esta clase abstracta es la superclase de todas las clases que representan un enlace de comunicación entre la aplicación y una URL.

Las instancias de esta clase permiten leer y escribir en el recurso referenciado por la URL.

`HttpURLConnection` es la subclase más relevante.

Los métodos más importantes se muestran en formato Tabla a continuación:

Método	Descripción
`getByName`	Método estático que proporciona la dirección IP de un hosta partir de su nombre.
`getLocalHost`	Método estático que proporciona la dirección IP del host local.
`getHostAddress`	Proporciona la dirección IP del host.
`getHostName`	Proporciona el alias del host.
`getAddress`	Proporciona la dirección IP del host como un array de bytes.
`getCanonicalHostName`	Proporciona el nombre del host.
`getInputStream`	Proporciona un stream de lectura.
`getOutputStream`	Proporciona un stream de escritura.
`setRequestProperty`	Asigna el valor a una propiedad.

El siguiente código de ejemplo establece una conexión con un recurso de la web y se lee su contenido:

```
URL url = new URL("http://www.martinezpenya.es”);
URLConnection conexionURL = url.openConnection();
InputStream is = conexionURL.getInputStream();
int c;
while ((c=is.read())!=-1) {
    system.out .print ((char) c);
}
is.close();
```

Clase disponible desde la versión 1.0 de Java.

3.1.3. `java.net.HttpURLConnection`

Esta clase proporciona los mecanismos para gestionar una conexión HTTP.

En la siguiente tabla se muestran los métodos principales. Se debe tener en cuenta heredar de `URLConnection` dispone además de sus mismos métodos.

Método	Descripción.
`disconnect`	Desconecta la conexión.
`getResponseCode`	Proporciona el código de retorno HTTP enviado por el servidor.
`setRequestMethod`	Proporciona el método de petición.

Contiene también las constantes que representan los códigos de estado del protocolo HTTP, como, por ejemplo, `HttpURLConnection.HTTP_OK` que tiene el valor entero 200 y que significa que la petición se ha realizado correctamente (OK).

Clase disponible desde la versión 1.1 de Java.

3.1.4. `java.net.http.HttpClient`

Esta clase abstracta permite realizar peticiones HTTP y obtener sus respuestas. Las instancias deben ser creadas a través de un builder u objeto de instanciación.

Algunos de los métodos más relevantes son los que se muestran en la siguiente tabla:

Método	Descripción
`newBuilder`	Crea un builder (objeto de la interfaz `HttpClient.Builder`
`send`	Envía la petición HTTP y devuelve una instancia de `HttpResponse`. Recibe como parámetro, además de la petición, un objeto de la clase `HttpResponse.BodyHandlers`, encargado de gestionar el contenido de la respuesta de la petición.

Los métodos de `HttpClient.Builder` se exponen a continuación:1

Método	Descripción
`build`	Proporciona el objeto `HttpClient` con la configuración proporcionada.
`followsRedirect`	Proporciona mecanismos para determinar cómo debe comportarse la petición frente a las redirecciones del servidor.
`version`	Permite especificar la versión del protocolo HTTP.

Clase disponible desde la versión 11 de Java.

3.1.5. `java.net.http.HttpRequest`

Clase abstracta que representa una petición HTTP. Las instancias se configuran y crean a través de un constructor o builder. Este constructor se obtiene a través del método estático newBuilder de la propia clase al que se le indicarán el método de HTTP a utilizar, los parámetros de la petición o el tiempo límite de espera entre otros parámetros de configuración.

El método principal es:

Método	Descripción
`newBuilder`	Método estático que crea un builder (objeto de la interfaz `HttpRequest.Builder`).

Por su parte, la interfaz HttpRequest.Builder proporciona los métodos siguientes:

Método	Descripción
`build`	Proporciona el objeto `HttpClient` con la configuración proporcionada.
`DELETE`	Asigna el método `DELETE` al builder.
`GET`	Asigna el método `GET` al builder.
`header`	Permite añadir un par parámetro-valor a la petición.
`headers`	Permite añadir pares de parámetro-valor a la petición.
`POST`	Asigna el método `POST` al builder.
`PUT`	Asigna el método `PUT` al builder.
`setHeader`	Permite asignar un par clave-valor a la petición.
`timeout`	Permite determinar un tiempo límite (`timeout`) para la petición.
`uri`	Asigna la `URI` a la petición.
`version`	Permite especificar la versión del protocolo HTTP.

Clase disponible desde la versión 11 de Java.

3.1.6. `java.net.http.HttpResponse`

Interfaz que representa la respuesta de una petición HTTP. Las instancias de esta interfaz no se crean directamente, sino que son proporcionadas por el método send de la clase HttpClient.

El método más importante es:

Método	Descripción
`statusCode`	Proporciona el código de estado de la petición HTTP.

La clase `HttpResponse` permite obtener instancias de la interfaz funcional `HttpResponse.BodyHandler`. Dichas instancias son utilizadas como parámetros en la llamada al método `send` de `HttpClient` y determinan la manera en la que va a procesarse el cuerpo de la respuesta de la petición HTTP.

Dispone de diversos métodos estáticos para la obtención de las instancias de `BodyHandler`, siendo los más relevantes:

Método	Descripción
`ofByteArray`	Método estático. Devuelve un objeto de tipo `BodyHandler<byte[]>`.
`ofFile`	Método estático. Devuelve un objeto de tipo `BodyHandler<Path>`.
`ofInputStream`	Método estático. Devuelve un objeto de tipo `BodyHandler<InputStream>`.
`ofString`	Método estático. Devuelve un objeto de tipo `BodyHandler<String>`.

La interfaz `BodyHandler` solo dispone de un único método, `apply`, que se invoca de manera automática cuando se utiliza. En función del tipo de clase utilizado este método realizará una u otra acción.

Por ejemplo, tras la ejecución del siguiente código, una llamada a `response.body()` proporciona un array de bytes:

```
1 HttpResponse<byte[]> response = httpClient.send(request,
  HttpResponse.BodyHandlers.ofByteArray());
```

En cambio, la misma llamada al método `body()` sobre el objeto response obtenido en la siguiente ejecución, proporciona un `String`:

```
1 HttpResponse<String> response = httpClient.send(request, HttpResponse.BodyHandlers.ofString);
```

Por último, la ejecución de la siguiente sentencia almacena en el fichero indicado en la variable `path` el contenido del cuerpo de la respuesta.

```
1 HttpResponse<Path> response = httpClient.send(request,
  HttpResponse.BodyHandlers.ofFile(Path.of(path)));
```

Clase disponible desde la versión 11 de Java.

3.1.7. `JavaMail`

El API `JavaMail` proporciona un entorno independiente del protocolo y de la plataforma para la creación de aplicaciones de mensajería a través del correo electrónico.

Es un paquete incluido en la versión empresarial de Java (Java EE o Java Enterprise Edition) que opcionalmente se puede incluir en proyectos desarrollados con la versión estándar (Java SE o Java Standard Edition).

Algunas de las clases más importantes de esta librería son:

Clase	Descripción
`com.sun.mail.imap.IMAPFolder`	Representa una carpeta de mensajes IMAP. Hereda de `javax.mail.Folder`.
`com.sun.mail.pop3.POP3Folder`	Representa una carpeta de mensajes POP3. Hereda de `javax.mail.Folder`.
`javax.mail.Address`	Clase abstracta que representa una dirección de correo electrónico.
`javax.mail.Folder`	Clase abstracta que representa una carpeta de mensajes.
`javax.mail.Message`	Clase abstracta que representa un mensaje de correo electrónico.
`javax.mail.Message.Reciplentiype`	Clase estática interna. Define los tipos de destinatario permitido. Los tipos definidos son TO, CC y BCC.
`javax.mail.Multipart`	Clase abstracta que representa un contenedor que admite múltiples partes del cuerpo del mensaje (body).
`javax.mail.Session`	Representa una sesión de correo electrónico.
`javax.mail.Store`	Clase abstracta que representa un almacén de mensajes y su protocolo de acceso. Las instancias de esta clase se obtienen invocando al método `getStore` de un objeto `Session`.
`javax.mail.Transport`	Clase abstracta que representa el método de transporte de los mensajes. Las instancias de esta clase se obtienen invocando al método `getTransport` de un objeto `Session`.
`javax.mail.internet.InternetAddress`	Representa una dirección de correo de internet usando el estándar RFC822. Hereda de `javax.mail.Address`.
`javax.mail.internet.MimeBodyPart`	Representa una parte del cuerpo (`body`) de un mensaje MIME.
`javax.mail.internet.MimeMessage`	Representa un mensaje de correo electrónico que utiliza la convención MIME. Hereda de la clase `javax.mail.Menssage`.
`javax.mail.internet.MimeMultipart`	Implementación de `javax.mail.Multipart` que utiliza la convención MIME

3.2. Librerías `Apache Commons`

Apache Commons es un conjunto de librerías Java de código abierto creadas en el ámbito de la Apache Software Foundation, una organización sin ánimo de lucro dedicada al desarrollo de software.

Las librerías de `Apache Commons` cubren diversos aspectos del desarrollo de software a través de componentes dedicados a temas tan dispares como la gestión de trazas (logs), las operaciones matemáticas y estadísticas o la generación de números aleatorios, por citar algunos ejemplos.

En lo referente a la generación de servicios en red, las librerías más relevantes son `Commons Email` para el envío y recepción de correos electrónicos y `Apache Commons Net` con implementaciones de los protocolos de internet.

Estas librerías proporcionan implementaciones alternativas a las disponibles en el lenguaje Java.

4. Comunicación mediante HTTP

La generación de servicios basados en el protocolo HTTP tiene dos perspectivas, correspondientes al cliente y al servidor. En este libro se aborda la perspectiva cliente, aprendiendo a desarrollar aplicaciones en Java que sean capaces de realizar peticiones HTTP para obtener información proporcionada por sitios o servicios web. La otra perspectiva proporciona la capacidad de desarrollar aplicaciones y servicios web, pero queda fuera
del ámbito de este libro.

Debido a cuestiones relacionadas con las licencias y la evolución de las versiones de Java, existen dos modelos para la realización de peticiones HTTP: el utilizado hasta la versión 1.8 de Java y el incorporado a partir de la versión 11. En esta unidad se van a presentar ambas opciones.

4.1. Peticiones HTTP basadas en `HttpUrlConnection`

Hasta la versión 1.8 de Java, la clase HttpURLConnection del paquetejava.net ha sido la base de la programación de aplicaciones capaces de acceder a recursos de la web.

Los pasos para realizar con esta clase una petición HTTP sin parámetros son los siguientes:

- Creación de URL.
- Apertura de la conexión.
- Configuración de la conexión:
    - Método HTTP.
    - Tipo de contenido.
    - Sistema de codificación.
    - Agente de usuario a utilizar.
- Obtención y evaluación del código de respuesta HTTP. Si el código es 200:
    - Obtención del objeto InputStream para lectura (si aplica).
        - Lectura del stream.
    - Obtención del objeto OutputStream para escritura (si aplica).
        - Escritura en el stream.
    - Desconexión.

Observa el Ejemplo1 y el Ejemplo2

4.2. Peticiones HTTP basadas en `java.net.http`

A partir de la versión 11 de Java, se introdujo el paquetejava.net.http para proporcionar una alternativa más potente, sencilla y actualizada de realizar peticiones HTTP (admite HTTP/1.1, HTTP/2 y WebSockets).

> Los WebSockets permiten establecer una comunicación bidireccional basada en eventos entre un servidor web y un navegador.

Para la realización de las peticiones HTTP convencionales se utilizan principalmente tres classes:

- `HttpClient`

- `HttpRequest`
- `HttpResponse`

Para realizar una petición HTTP se deben realizar los siguientes pasos:

- Crear el objeto `HttpClient`, indicando versión del protocolo, así como otros datos opcionales como el comportamiento en caso de que existan redirecciones del servidor.
- Crear el objeto `HttpRequest`, indicando la URI y los parámetros de la cabecera de la petición.
- Realizar la petición a través del método send del `HttpClient` y asignar la respuesta de la petición a un objeto `HttpResponse`.
- Procesar la respuesta.

Observa el Ejemplo3

5. Transferencia de ficheros mediante FTP

En Java, de manera nativa, es posible realizar transferencias de ficheros mediante este protocolo, pero es sumamente árido.

La librería Apache Commons Net proporciona clases y utilidades para realizar cualquier operación sobre un servidor FTP o FTPS desde un cliente Java.

Esta librería se puede descargar desde la web de apache.org a través del siguiente enlace: https://commons.apache.org/proper/commons-net/

Las clases principales se encuentran en el paquete `org.apache.commons.net.ftp` y se muestran a continuación.

Clase	Descripción
`FTP`	Proporciona la funcionalidad necesaria para implementar un cliente FTP. Incluye constantes para indicar el tipo de ficheros que se van a transmititr y la configuración de estos.
`FTPClient`	Subclase de FTP, encapsula la funcionalidad necesaria para subir y descargar ficheros través del protocolo FTP.
`FRPSClient`	Subclase de FTPClient, permite utilizar el protocolo FTP sobre SSL.
`FTPFile`	Representa información sobre los ficheros almacenados en el servidor.
`FTPReply`	Almacena las constantes que representan los códigos de retorno del protocolo FTP.

La clase `FTPClient` es la que proporciona los métodos de comunicación con el servidor, permitiendo realizar todas las operaciones que admite el protocolo FTP. Los métodos más comunes son:

Método	Descripción
`connect`	Permite establecer la conexión con un servidor a partirde su nombre de host puerto (método heredado de org.apache.commons.net.SocketClient). ysu
`changeToParentDirectory`	Cambia el directorio de trabajo del servidor al directorio padre del actual.
`changeWorkingDirectory`	Cambia el directorio de trabajo del servidor.
`deleteFile`	Borra un fichero en el servidor.
`disconnect`	Cierra la conexión con servidor FTP.
`getReplyCode`	Proporciona el código de respuesta de una petición FTP (método heredado de `org.apache.commons.net.ftp.FTP`).
`listDirectories`	Proporciona los directorios existentes en el directorio de trabajo del servidor.
`listFiles`	Proporciona los ficheros existentes en el directorio de trabajo del servidor.
`login`	Permite acceder al servidor usando una cuenta de usuario.
`logout`	Desconecta la cuenta del usuario validado.
`makeDirectory`	Crea un directorio en el servidor.
`rename`	Renombra un fichero en el servidor.
`retrieveFile`	Descarga un fichero del servidor al cliente.
`setFileType`	Permite indicar el tipo de fichero que se va a transferir.
`storeFile`	Sube un fichero desde el cliente al servidor.

Las cuentas de usuario de FTP pueden tener distintos permisos, por lo que los métodos de la clase FTPClient funcionarán en función de que dichos permisos estén correctamente configurados.

Consulta el Ejemplo4

6. Programación de envios y recepción de correos electrónicos

El envío y recepción de correos electrónicos desde las clases nativas de Java es posible pero sumamente árido y laborioso. Por suerte, existen alternativas que proporcionan mecanismos más sencillos y compactos.

El framework JavaMail dispone de las clases necesarias para poder programar las acciones más habituales que podría realizar un sistema informático en relación con los correos electrónicos, tales como el envío y la recepción de mensajes con y sin adjuntos.

Se puede utilizar `JavaMail` descargando y añadiendo al proyecto las librerías contenidas en los ficheros `javax.mail.jar` y `javax.activation-1.2.0.jar` (Ojo! no confundir con `javax.activation-api-1.2.0.jar`).

El proceso de envío de correos electrónicos compuestos únicamente por textos consta de los siguientes pasos:

- Creación de la sesión, indicando la URL del servidor de SMTP, el puerto, si utiliza SSL y si se requiere autenticación.
- Creación del mensaje (objeto Message). En este objeto se incluye la dirección de correo del emisor, la del destinatario, el asunto y el texto del mensaje.
- Establecimiento de la conexión (creación de objeto Transport), indicando el sistema de transporte.
- Envío del mensaje.
- Cierre de la conexión.

Si el correo lleva ficheros adjuntos, la creación del mensaje se divide en varias etapas. El proceso completo es el siguiente:

- Creación de la sesión, indicando la URL del servidor de SMTP, el puerto, si utiliza SSL y si se requiere autenticación.
- Creación del mensaje (objeto Message). En este objeto se incluyen la dirección de correo del emisor, la del destinatario y el asunto. Además, la creación del mensaje implica:
    - Creación de la instancia de `BodyPart` que contiene el texto del mensaje.
    - Creación de la instancia de `MimeBodyPart` que contiene el adjunto del mensaje.
    - Creación de la instancia de `Multipart`, que agrupa al objeto `BodyPart` y al objeto `MimeBodyPart`.
    - Inclusión del objeto `Multipart` en el mensaje.
- Establecimiento de la conexión (creación de objeto Transport), indicando el sistema de transporte.
- Envío del mensaje.
- Cierre de la conexión.

Consulta el Ejemplo5

Por su parte, la lectura de los correos electrónicos almacenados en un servidor IMAP consta de los siguientes pasos:

- Creación de la sesión (Session) IMAP, indicando el protocolo, el nombre del host, el puerto, si utiliza SSL y el servidor de confianza asociado.
- Configuración y obtención del almacén (Store).
- Obtención de la conexión a través del almacén, indicando identificador y contraseña de la cuenta.
- Obtención de la carpeta que se desea leer.
- Apertura de la carpeta.
- Obtención de los mensajes
- Procesado de los mensajes

Consulta el Ejemplo6

7. Programación distribuida

La programación distribuida es un paradigma de programación consistente en distribuir un sistema software entre un conjunto de ordenadores conectados en red. La ventaja de este sistema de computación se encuentra en que habitualmente la potencia de cálculo que se puede obtener mediante el uso conjunto de un grupo de ordenadores tiene un coste mucho más bajo que el que tendría un único ordenador con la misma potencia de
cálculo.

En un sistema distribuido, el sistema software se divide en componentes y estos son distribuidos y ejecutados en distintos ordenadores, unificándose los resultados obtenidos posteriormente. Es fácil intuir que no todos los problemas son susceptibles de ejecutarse en un entorno distribuido y obtener mejoras evidentes. No obstante, en muchos casos, la programación distribuida es la mejor alternativa para realizar procesos computacionales complejos.

Los principales elementos que forman parte de un sistema distribuido son:

- Nodos. Cada uno de los ordenadores que forman parte de la red.
- Componentes de software. Los elementos de software que implementan la funcionalidad del sistema, principalmente clases.
- Registro remoto de objetos. Elemento de la red que conoce la ubicación de los componentes en los nodos.
- Red y protocolos. Infraestructura física y lógica necesaria para comunicar los diferentes participantes de la red.
- A Interfaz remota. Es la interfaz compartida por el cliente y el servidor, aunque la implementación se encuentra en el servidor. El cliente, por su parte, obtendrá una instancia remota de dicha interfaz y sobre ella realizará las operaciones.
- Stubs. Es la proyección del objeto remoto en el cliente. El cliente invoca a los métodos remotos sobre el stub y este propaga las llamadas a la implementación remota equivalente o skeleton.
- Skeleton. Es el objeto remoto que recibe las llamadas desde el stub y hace que se ejecute la funcionalidad en el lado servidor.

En un sistema distribuido todos los nodos de la red pueden ser clientes y servidores. No existe un servidor único, ya que el sistema se reparte (se distribuye) entre los nodos. Cada elemento de la red puede tener y ejecutar una o más partes del sistema, actuando como un conjunto único. Los componentes se reparten entre los nodos que lo forman, obteniendo las siguientes ventajas frente a un sistema convencional:

- **Potente**. La suma de los procesadores de los nodos hace que la capacidad de cálculo de un sistema distribuido sea potencialmente ilimitada.
- **Escalable**. Dado que por definición el sistema se distribuye entre varios equipos, el número de estos puede crecer en número y en capacidad.
  Tolerante a fallos. Los componentes del sistema están distribuidos entre los distintos nodos, pero no existe una relación nodo-componente predeterminada. Si un nodo deja de funcionar, los componentes que proporcionaba pueden pasar a ser proporcionados por otros nodos.
- **Eficiente**. Varios ordenadores de capacidad y coste moderado pueden proporcionar la misma capacidad de cálculo que un ordenador de capacidad y coste alto con una menor inversión.

- **Transparente**. Dada la arquitectura de los sistemas distribuidos, los nodos no tienen información sobre los otros nodos. El propio sistema hace de intermediario entre ellos, por lo que los cambios en la red, ya sea añadiendo o eliminando nodos,
  no afectan a su sistema.

El funcionamiento concreto de un sistema distribuido que utiliza el paradigma de la orientación a objetos es el siguiente. Las instancias se distribuyen entre los nodos del sistema distribuido, ejecutándose cada uno en su propio procesador, haciendo en cierto modo el rol de servidores. La aplicación que necesita para su ejecución de dichas instancias accede a un servicio intermediario que proporciona las referencias remotas a los objetos.

Desde un punto de vista programático, la invocación a los métodos sobre los objetos remotos se efectúa de manera local, pero la ejecución en sí se realiza en el ordenador que tiene la instancia.

De manera más esquemática, los pasos para construir y utilizar un sistema distribuido son los siguientes:

- Elementos servidor:
  - Creación de los objetos e Instalación en los nodos de la red.
  - Registro de los objetos en el registro remoto de objetos.
- Elementos del cliente:
  - Acceso al registro de objetos para obtener las referencias remotas a los objetos.
  - Invocación a los métodos proporcionados por los objetos remotos.

Hay que percibir que las referencias remotas son proyecciones del objeto servidor en la máquina cliente en la que se están utilizando. Se puede resumir que el objeto es remoto, pero el uso es local.

Java dispone de su propia tecnología para el desarrollo de aplicaciones distribuidas, llamada Java Remote Method Invocation o RMI.
Esta tecnología proporciona las herramientas necesarias para crear los diversos componentes que participan en una aplicación desarrollada en esta arquitectura. De todos los componentes proporcionados los más relevantes son los siguientes:

- La interfaz `java.rmi.Remote` que permite definir los métodos que se pueden ejecutar en una máquina virtual Java no local. Los métodos definidos en esta interfaz deben lanzar la excepción `java.rmi.RemoteException`. Utilizada en cliente y servidor.
- La interfaz `java.rmi.registry.Registry` permite el registro de objetos remotos para almacenar y recuperar referencias de los mismos. Utilizada en cliente y servidor.
- La clase final `java.rmi.registry.LocateRegistry` usada para obtener la referencia al registro de objetos remotos (Registry). Utilizada en cliente y servidor.
- La clase `java.rmi.server.UnicastRemoteObject` utilizada para la exportación del objeto remoto y la generación del stub. Utilizada en cliente y servidor.
- Programa `rmiregistry`, responsable de arrancar el registro de objetos remotos en un host y puerto determinado. Se distribuye de manera conjunta con el JDK de Java.

Consulta el Ejemplo7

8. Ejemplos de la Unidad 04

8.1. Ejemplo1

Petición HTTP con `HttpUrlConnection` (Java 8)

La web del diccionario de la RAE (Real Academia Española) permite consultar el significado de las palabras de la lengua española. Dispone de un sencillo formulario que facilita la introducción de una palabra para realizar la búsqueda. Esta palabra se concatena a la URL para realizar la petición del recurso correspondiente.

Por ejemplo, si se introduce la palabra software en el formulario de búsqueda, se solicita el recurso https://dle.rae.es/software

Por lo tanto, la petición es de tipo GET y no dispone de parámetros.

En este ejemplo vamos a crear un programa Java que haga peticiones a la web de la RAE y que almacene el código HTML obtenido en un fichero.

Hay que destacar la necesidad de codificar la palabra buscada mediante el método `encode` de la clase `URLEncoder` para que caracteres como la letra ñ, los espacios en blanco o las vocales con tilde tengan el formato aceptado por el protocolo.

```
1  package UD04.Ejemplo01;
2
3  import java.io.IOException;
4  import java.io.InputStreamReader;
5  import java.io.Reader;
6  import java.net.HttpURLConnection;
7  import java.net.URL;
8  import java.net.URLEncoder;
9  import java.nio.charset.StandardCharsets;
10 import java.nio.file.Files;
11 import java.nio.file.Path;
12 import java.nio.file.Paths;
13
14 public class RAE {
15
16     public static StringBuilder htmlDownload(String address) throws Exception {
17         StringBuilder answer = new StringBuilder();
18         URL url = new URL(address);
19         HttpURLConnection connection = (HttpURLConnection) url.openConnection();
20         connection.setRequestMethod("GET");
21         connection.setRequestProperty("Content-Type", "text/plain");
22
23         connection.setRequestProperty("charset", "utf-8");
24         connection.setRequestProperty("User-Agent", "Mozilla/5.0");
25         int state = connection.getResponseCode();
26         Reader streamReader = null;
27         if (state == HttpURLConnection.HTTP_OK) {
```

```
            streamReader = new InputStreamReader(connection.getInputStream());
            int character;
            while ((character = streamReader.read()) != -1) {
                answer.append((char) character);
            }
        } else {
            throw new Exception("HTTP Error: " + state);
        }
        connection.disconnect();
        return answer;
    }

    public static void writeFile(String strPath, String content) throws IOException {
        Path path = Paths.get(strPath);
        byte[] strToBytes = content.getBytes();
        Files.write(path, strToBytes);
    }

    public static void main(String[] args) {
        try {
            String scheme = "https://";
            String server = "dle.rae.es/";
            String resource = URLEncoder.encode("Tiburón",
                    StandardCharsets.UTF_8.name());
            String address = scheme + server + resource;
            StringBuilder result = htmlDownload(address);
            RAE.writeFile("src/UD04/Ejemplo01/tiburon.html",
                    result.toString());
            System.out.println("Download completed");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

8.2. Ejemplo2

Petición HTTP con parámetros con `HttpUrlConnection` (Java 8)

IMDb (Internet Movie Database o Base de Datos de Películas en Internet) es una web que contiene infinidad de información relacionada con el cine y la televisión, incluyendo películas, series, actores, directores, etc.

Esta web proporciona un formulario de búsqueda en el que se puede introducir cualquier palabra o palabras para buscar películas, actores o empresas que contengan dichas palabras.

Al introducir las palabras, el formulario genera una petición HTTP de tipo GET que incluye un parámetro con nombre q y con valor las palabras introducidas. Por ejemplo, si se introducen las palabras star wars la petición creada es https://www.imdb.com/find?q=star+wars.

En este ejemplo se creará un programa en Java que dada una palabra o palabras realice una petición HTTP a la web IMDb y almacene el resultado en un fichero HTML.

Es exactamente igual que el Ejemplo1 salvo en la composición de la petición HTTP.

```java
package UD04.Ejemplo02;

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class IMDb {

    public static StringBuilder htmlDownload(String address) throws Exception {
        StringBuilder answer = new StringBuilder();
        URL url = new URL(address);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Content-Type", "text/plain");

        connection.setRequestProperty("charset", "utf-8");
        connection.setRequestProperty("User-Agent", "Mozilla/5.0");
        int state = connection.getResponseCode();
        Reader streamReader = null;
        if (state == HttpURLConnection.HTTP_OK) {
            streamReader = new InputStreamReader(connection.getInputStream());
            int character;
            while ((character = streamReader.read()) != -1) {
                answer.append((char) character);
            }
        } else {
            throw new Exception("HTTP Error: " + state);
        }
        connection.disconnect();
        return answer;
    }

    public static void writeFile(String strPath, String content) throws IOException {
        Path path = Paths.get(strPath);
        byte[] strToBytes = content.getBytes();
        Files.write(path, strToBytes);
    }

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

```
        try {
            String scheme = "https://";
            String server = "www.imdb.com";
            String path = "/find";
            String text = URLEncoder.encode("Tiburón",
                    StandardCharsets.UTF_8.name());
            String parameters = "?q=";
            String address = scheme + server + path + parameters + text;
            StringBuilder result = htmlDownload(address);
            IMDb.writeFile("src/UD04/Ejemplo02/tiburon.html", result.toString());
            System.out.println("Download completed");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

8.3. Ejemplo3

Petición HTTP con `java.net.http` (Java 11 y superior)

Este ejemplo realiza las mismas acciones que el Ejemplo1 pero con las clases de la libreria `java.net.http`.

```
package UD04.Ejemplo03;

import java.io.UnsupportedEncodingException;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.HttpURLConnection;
import java.net.URI;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.nio.file.Path;

public class RAE2 {

    public static int htmlDownload(String address) throws Exception {
        URI url = new URI(address);
        HttpClient httpClient = HttpClient.newBuilder()
                .version(HttpClient.Version.HTTP_1_1)
                .followRedirects(HttpClient.Redirect.NORMAL)
                .build();
        HttpRequest request = HttpRequest.newBuilder()
                .GET()
                .uri(url)
                .headers("Content-Type", "text/plain")
                .setHeader("User-Agent", "Mozilla/5.0")
                .build();
```

```
        HttpResponse<Path> response =
httpClient.send(request,HttpResponse.BodyHandlers.ofFile(Path.of("src/UD04/Ejemplo03/tiburon.
html")));
        return response.statusCode();
    }

    public static void main(String[] args) {
        try {
            String scheme = "https://";
            String server = "dle.rae.es/";
            String resource = URLEncoder.encode("Tiburón",
                    StandardCharsets.UTF_8.name());
            String address = scheme + server + resource;
            int stateCode = htmlDownload(address);
            if (stateCode == HttpURLConnection.HTTP_OK) {
                System.out.println("Download completed");
            } else {
                System.err.println("Error: " + stateCode);
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

8.4. Ejemplo4

Subida y descarga de un fichero a través de FTP

Este ejemplo sube un fichero al servidor FTP.

> Este ejemplo requiere el uso del paquete `org.apache.commons` que hemos visto en teoria.

```
package UD04.Ejemplo04;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.SocketException;
import org.apache.commons.net.ftp.*;

/**
 *
 * @author David Martínez (wwww.martinezpenya.es|www.eduardoprimo.es)
 */
public class FTPManager {

    private FTPClient FTPClient;
```

```
    private static final String SERVER = "ftp.dlptest.com";
    private static final int PORT = 21;
    private static final String USER = "dlpuser";
    private static final String PASSWORD = "rNrKYTX9g7z3RgJRmxWuGHbeu";

    public FTPManager() {
        FTPClient = new FTPClient();
    }

    private void connect() throws SocketException, IOException {
        FTPClient.connect(SERVER, PORT);
        int answer = FTPClient.getReplyCode();
        if (!FTPReply.isPositiveCompletion(answer)) {
            FTPClient.disconnect();
            throw new IOException("Error connecting to FTP Server");
        }
        boolean credentials = FTPClient.login(USER, PASSWORD);
        if (!credentials) {
            throw new IOException("Error connecting to FTP. Wrong credentials");
        }
        FTPClient.setFileType(FTP.BINARY_FILE_TYPE);
    }

    private void desconectar() throws IOException {
        FTPClient.disconnect();
    }

    private boolean uploadFile(String path) throws IOException {
        File localFile = new File(path);
        boolean sent;
        try ( InputStream is = new FileInputStream(localFile)) {
            sent = FTPClient.storeFile(localFile.getName(), is);
        }
        return sent;
    }

    public static void main(String[] args) {
        FTPManager ftpManager = new FTPManager();
        try {
            ftpManager.connect();
            System.out.println("Connected");
            boolean uploaded = ftpManager.uploadFile(
                    "src/UD04/Ejemplo04/upload.txt");
            if (uploaded) {
                System.out.println("File successfully uploaded.");
            } else {
                System.err.println("Something went wrong uploading file.");
            }
            ftpManager.desconectar();
            System.out.println("Disconnected");

```

```
68          } catch (Exception e) {
69              System.err.println("Error: " + e.getMessage());
70          }
71      }
72  }
```

8.5. Ejemplo5

Envío de correos electrónicos sin adjuntos desde una cuenta GMail

Este ejemplo tiene como objetivos el envío de un correo electrónico que incluya solo texto utilizando una cuenta de Gmail.

Durante la ejecución, el programa solicitará al usuario la dirección de correo electrónico del destinatario y del emisor y la contraseña. Dicha contraseña no se muestra cifrada en pantalla, por lo que hay que asegurarse de que no hay nadie observando.

El texto del mensaje está codificado en el código.

Configurar la cuenta gmail del instituto o la personal para poder utilizar aplicaciones externas (no seguras)

Entrar a la configuración de tu cuenta. Haz clic en el icono con tu inicia y elige el botón [Gestionar tu cuenta de Google], accede al apartado [Seguridad] (en la parte izquierda de la pantalla). A continuación busca la opción "Acceso a aplicaciones menos seguras" y asegurate de que esté activada.

Una vez enviados el correo, debe aparecer en el buzón de la cuenta del destinatatio.

Este ejemplo requiere del paquete `javax.mail` (JavaMail) y `javax.activation-1.2.0` tal y como se ha comentado en la teoria.

```
1   package UD04.Ejemplo05;
2
3   import java.io.IOException;
4   import java.util.Properties;
5   import java.util.Scanner;
6   import javax.mail.Message;
7   import javax.mail.MessagingException;
8   import javax.mail.NoSuchProviderException;
9   import javax.mail.Session;
10  import javax.mail.Transport;
11  import javax.mail.internet.AddressException;
12  import javax.mail.internet.InternetAddress;
13  import javax.mail.internet.MimeMessage;
14
15  /**
16   *
17   * @author David Martínez (wwww.martinezpenya.es|www.eduardoprimo.es)
18   */
19  public class EmailManager {
20
21      private Properties properties;
```

```
22      private Session session;
23
24      private void setSMTPServerProperties() {
25          properties = System.getProperties();
26          properties.put("mail.smtp.auth", "true");
27          properties.put("mail.smtp.host", "smtp.gmail.com");
28          properties.put("mail.smtp.port", 587);
29          properties.put("mail.smtp.starttls.enable", "true");
30          session = Session.getInstance(properties, null);
31      }
32
33      private Transport connectSMTPServer(String emailAddress, String password) throws
    NoSuchProviderException, MessagingException {
34          Transport t = (Transport) session.getTransport("smtp");
35          t.connect(properties.getProperty("mail.smtp.host"), emailAddress,
36                  password);
37          return t;
38      }
39
40      private Message createMessageCore(String from, String to, String subject) throws
    AddressException, MessagingException {
41          Message message = new MimeMessage(session);
42          message.setFrom(new InternetAddress(from));
43          message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
44          message.setSubject(subject);
45          return message;
46      }
47
48      private Message createTextMessage(String from, String to, String subject,
49              String messageText) throws MessagingException, AddressException, IOException {
50          Message message = createMessageCore(from, to, subject);
51          message.setText(messageText);
52          return message;
53      }
54
55      public void sendTextMessage(String from, String to, String subject,
56              String messageText, String user, String password) throws AddressException,
    MessagingException, IOException {
57          setSMTPServerProperties();
58          Message message = createTextMessage(from, to, subject, messageText);
59          Transport t = connectSMTPServer(user, password);
60          t.sendMessage(message, message.getAllRecipients());
61          t.close();
62      }
63
64      public static void main(String[] args) {
65          try {
66              Scanner sc = new Scanner(System.in);
67              System.out.print("Introduce email to:");
68              String emailTo = sc.nextLine();
69              System.out.print("Introduce email from:");
```

```
            String emailFrom = sc.nextLine();
            System.out.print("Introduce password:");
            String passwordFrom = sc.nextLine();
            sc.close();
            EmailManager emailManager = new EmailManager();
            emailManager.sendTextMessage(emailFrom, emailTo,
                    "Sample text email without attachment",
                    "Test Message from Java.",
                    emailFrom, passwordFrom);
            System.out.println("Email sent.");
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

8.6. Ejemplo6

Lectura de los correos de la carpeta INBOX de una cuenta de correo de GMail

Este ejemplo lee la carpeta de entrada de una cuenta de correo electrónico de Gmail. De cada mensaje disponible se mostrarán la dirección de correo del emisor, la dirección de correo del receptor, el asunto y el texto del mensaje en el caso de que el contenido sea únicamente texto. En caso que el correo lleve ficheros adjuntos se mostrará un aviso en lugar del texto.

Activar IMAP en la cuenta gmail del instituto o la personal para poder leer correos

La ventaja de IMAP frente a POP3 es que la lectura de los mensajes no supone su eliminación del servidor, por lo que seguirán siendo accesibles posteriormente desde cualquier dispositivo. Debido a que las cuentas de Gmail están diseñadas para ser accedidas desde la página web, no están configuradas para ser utilizadas desde aplicaciones externas, teniendo deshabilitados los accesos a través de IMAP y de POP3.

Accede a la configuración de GMail (icono de una rueda dentada en la parte superior derecha de la página) y pulsar sobre el botón "Ver todos los ajustes". Dentro de la pestaña "Reenvio y correo POP/IMAP" asegurate que la casilla "Habilitar IMAP" esté seleccionada.

Este ejemplo requiere del paquete `javax.mail` (JavaMail) y `javax.activation-1.2.0` tal y como se ha comentado en la teoria.

```
package UD04.Ejemplo06;

import com.sun.mail.imap.IMAPFolder;
import java.util.Properties;
import java.util.Scanner;
import javax.mail.*;
import javax.mail.Message.*;
import javax.mail.internet.*;

public class EmailReader {
```

```
11
12      private Session getSesionImap() {
13          Properties properties = new Properties();
14          properties.setProperty("mail.store.protocol", "imap");
15          properties.setProperty("mail.imap.host", "imap.gmail.com");
16          properties.setProperty("mail.imap.port", "993");
17          properties.setProperty("mail.imap.ssl.enable", "true");
18          properties.setProperty("mail.imap.ssl.trust", "imap.gmail.com");
19          Session session = Session.getDefaultInstance(properties);
20          return session;
21      }
22
23      public void readINBOX(String email, String password) throws Exception {
24          Session session = this.getSesionImap();
25          Store storage = session.getStore("imaps");
26          storage.connect("imap.gmail.com", 993, email, password);
27          IMAPFolder inbox = (IMAPFolder) storage.getFolder("INBOX");
28          inbox.open(Folder.READ_WRITE);
29          Message[] messages = inbox.getMessages();
30          for (Message message : messages) {
31              Address[] fromAddress = message.getFrom();
32              String from = fromAddress[0].toString();
33              Address[] toAddress = message.getRecipients(RecipientType.TO);
34              String to = toAddress[0].toString();
35              String subject = message.getSubject();
36              MimeMultipart mimeMultipart = (MimeMultipart) message.getContent();
37              if (mimeMultipart.getBodyPart(0).isMimeType("text/plain")) {
38                  String textoMensaje = (String) mimeMultipart.getBodyPart(0).getContent();
39                  System.out.printf("From %s To %s Subject: %s Message: %s\n", from,
40                          to, subject, textoMensaje);
41              } else {
42                  System.out.printf("From %s To %s Subject: %s Message: %s\n", from,
43                          to, subject, "[*ATTACHED FILES*]");
44              }
45          }
46      }
47
48      public static void main(String[] args) {
49          Scanner sc = new Scanner(System.in);
50          System.out.print("Introduce your gmail address:");
51          String email = sc.nextLine();
52          System.out.print("Introduce your password:");
53          String password = sc.nextLine();
54          sc.close();
55          try {
56              new EmailReader().readINBOX(email, password);
57          } catch (Exception e) {
58              e.printStackTrace();
59          }
60      }
61  }
```

8.7. Ejemplo7

A continuación, se presenta un ejemplo que ilustra el proceso de creación y ejecución de una aplicación distribuida utilizando Java RMI.

El sistema consta de dos aplicaciones, cliente y servidor. Ambas comparten la misma interfaz remota, en este caso `Calculator`.

En el lado servidor, se encuentran la clase `Calculator`, donde se localizan las implementaciones de los métodos remotos, y la clase `Register`, responsable de crear el registro (mediante el método `createRegistry` de la clase `LocateRegistry`, indicando host y puerto) y de instanciar y registrar (método `bind` de `Registry`) los objetos remotos, en este caso la clase `Calculator`.

En el lado cliente, además de la interfaz, se encuentra la clase `Client`. En esta clase se obtiene la referencia al registro de objetos remotos (debe conocer la máquina y el puerto en el que se encuentra) para, a partir de esta, obtener la referencia al objeto remoto mediante el método lookup. Una vez se ha obtenido la referencia, se invocan a los métodos de la interfaz remota como si se tratase de un objeto local.

Interfaz:

```
1  package UD04.Ejemplo07.interfaces;
2
3  import java.rmi.Remote;
4  import java.rmi.RemoteException;
5
6  public interface CalculatorInterface extends Remote {
7      public int add(int o1, int o2) throws RemoteException;
8      public int substract(int o1, int o2) throws RemoteException;
9      public int multiply(int o1, int o2) throws RemoteException;
10     public int divide(int o1, int o2) throws RemoteException;
11 }
```

Registro:

```
1  package UD04.Ejemplo07.server;
2
3  import UD04.Ejemplo07.interfaces.CalculatorInterface;
4  import java.rmi.AlreadyBoundException;
5  import java.rmi.RemoteException;
6  import java.rmi.registry.LocateRegistry;
7  import java.rmi.registry.Registry;
8  import java.rmi.server.UnicastRemoteObject;
9
10 public class Register {
11
12     public static void registerCalculator() {
13         try {
14             Registry register = LocateRegistry.createRegistry(5555);
15             Server calculator = new Server();
16             register.bind("Server",
```

```
                    (CalculatorInterface) UnicastRemoteObject.exportObject(
                            calculator, 0));
            System.out.println("Server ready...");
        } catch (RemoteException ex) {
            ex.printStackTrace();
        } catch (AlreadyBoundException ex) {
            ex.printStackTrace();
        }

    }

    public static void main(String[] args) {
        registerCalculator();
    }
}
```

Servidor:

```
package UD04.Ejemplo07.server;

import UD04.Ejemplo07.interfaces.CalculatorInterface;
import java.rmi.RemoteException;

public class Server implements CalculatorInterface {

    @Override
    public int add(int o1, int o2) throws RemoteException {
        return o1 + o2;
    }

    @Override
    public int substract(int o1, int o2) throws RemoteException {
        return o1 - o2;
    }

    @Override
    public int multiply(int o1, int o2) throws RemoteException {
        return o1 * o2;
    }

    @Override
    public int divide(int o1, int o2) throws RemoteException {
        return o1 / o2;
    }
}
```

Cliente:

```
package UD04.Ejemplo07.client;

```

```java
import UD04.Ejemplo07.interfaces.CalculatorInterface;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {

    private CalculatorInterface calculator = null;

    public Client() {
        try {
            Registry registro = LocateRegistry.getRegistry("localhost", 5555);
            calculator = (CalculatorInterface) registro.lookup("Server");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Client client = new Client();
        int result;
        try {
            result = client.calculator.add(34, 5);
            System.out.println(result);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

La ejecución consta de tres fases:

- El Arranque del servidor de registro mediante la ejecución del programa `rmiregistry`.

```
$ rmiregistry
```

- Ejecución del servidor, que creará y registrará los objetos remotos.

```
Server ready...
```

- Ejecución de los clientes que obtienen las referencias a los objetos remotos y las utilizan invocando a sus métodos.

```
39
```

9. Fuentes de información

- Wikipedia
- Programación de servicios y procesos - FERNANDO PANIAGUA MARTÍN [Paraninfo]
- Programación de Servicios y Procesos - ALBERTO SÁNCHEZ CAMPOS [Ra-ma]
- Programación de Servicios y Procesos - Mª JESÚS RAMOS MARTÍN - [Garceta] (1ª y 2ª Edición)
- Programación de servicios y procesos - CARLOS ALBERTO CORTIJO BON [Sintesis]
- Programació de serveis i processos - JOAR ARNEDO MORENO, JOSEP CAÑELLAS BORNAS i JOSÉ ANTONIO LEO MEGÍAS [IOC]
- GitHub repositories:
    - https://github.com/ajcpro/psp
    - https://oscarmaestre.github.io/servicios/index.html
    - https://github.com/juanro49/DAM/tree/master/DAM2/PSP
    - https://github.com/pablohs1986/dam_psp2021
    - https://github.com/Perju/DAM
    - https://github.com/eldiegoch/DAM
    - https://github.com/eldiegoch/2dam-psp-public
    - https://github.com/franlu/DAM-PSP
    - https://github.com/ProgProcesosYServicios
    - https://github.com/joseluisgs
    - https://github.com/oscarnovillo/dam2_2122
    - https://github.com/PacoPortillo/DAM_PSP_Tarea02_La-Cena-de-los-Filosofos