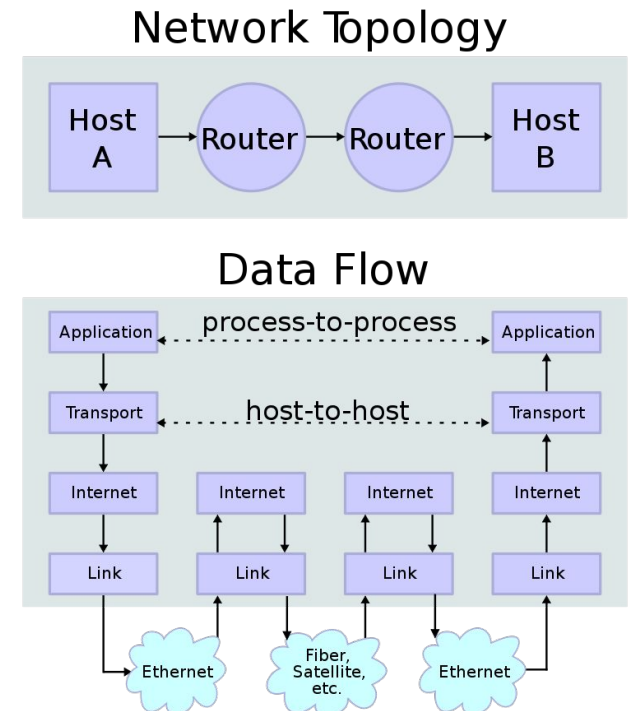


# Programación de servicios en red

Programación de servicios y procesos



# Objetivos:

---

Los servicios son programas auxiliares utilizados en un sistema para gestionar una colección de recursos y prestar su funcionalidad a los usuario y aplicaciones.

El acceso a los servicios está formado por el conjunto de operaciones que ofrece, por ejemplo, un servicio de archivos ofrece operaciones de lectura, escritura y borrado de ficheros.

Todos los servicios de Internet implementan una relación cliente-servidor. Existen multitud de librerías para trabajar con los servicios más comunes.

Objetivos de esta unidad:

- Conocer algunos de los protocolos estándar de comunicación en red a nivel de aplicación (telnet, ftp, http, pop3, smtp, entre otros).
- Conocer y utilizar librerías Java para usar algunos de los protocolos de aplicación más importantes de TCP/IP.
- Programar una aplicación cliente

# Contenidos:

---

- 1) Conceptos básicos
- 2) Protocolos de comunicación.
- 3) El protocolo HTTP y HTTPS
- 4) El protocolo FTP
- 5) El protocolo SMTP y POP3
- 6) El protocolo DNS
- 7) Bibliotecas de clases y componentes Java
- 8) Ejemplo: Petición con HttpURLConnection

# Conceptos básicos.

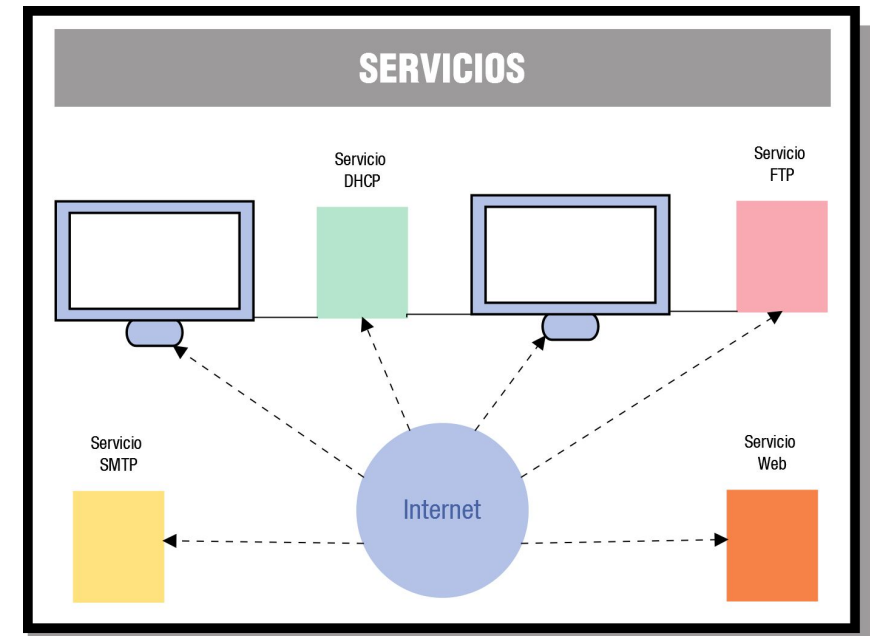
Una red de ordenadores o red informática la podemos definir como un sistema de comunicaciones que conecta ordenadores y otros equipos informáticos entre sí, con la finalidad de compartir información y recursos.

Mediante la compartición de información y recursos en una red, los usuarios de la misma pueden hacer un mejor uso de ellos y, por tanto, mejorar el rendimiento global del sistema u organización.

Las principales ventajas de utilizar una red de ordenadores las podemos resumir en las siguientes:


- Reducción en el presupuesto para software y hardware.
- Posibilidad de organizar grupos de trabajo.
- Mejoras en la administración de los equipos y las aplicaciones.
- Mejoras en la integridad de los datos.
- Mayor seguridad para acceder a la información.
- Mayor facilidad de comunicación.

Pues bien, **los servicios en red** son responsables directos de estas ventajas.



# Conceptos básicos.

---

Un servicio en red es un software o programa que proporciona una determinada funcionalidad o utilidad al sistema. Por lo general, estos programas están basados en un conjunto de  protocolos y estándares

Una clasificación de los servicios en red atendiendo a su finalidad o propósito puede ser la siguiente:

- Administración/Configuración. Esta clase de servicios facilita la administración y gestión de las configuraciones de los distintos equipos de la red, por ejemplo: los servicios **DHCP** y **DNS**.
- Acceso y control remoto. Los servicios de acceso y control remoto, se encargan de permitir la conexión de usuarios a la red desde lugares remotos, verificando su identidad y controlando su acceso, por ejemplo **Telnet** y **SSH**.
- De Ficheros. Los servicios de ficheros consisten en ofrecer a la red grandes capacidades de almacenamiento para descongestionar o eliminar los discos de las estaciones de trabajo, permitiendo tanto el almacenamiento como la transferencia de ficheros, por ejemplo **FTP**.
- Información. Los servicios de información pueden servir ficheros en función de sus contenidos, como pueden ser los documentos de hipertexto, por ejemplo **HTTP**, o bien, pueden servir información para ser procesada por las aplicaciones, como es el caso de los servidores de bases de datos.
- Comunicación. Permiten la comunicación entre los usuarios a través de mensajes escritos, por ejemplo email o correo electrónico mediante el protocolo **SMTP**.

# 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 la 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 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 S.O.

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. La capa de aplicación del modelo TCP/IP contiene las aplicaciones y los servicios que puede usar un usuario. Es la capa más próxima a las personas y sobre la que se desarrollan las aplicaciones que usan a diario.

Dentro de la capa de aplicación se ubican varios protocolos, cada uno de los cuales con su cometido específico. El número de protocolos es extenso por lo que en esta unidad veremos sólo algunos de los más relevantes.

# El protocolo HTTP y HTTPS

---

El protocolo HTTP o Protocolo de Transferencia de Hipertexto es un conjunto de normas que posibilitan la comunicación entre servidor y cliente, permitiendo la transmisión de información entre ambos. La información transferida son las conocidas páginas HTML o páginas web. Se trata del método más común de intercambio de información en la World Wide Web.

HTTPS es la versión segura de HTTP ya que cifra toda la información intercambiada entre cliente y servidor.

HTTP define tanto la sintaxis como la semántica que utilizan clientes y servidores para comunicarse. Algunas consideraciones importantes sobre HTTP son las siguientes:

- ◆ Es un protocolo que sigue el esquema petición-respuesta entre un cliente y un servidor.
- ◆ Utiliza por defecto el puerto 80 (https utiliza por defecto el puerto 443)
- ◆ Al cliente que efectúa la petición, por ejemplo un navegador web, se le conoce como agente del usuario (user agent).
- ◆ A la información transmitida se la llama recurso y se la identifica mediante un localizador uniforme de recursos (URL).
- ◆ Por ejemplo: <http://www.ieclaradelrey.es>
- ◆ Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

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

# El protocolo HTTP y HTTPS

---

Uno de los datos técnicos que resulta necesario conocer de este protocolo es el referente al tipo de peticiones que se pueden realizar. Se conoce como método y son los que se muestran en la siguiente tabla:

 <b>GET:</b> El método GET solicita una un recurso específico.  Las peticiones que usan el método GET solo deben recuperar datos.	 <b>HEAD:</b> Este método pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta, únicamente con el encabezado de la solicitud.	 <b>POST:</b> Se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
 <b>PUT:</b> Es similar al POST, solo que el método PUT se utiliza para la actualización de información existente, es semejante a un UPDATE a nivel de base de datos.	 <b>DELETE:</b> Permite eliminar un recurso específico, generalmente se utiliza para eliminar información existente, es semejante a un DELETE a nivel de base de datos.	 <b>PATCH</b> Este método se emplea generalmente para realizar modificaciones parciales de un recurso en particular.



# El protocolo HTTP y HTTPS

---

Otro de los elementos que resulta fundamental conocer son los códigos de respuesta. Cuando se realiza una petición HTTP a un servidor este genera una respuesta que va identificada con un código y que proporciona información sobre el resultado de la petición:

- **1xx:** Mensaje informativo.
- **2xx:** Éxito
  - 200 OK
  - 201 Created
  - 202 Accepted
  - 204 No Content
- **3xx:** Redirección
  - 300 Multiple Choice
  - 301 Moved Permanently
  - 302 Found
  - 304 Not Modified
- **4xx:** Error del cliente
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- **5xx:** Error del servidor
  - 500 Internal Server Error
  - 501 Not Implemented
  - 502 Bad Gateway
  - 503 Service Unavailable

# El protocolo FTP

---

El protocolo FTP o Protocolo de Transferencia de Archivos proporciona un mecanismo estándar de transferencia de archivos entre sistemas a través de redes TCP/IP.

Las principales prestaciones de un servicio basado en el protocolo FTP o servicio FTP son las siguientes:

- ✓ Permite el intercambio de archivos entre máquinas remotas a través de la red.
- ✓ Consigue una conexión y una transferencia de datos muy rápidas.

Sin embargo, el servicio FTP adolece de una importante deficiencia en cuanto a seguridad: La información y contraseñas se transmiten en texto plano.

Esto está diseñado así, para conseguir una mayor velocidad de transferencia. Al realizar la transmisión en texto plano, un posible atacante puede capturar este tráfico, acceder al servidor y/o apropiarse de los archivos transferidos.

Este problema de seguridad, se puede solucionar mediante la encriptación de todo el tráfico de información, a través del protocolo no estándar SFTP usando SSH o del protocolo FTPS usando SSL. De encriptación ya hablaremos más adelante, en otra unidad de este módulo.

# Transferencia de ficheros mediante FTP



En Java, de manera nativa, es posible realizar transferencias de ficheros mediante este protocolo, pero es sumamente árido. Apache Commons Net proporciona clases y utilidades para realizar cualquier operación sobre un servidor FTP o FTPS desde un cliente Java:

Clase	Descripción
FTP	Proporciona la funcionalidad básica para implementar un cliente FTP. Incluye constantes para indicar el tipo de ficheros a transmitir y configuración de estos. Esta clase hereda de SocketClient.
FTPClient	Subclase de FTP, encapsula la funcionalidad necesaria para subir y bajar ficheros a través del protocolo FTP.
FTPSCient	Subclase de FTPClient, permite utilizar el protocolo FTP sobre SSL.
FTPFile	Representa la información sobre los ficheros almacenados en el servidor.
FTPReply	Permite almacenar los valores devueltos por el servidor como código de respuesta a las peticiones del cliente.

# FTPCLIENT

- **CONECTAR**

- connect(String host)
- getReplyString()
- getReplyCode() (ver [RFC 959](#))
  - FTPReply.isPositiveCompletion(code)
- disconnect()

- **AUTENTICACIÓN**

- login(String user, String pwd)
- logout()

- **MODO**

- enterLocalPassiveMode()
- enterLocalActiveMode()

- **NAVEGACIÓN**

- printWorkingDirectory()
- listFiles(), listFiles(String path), listNames()
- listDirectories(), listDirectories(String path)
- changeWorkingDirectory(String path), changeToParentDirectory()

# FTPCLIENT

- **FICHEROS**

- **setFileType**(int fileType) (ASCII\_FILE\_TYPE o BINARY\_FILE\_TYPE)
- boolean **storeFile**(String nombre, InputStream local)
- boolean **retrieveFile**(String name, OutputStream local)
- boolean **deleteFile**(String pathName)
- rename(old, new)

- **DIRECTORIOS**

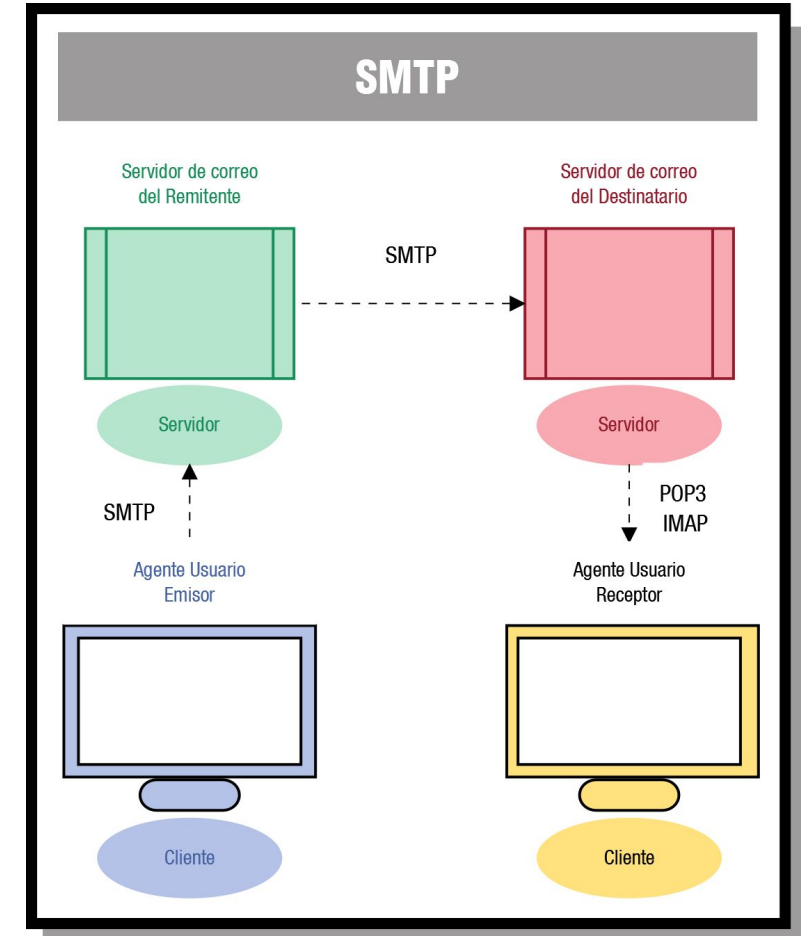
- removeDirectory(String path),  
makeDirectory(String path)

Oh, vaya,  
Streams 🤔

# El protocolo SMTP y POP3

El correo electrónico es un servicio que permite a los usuarios enviar y recibir mensajes y archivos rápidamente a través de la red.

- Principalmente se usa este nombre para denominar al sistema que provee este servicio en Internet, mediante el protocolo SMTP o Protocolo Simple de Transferencia de Correo, aunque por extensión también puede verse aplicado a sistemas análogos que usen otras tecnologías.
- Por medio de mensajes de correo electrónico se puede enviar, no solamente texto, sino todo tipo de documentos digitales.



# El protocolo SMTP y POP3

---

El servicio de correo basado en el protocolo SMTP sigue el modelo cliente/servidor, por lo que el trabajo se distribuye entre el programa Servidor y el Cliente. Algunas consideraciones importantes sobre el servicio de correo a través de SMTP:

- El servidor mantiene las cuentas de los usuarios así como los buzones correspondientes.
- Los clientes de correo gestionan la descarga de mensajes así como su elaboración.
- El protocolo SMTP se encarga del transporte del correo saliente desde la máquina del usuario remitente hasta el servidor que almacene los mensajes de los destinatarios.
  - ◆ El usuario remitente redacta su mensaje y lo envía hacia su servidor de correo.
  - ◆ Desde allí se reenvía al servidor del destinatario, quién lo descarga del buzón en la máquina local mediante el protocolo POP3, o la consulta vía web, haciendo uso del protocolo IMAP.

# El protocolo DNS

---

Todas las computadoras y dispositivos conectados a una red TCP/IP se identifican mediante una dirección IP, como por ejemplo 117.142.234.125.

En su forma actual (IPv4), una dirección IP se compone de cuatro bytes sin signo (de 0 a 254) separados por puntos para que resulte más legible, tal y como has visto en el ejemplo anterior. Por supuesto, se trata de valores ideales para los ordenadores, pero para los seres humanos que quieran acordarse de la IP de un nodo en concreto de la red, son todo un problema de memoria.

El sistema DNS o Sistema de Nombres de Dominio es el mecanismo que se inventó, para que los nodos que ofrecen algún tipo de servicio interesante tengan un nombre fácil de recordar, lo que se denomina un nombre de dominio, como por ejemplo **www.todofp.es**.

DNS es un sistema de nomenclatura jerárquica para computadoras, servicios o cualquier recurso conectado a Internet o a una red privada.



# El protocolo DNS

---

El objetivo principal de los nombres de dominio y del servicio DNS, es traducir o resolver nombres (por ejemplo `www.dominio.es`) en las direcciones IP (identificador binario) de cada equipo conectado a la red, con el propósito de poder localizar y direccionar estos equipos dentro de la red.

Sin la ayuda del servicio DNS, los usuarios de una red TCP/IP tendrían que acceder a cada servicio de la misma utilizando la dirección IP del nodo.

Además el servicio DNS proporciona otras ventajas:

- Permite que una misma dirección IP sea compartida por varios dominios.
- Permite que un mismo dominio se corresponda con diferentes direcciones IP.
- Permite que cualquier servicio de red pueda cambiar de nodo, y por tanto de IP, sin cambiar de nombre de dominio.

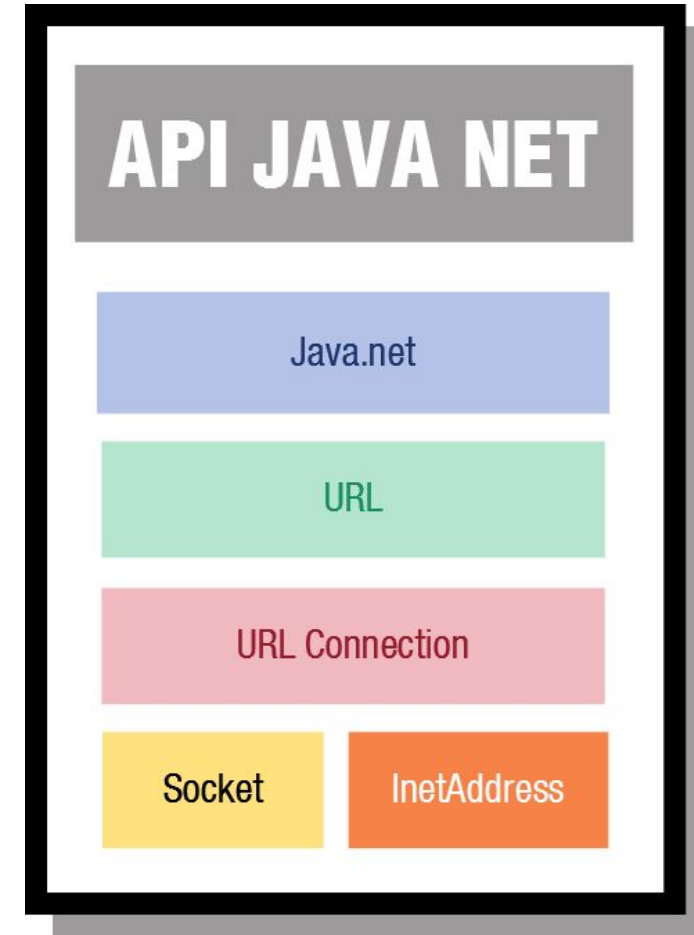
# Bibliotecas de clases y componentes Java

Prácticamente todos los lenguajes de programación modernos disponen de librerías para la programación de aplicaciones que hacen uso de la red. Sin estas librerías, la programación sería de un nivel de abstracción bajo y sería necesario un conocimiento exhaustivo de los protocolos.

Java se ha construido con extensas capacidades de interconexión TCP/IP y soporta diferentes niveles de conectividad en red, facilitando la creación de aplicaciones cliente/servidor y generación de servicios en red. El paquete principal que proporciona el API de Java para programar aplicaciones con comunicaciones en red es:

- **java.net:** Este paquete soporta clases para generar diferentes servicios de red, servidores y clientes.

Para ciertos servicios estándar, Java no proporciona objetos predefinidos y por tanto una solución fácil para generarlos es recurrir a bibliotecas externas, como por ejemplo, el API proporcionado por Apache Commons Net (las bibliotecas **org.apache.commons.net**). Esta API permite implementar aplicaciones cliente para los protocolos estándar más populares, como: Telnet, FTP, o FTP sobre HTTP entre otros.



# Objetos predefinidos

---

El paquete java.net, proporciona una API que se puede dividir en dos niveles:

- Una API de bajo nivel, que permite representar los siguientes objetos:
  - **Direcciones.** Son los identificadores de red, esto es, las direcciones IP. Clase InetAddress. Implementa una dirección IP.
  - **Sockets.** Son los mecanismos básicos de comunicación bidireccional de datos (Clase Socket, Clase ServerSocket, Clase DatagramSocket y Clase MulticastSocket).
- Una API de alto nivel, que se ocupa de representar los siguientes objetos, mediante las clases que te indicamos:
  - **URI.** Representan los identificadores de recursos universales. Clase URI.
  - **URL.** Representan localizadores de recursos universales. Clase URL. Representa una dirección URL.
  - **Conexiones.** Representa las conexiones con el recurso apuntado por URL.
    - Clase URLConnection. Es la superclase de todas las clases que representan un enlace de comunicaciones entre la aplicación y una URL.
    - Clase HttpURLConnection. Representa una URLConnection con soporte para HTTP y con ciertas características especiales.

Mediante las clases de alto nivel se consigue una mayor abstracción, de manera que, esto facilita bastante la creación de programas que acceden a los recursos de la red.

# La clase URL

La clase **URL** (Uniform Resource Locator) representa un puntero a un recurso en la Web. Un recurso puede ser algo tan simple como un fichero o un directorio, o puede ser una referencia a un objeto más complicado, como una consulta a una base de datos o a un motor de búsqueda.

En general una URL que localiza recursos empleando el protocolo HTTP se divide en varias partes (las partes encerradas entre corchetes son opcionales): `http://host[:puerto]/nombredelpathdelservidor] [. ?argumentos]`

## FORMATO URL

protocolo :// nombrehost (: puerto) ruta (#referencia)

Esta clase tiene varios constructores:

CONSTRUCTOR	MISIÓN
<code>URL (String url)</code>	Crea un objeto URL a partir del String indicado en <i>url</i> .
<code>URL(String protocolo, String host, String fichero)</code>	Crea un objeto URL a partir de los parámetros <i>protocolo</i> , <i>host</i> y <i>fichero</i> (o directorio).
<code>URL(String protocolo, String host, int puerto, String fichero)</code>	Crea un objeto URL en el que se especifica el <i>protocolo</i> , <i>host</i> , <i>puerto</i> y <i>fichero</i> (o directorio) representados mediante String.
<code>URL(URL contexto, String especificación)</code>	Crea un objeto URL analizando la especificación dada dentro de un contexto específico.

# La clase URL - métodos

En el ejemplo **UT5\_Ejem1\_URL** se usan varios constructores de la clase URL y algunos de sus métodos.

Método más relevante →

MÉTODOS	MISIÓN
<b>String</b> getAuthority ()	Obtiene la autoridad del objeto URL.
<b>int</b> getDefaultPort()	Devuelve el puerto asociado por defecto al objeto URL.
<b>int</b> getPort()	Devuelve el número de puerto de la URL, -1 si no se indica.
<b>String</b> getHost()	Devuelve el nombre de la máquina.
<b>String</b> getQuery()	Devuelve la cadena que se envía a una página para ser procesada (es lo que sigue al signo? de una URL).
<b>String</b> getPath()	Devuelve una cadena con la ruta hacia el fichero desde el servidor y el nombre completo del fichero.
<b>String</b> getFile()	Devuelve lo mismo que <i>getPath ()</i> , además de la concatenación del valor de <i>getQuery()</i> si lo hubiese. Si no hay una porción consulta, este método y <i>getPath()</i> devolverán los mismos resultados.
<b>String</b> getProtocol()	Devuelve el nombre del protocol asociado al objeto URL.
<b>String</b> getUserInfo()	Devuelve la parte con los datos del usuario o nulo si no existe.
<b>InputStream</b> openStream()	Devuelve un <b>InputStream</b> del que podremos leer el contenido del recurso que identifica la URL.
<b>URLConnection</b> openConnection()	Devuelve un objeto <b>URLConnection</b> que nos permite abrir una conexión con el recurso y realizar operaciones de lectura y escritura sobre él.

# La clase URLConnection

---

Una vez que tenemos un objeto de la clase URL, si se invoca al método **openConnection()** para realizar la comunicación con el objeto y la conexión se establece satisfactoriamente, entonces tenemos una instancia de un objeto de la clase URLConnection:

```
URL url = new URL ("http://www.iesclaradelrey.es");  
URLConnection urlCon= url.openConnection();
```

La clase URLConnection es una clase abstracta que contiene métodos que permiten la comunicación entre la aplicación y una URL. Para conseguir un objeto de este tipo se invoca al método openConnection(), con ello obtenemos una conexión al objeto URL referenciado. Las instancias de esta clase se pueden utilizar tanto para leer como para escribir al recurso referenciado por la URL. Puede lanzar la excepción IOException.

Algunos de los métodos de esta clase son:



# La clase URLConnection

En el ejemplo **UT5\_Ejem2\_URLCon** se crea un objeto URL a la dirección `http://www.iesclaradelrey.es`, se invoca al objeto para crear una conexión y se obtiene un `URLConnection`.

Después se abre un stream de entrada sobre la conexión mediante el método `getInputStream()`.

El stream recogido se pinta por consola y corresponde con la página html del centro.

MÉTODOS	MISIÓN
<b>InputStream</b> <code>getInputStream()</code>	Devuelve un objeto <b>InputStream</b> para leer datos de esta conexión.
<b>OutputStream</b> <code>getOutputStream()</code>	Devuelve un objeto <b>OutputStream</b> para escribir datos en esta conexión.
<b>void setDoInput</b> (boolean b)	Permite que el usuario reciba datos desde la URL si el parámetro <i>b</i> es <i>true</i> (por defecto está establecido a <i>true</i> )
<b>void setDoOutput</b> (boolean b)	Permite que el usuario envíe datos si el parámetro <i>b</i> es <i>true</i> (no está establecido al principio)
<b>void connect()</b>	Abre una conexión al recurso remoto si tal conexión no se ha establecido ya.
<b>int</b> <code>getLength()</code>	Devuelve el valor del campo de cabecera <i>content-length</i> o -1 si no está definido.
<b>String</b> <code>getContentType()</code>	Devuelve el valor del campo de cabecera <i>content-type</i> o null si no está definido.
<b>long</b> <code>getDate()</code>	Devuelve el valor del campo de cabecera <i>date</i> o 0 si no está definido.
<b>long</b> <code>getLastModified()</code>	Devuelve el valor del campo de cabecera <i>last-modified</i>
<b>String</b> <code>getHeaderField(int n)</code>	Devuelve el valor del enésimo campo de cabecera especificado o null si no está definido.
<b>Map&lt;String, List&lt;String&gt;&gt;</b> <code>getHeaderFields()</code>	Devuelve una estructura Map (estructura de Java que nos permite almacenar pares clave/valor.) con los campos de cabecera. Las claves son cadenas que representan los nombres de los campos de cabecera y los valores son cadenas que representan los valores de los campos correspondientes.
<b>URL</b> <code>getURL()</code>	Devuelve la dirección URL.

# La clase HttpURLConnection

---

Esta clase proporciona los mecanismos para gestionar una conexión HTTP.

Hereda de URLConnection por lo que tiene los mismos métodos y además otros métodos de los que destacamos los más relevantes:

- ❑ `getResponseCode()`: se utiliza para recuperar el estado de respuesta del servidor.
- ❑ `setRequestMethod()`: se utiliza para establecer el método de solicitud. El valor predeterminado es GET.



# La clase HttpURLConnection

---

Hasta la versión 1.8 de java esta clase ha sido la base de la programación de aplicaciones capaces de acceder a recursos de la web. Los pasos a realizar con esta clase para una petición HTTP sin parámetros son las siguientes:

- ✓ Creación de la 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 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

# Ejemplo: Petición con HttpURLConnection

La web de la RAE permite consultar el significado de las palabras de lengua española. Desde la dirección:  
<https://www.rae.es/drae2001/>

Accedemos a una página con un formulario. Si escribimos la palabra cuyo significado buscamos simplemente navegaremos a una página web cuya raíz es igual que la anterior pero que se le añade la palabra que estamos buscando.



# Ejemplo: Petición con HttpURLConnection



**piedra.**

(Del lat. *petra*).

1. f. Sustancia mineral, más o menos dura y compacta, que no es terrosa ni de aspecto metálico.
2. f. Trozo de roca tallado para la construcción.

# Ejemplo: Petición con HttpURLConnection

```
20 public class PeticionHttpRAE {
21
22     public StringBuilder getContenido(String direccion) throws IOException {
23         StringBuilder respuesta = new StringBuilder();
24
25         URL url = new URL(direccion);
26         HttpURLConnection conexion;
27         conexion = (HttpURLConnection) url.openConnection();
28         conexion.setRequestMethod("GET");
29         conexion.setRequestProperty("Content-Type", "Text/plain");
30         conexion.setRequestProperty("charset", "utf-8");
31         conexion.setRequestProperty("User-Agent", "Chrome");
32
33         int estado = conexion.getResponseCode();
34         Reader streamReader;
35         if (estado==HttpURLConnection.HTTP_OK) {
36             streamReader = new InputStreamReader(conexion.getInputStream());
37             int caracter;
38             while ((caracter=streamReader.read())!= -1){
39                 respuesta.append((char)caracter);
40             }
41         }else {
42             System.out.println("Error Http "+ estado);
43         }
44     }
45 }
```

Recibimos la url a conectarnos.  
Definimos la variable StringBuilder

Creamos la URL

Abrimos la conexión

Configuramos la conexión

Capturamos la respuesta

Si todo ha ido OK...

Obtenemos el InputStream

y lo leemos

# Ejemplo: Petición con HttpURLConnection

---

En este método simplemente recibimos la ruta de mi Pc dónde vamos a almacenar el fichero con el html de respuesta y la respuesta que hemos obtenido en la conexión http. El método se encarga de escribir el fichero

```
public static void escribirFichero(String path, String contenido) {  
    Path mipath = Paths.get(path);  
    byte[] miBytes = contenido.getBytes();  
    try {  
        Files.write(mipath, miBytes);  
    } catch (IOException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}
```

# Ejemplo: Petición con HttpURLConnection

En el main simplemente pediremos la palabra a buscar, montaremos la url que queremos consultar y encadenaremos una llamada al método que obtiene el contenido y otra al método que escribe el fichero

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

    Scanner sc = new Scanner(System.in);
    System.out.println("¿Qué palabra quieres buscar?");
    // entrada de una cadena
    String recurso = sc.nextLine();

    PeticionHttpRAE miPeticion = new PeticionHttpRAE();
    String url = "https://www.rae.es/drae2001/";
    String direccion = url + recurso;

    StringBuilder resultado;
    try {
        resultado = miPeticion.getContenido(direccion);
        String ruta = "c:/julio/"+recurso+".html";
        miPeticion.escribirFichero(ruta, resultado.toString());
        System.out.println("descarga finalizada");
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

# Peticiones HTTP basadas en java.net.http

---

A partir de la versión 11 de java, se introdujo el paquete java.net.http para proporcionar una alternativa más potente, sencilla y actualizada de realizar peticiones HTTP.

Para la realización de peticiones convencionales se utilizan principalmente 3 clases

- ✓ HttpClient
- ✓ HttpRequest
- ✓ HttpResponse

Para realizar una petición HTTP se deben seguir 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 en el servidor.
- ❖ Crear el objeto HttpRequest, indicando la URI y los parámetros de cabecera de la petición
- ❖ Realizar la petición a través del método send del HttpClient y asignar las respuestas de la petición a un objeto HttpResponse
- ❖ Procesar la respuesta

Vemos al mismo ejemplo que antes utilizando estas clases: **UT5\_Ejem4\_HttpRAEv2**



# Ejemplo: Petición HTTP basada en java.net.http

Este es el mismo ejemplo que antes, la petición del significado de una palabra a la RAE, pero haciendo la petición con las clases HttpClient, HttpRequest y HttpResponse:

```
public class PeticionHttpRAE {  
    public int almacenarPagina(String url, String recurso, String path) throws Exception {  
        String direccion = url + recurso;  
        // Crear el objeto HttpClient, indicando versión del protocolo, así como otros  
        // datos opcionales  
        HttpClient myHttp = HttpClient.newBuilder().version(HttpClient.Version.HTTP_1_1)  
            .followRedirects(HttpClient.Redirect.NORMAL).build();  
        // Crear el objeto HttpRequest, indicando la URI y los parámetros de cabecera de la petición  
        HttpRequest myRequest = HttpRequest.newBuilder().GET().uri(URI.create(direccion))  
            .headers("Content-Type", "Text/plain").setHeader("User-Agent", "Chrome").build();  
        // Realizar la petición a través del método send del HttpClient y asignar las respuestas de la  
        // petición a un objeto HttpResponse  
        HttpResponse<Path> myResponse = myHttp.send(myRequest, HttpResponse.BodyHandlers  
            .ofFile(Path.of(path)));  
        return myResponse.statusCode();  
    }  
}
```



# Ejemplo: Petición HTTP basada en java.net.http

---

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

    String url = "https://www.rae.es/drae2001/";
    Scanner sc = new Scanner(System.in);

    System.out.println("¿Qué palabra quieres buscar?");

    // entrada de una cadena
    String recurso = sc.nextLine();

    PeticionHttpRAE miPeticion = new PeticionHttpRAE();
    String ruta = "c:/julio/"+recurso+".html";
    try {
        Integer resultado = miPeticion.almacenarPagina(url,recurso,ruta);

        System.out.println("descarga finalizada");
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```