Comunicaciones en red

Objetivos del tema:

Estudiar los sockets en Java. Aprender a crear y gestionar aplicaciones cliente-servidor comunicándose a través de sockets.

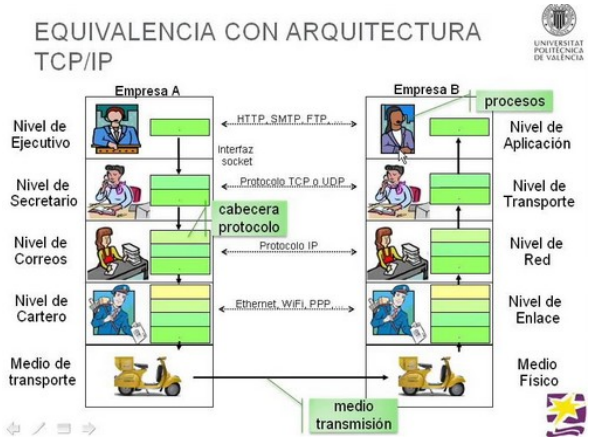
3.1.- Introducción

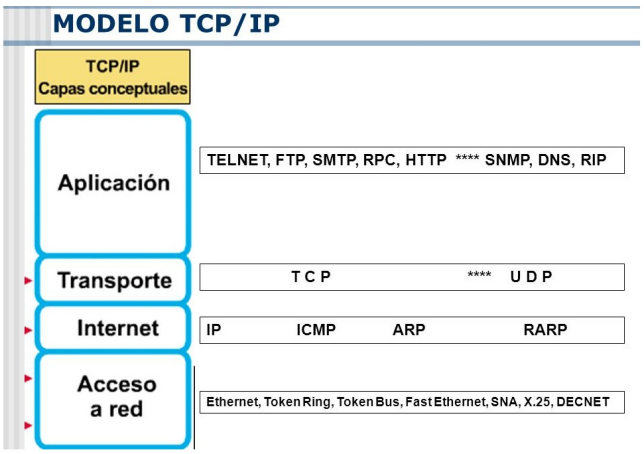
Java dispone de clases para establecer conexiones , crear servidores, enviar y recibir datos, y para el resto de las operaciones utilizadas en las comunicaciones a través de redes de ordenadores. Además, el uso de hilos, nos va a permitir la manipulación simultánea de múltiples conexiones.

3.2.- Clases java para comunicaciones en red

TCP/IP es una familía de protocolos desarrollados para permitir la comunicación entre cualquier par de ordenadores de cualquier red o fabricante, respetando los protocolos de cada red individual. Tiene cuatro capas:

- Capa de aplicación: en este nivel se encuentran las aplicaciones disponibles para los usuarios. Por ejemplo, FTP, SMTP, Telnet, HTTP, etc.
- Capa de transporte: suministra a las aplicaciones servicio de comunicaciones extremo a extremo utilizando dos tipos de protocolos: TCP (Transmission Control Protocol) y UDP (User Datagram Protocol).
- Capa de red: tiene como propósito seleccionar la mejor ruta para enviar paquetes por la red.El protocolo principal que funciona en esta capa es el Protocolo de Internet (IP).
- Capa de enlace o interfaz de red: es la interfaz con la red real. Recibe los datagramas de la capa de red y los transmite al hardware de la red.





Los equipos conectados a Internet se comunican entre si utilizando el protocolo TCP o UDP. Cuando se escriben aplicaciones Java que se comunican a través de la red, se está programando en la capa de aplicación . Normalmente , no es necesario preocuparse por las capas TCP y UDP , usando en su lugar las clases del paquete java.net. Sin embargo, hay diferencias entre la capa TCP y UDP que debemos conocer para decidir que clase usar en los programas.

- TCP: Protocolo basado en la conexión, garantiza que los datos enviados desde un extremo de la conexión llegan al otro extremo y en el mismo orden en que fueron enviados. De lo contrario, se notifica un error.
- UDP: No está basado en la conexión como TCP. Envía paquetes de datos independientes, denominados datagramas, de una aplicación a otra; el orden de entrega no es importante y no se garantiza la recepción de los paquetes enviados.

El paquete java.net contiene clases e interfaces para la implementación de aplicaciones de red. Estas incluyen:

- La clase URL , Uniform Resource Locator (Localizador Uniforme de Recursos). Representa un puntero a un recurso de la web.
- La clase URL Connection, que admite operaciones más complejas en las URL.
- Las clases ServerSocket y Socket , para dar soporte a sockets TCP.
  - ServerSocket: utilizada por el programa servidor para crear un socket en el puerto en el que escucha las peticiones de conexión de los clientes.
  - Socket: utilizada tanto por el cliente como por el servidor para comunicarse entre si leyendo y escribiendo datos usando streams.
- Las clases DatagramSocket, MulticastSocket y DatagramPacket para dar soporte a la comunicación via datagramas UDP.
- La clase InetAddress, que representa las direcciones de internet.

3.2.1.- Los Puertos

Los protocolos TCP y UDP usan puertos para asignar datos entrantes a un proceso en particular que se ejecuta en un ordenador.

En términos generales, un ordenador tiene una única conexión física a la red. Los datos destinados a este ordenador llegan a través de esa conexión . Sin embargo, los datos pueden estar destinados a diferentes aplicaciones que se ejecutan en el ordenador. Entonces, ¿cómo sabe el ordenador a qué aplicación enviar los datos? Mediante el uso de puertos.

Los datos transmitidos a través de Internet van acompañados de información de direccionamiento que identifica la máquina y el puerto para el que está destinada.
La maquina se identifica por su dirección IP . Los puertos se identifican mediante un número de 16 bits , que TCP y UDP utilizan para entregar los datos a la aplicación correcta.

En la comunicación basada en TCP , una aplicación de servidor vincula un socket a un número de puerto específico. Esto tiene el efecto de registrar el servidor en el sistema para recibir todos los

datos destinados a ese puerto. Una aplicación cliente puede entonces comunicarse con el servidor enviándole peticiones a través de ese puerto.
En la comunicación basada en datagramas, como UDP , el paquete de datagrama contiene el número de puerto de su destino y UDP enruta el paquete a la aplicación adecuada.

3.2.2.- La clase InetAddress

La clase InetAddress es la abstracción que representa una dirección IP . Tiene dos subclases:

- Inet4Address para direcciones IPv4
- Inet6Address para direcciones IPv6

MÉTODOS	MISIÓN
InetAddress getLocalHost()	Devuelve un objeto *InetAddress* que representa la dirección IP de la máquina donde se esta ejecutando el programa.
InetAddress getByName(String host)	Devuelve un objeto *InetAddress* que representa la dirección IP de la máquina que se especifica como parámetro (*host*). Este parámetro puede ser el nombre de la máquina, un nombre de dominio o una dirección IP.
InetAddress[] getAllByName(String host)	Devuelve un array de objetos de tipo *InetAddress*. Este método es útil para averiguar todas las direcciones IP que tenga asignada una máquina en particular.
String getHostAddress()	Devuelve la dirección IP de un objeto *InetAddress* en forma de cadena.
String getHostName()	Devuelve el nombre del host de un objeto *InetAddress*.
String getCanonicalHostName()	Obtiene el nombre canónico completo (suele ser la dirección real del host) de un objeto *InetAddress*.

Los tres primeros métodos pueden lanzar la excepción UnknownHostException (Cuando se lanza esta excepción , nos indica que no se pudo determinar la dirección IP del host)

La forma más típica de crear instancias de InetAddress, es invocando al método estático getByName(String) pasándole el nombre DNS del host como parámetro. Este objeto representará la dirección IP de ese host , y *se podrá utilizar para construir sockets*

Ejercicio 1 (Clase InetAddress):

1. Definir un objeto **InetAddress** de nombre **dir**. En primer lugar usarlo para obtener la dirección IP de la maquina local en la que se ejecuta el programa (localhost / 127.0.0.1).
2. Crear un método "`private static void pruebaMetodos(InetAddress dir)`", que implemente los métodos de la clase InetAddress del cuadro anterior.
   - `GetByName()`
   - `getLocalHost()`
   - `getHostName()`
   - `getHostAddress()`
   - `toString()`
   - `getCanonicalHostName()`
3. Hacer una llamada a cada método con el objeto dir creado.

4. Usar el objeto para obtener la dirección IP de la URL www.google.es y volver a invocar a "pruebaMetodos()".
5. Usar el método getAllByName() para ver todas las direcciones IP asignadas a la máquina representada por www.google.es (hacerlo en un bloque try-catch)

Salida

```
=======================================================
SALIDA PARA LOCALHOST:
       Metodo getByName():  localhost/127.0.0.1
       Metodo getLocalHost(): Juan-PC/192.168.0.12
       Metodo getHostName(): localhost
       Metodo getHostAddress(): 127.0.0.1
       Metodo toString(): localhost/127.0.0.1
       Metodo getCanonicalHostName(): 127.0.0.1
=======================================================
SALIDA PARA UNA URL:
       Metodo getByName():  www.google.es/216.58.215.131
       Metodo getLocalHost(): Juan-PC/192.168.0.12
       Metodo getHostName(): www.google.es
       Metodo getHostAddress(): 216.58.215.131
       Metodo toString(): www.google.es/216.58.215.131
       Metodo getCanonicalHostName(): mad41s04-in-f3.1e100.net
       DIRECCIONES IP PARA: www.google.es
              www.google.es/216.58.215.131
=======================================================
```

3.2.3.- La clase URL

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

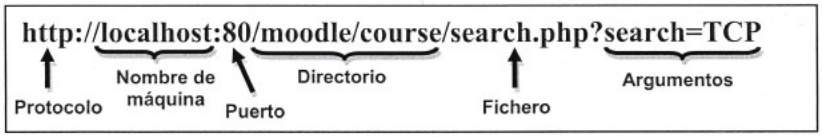
En general una URL que localiza recursos empleando el protocolo HTTP se divide en varias partes:

http://host[:puerto][/nombre del path del servidor][?argumentos]

Las partes encerradas entre corchetes son opcionales

- **host**: Es el nombre de la maquina en la que reside el recurso.
- **[:puerto]** Número de puerto en el que el servidor escucha las peticiones. Este parámetro es opcional y si no se indica, se considera el puerto por defecto (para el protocolo http es el puerto 80)
- **[/dirección del servidor]** Es el path o directorio donde se encuentra el recurso en el sistema de ficheros del servidor. Si no se indica, se proporciona la página por defecto del servidor web.
- **[?argumentos]** Parámetros que se envía al servidor. Por ejemplo, cuando realizamos una consulta se puede enviar parámetros a un fichero PHP para procesarla.

Ejemplo:



El fichero "search.php" está en el directorio "/moodle/course" dentro del servidor, y el argumento "search=TCP" que se envía al fichero "search.php" para realizar una búsqueda.

Constructores de la clase URL

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

Estos constructores pueden lanzar la excepción "MalformedURLException" si la URL está mal construida, no se hace ninguna verificación de que realmente exista la máquina o el recurso en la red.

Algunos Métodos de la clase URL

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

Ejercicio 2 (Uso de los constructores)

Nota: El método Visualizar() muestra información de la URL usando los métodos de la tabla anterior.

Crea la clase "Ejemplo1URL" que contendrá el main y el método "private static void Visualizar(URL url)" .
El main creará un objeto de la clase URL llamado url (URL url), usando cada uno de los constructores de la tabla anterior y hará una llamada al método visualizar para cada constructor, pasandole el objeto url (Visualizar (url))

```
public static void main(String[] args) {
        URL url;
        try {
          System.out.println("Constructor simple para una URL:");
          // usar la url http://docs.oracle.com/



          System.out.println("Otro constructor simple para una URL:");
          // usar la url http://localhost/PFC/gest/cli_gestion.php?S=3



          System.out.println("Const. para protocolo +URL + directorio:");
          // usar la url docs.oracle.com , el directorio /javase/9 y el protocolo http



          System.out.println("Constructor para protocolo + URL + puerto + directorio:");
    // usar:  url "localhost" , puerto "8084", directorio "/WebApp/Controlador?accion=modificar"

        System.out.println("Constructor para un objeto URL en un contexto:");

        // url base para el contexto "https://docs.oracle.com/"
        // especificación  "/javase/9/docs/api/java/net/URL.html"

        } catch (MalformedURLException e) {        System.out.println(e);}
 }// main
```

En el método visualizar usaremos los métodos siguientes:

- toString() para mostrar la url completa
- getProtocol()
- getHost()
- getPort()
- getFile()

- getUserInfo()
- getPath()
- getAuthority()
- getQuery()
- getDefaultPort()

Comentar los resultados

SALIDA

```
Constructor simple para una URL:
       URL completa: http://docs.oracle.com/
       getProtocol(): http
       getHost(): docs.oracle.com
       getPort(): -1
       getFile(): /
       getUserInfo(): null
       getPath(): /
       getAuthority(): docs.oracle.com
       getQuery(): null
       getDefaultPort(): 80
================================================
Otro constructor simple para una URL:
       URL completa: http://localhost/PFC/gest/cli_gestion.php?S=3
       getProtocol(): http
       getHost(): localhost
       getPort(): -1
       getFile(): /PFC/gest/cli_gestion.php?S=3
       getUserInfo(): null
       getPath(): /PFC/gest/cli_gestion.php
       getAuthority(): localhost
       getQuery(): S=3
       getDefaultPort(): 80
================================================
Const. para protocolo +URL + directorio:
       URL completa: http://docs.oracle.com/javase/9
       getProtocol(): http
       getHost(): docs.oracle.com
       getPort(): -1
       getFile(): /javase/9
       getUserInfo(): null
       getPath(): /javase/9
       getAuthority(): docs.oracle.com
       getQuery(): null
       getDefaultPort(): 80
================================================
Constructor para protocolo + URL + puerto + directorio:
       URL completa: http://localhost:8084/WebApp/Controlador?accion=modificar
       getProtocol(): http
       getHost(): localhost
       getPort(): 8084
       getFile(): /WebApp/Controlador?accion=modificar
       getUserInfo(): null
       getPath(): /WebApp/Controlador
```

```
       getAuthority(): localhost:8084
       getQuery(): accion=modificar
       getDefaultPort(): 80
================================================
Constructor para un objeto URL en un contexto:
       URL completa: https://docs.oracle.com/javase/9/docs/api/java/net/URL.html
       getProtocol(): https
       getHost(): docs.oracle.com
       getPort(): -1
       getFile(): /javase/9/docs/api/java/net/URL.html
       getUserInfo(): null
       getPath(): /javase/9/docs/api/java/net/URL.html
       getAuthority(): docs.oracle.com
       getQuery(): null
       getDefaultPort(): 443
```

Ejercicio 3 Crear objeto URL

Crear la clase Ejemplo2URL que solo contendrá el método main en que se realizará lo siguiente:

- Crear un objeto URL a la dirección http://www.elaltozano.es
- Abrir una conexión con él creando un objeto InputStream y usarlo como flujo de entrada para leer los datos de la página inicial del sitio; al ejecutar el programa se muestra en pantalla el código HTML de la página inicial del sitio.

```
import java.net.*;
import java.io.*;

public class Ejemplo2URL {
  public static void main(String[] args) {

       try {


       } catch (MalformedURLException e) {     e.printStackTrace();}


       try {


       } catch (IOException e) {e.printStackTrace();}
  }//
}
```

3.2.4.- 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 URLConnetion.

Ejemplo:

```
URL url = new URL ("http://www.elaltozano.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 métodos de la clase URLConnection

MÉTODOS	MISIÓN
InputStream getInputStream()	Devuelve un objeto **InputStream** para leer datos de esta conexión.
OutputStream getOutputStream()	Devuelve un objeto **OutputStream** para escribir datos en esta conexión.
void setDoInput (boolean b)	Permite que el usuario reciba datos desde la URL si el parámetro *b* es *true* (por defecto está establecidoa *true*)
void setDoOutput((boolean b)	Permite que el usuario envíe datos si el parámetro *b* es *true* (no está establecido al principio)
void connect()	Abre una conexión al recurso remoto si tal conexión no se ha establecido ya.
int getContentLength()	Devuelve el valor del campo de cabecera *content-lenght* o -1 si no está definido.
String getContentType()	Devuelve el valor del campo de cabecera *content-type* o null si no está definido.
long getDate()	Devuelve el valor del campo de cabecera *date* o 0 si no está definido.
long getLastModified()	Devuelve el valor del campo de cabecera *last-modified*
String getHeaderField(int n)	Devuelve el valor del enésimo campo de cabecera especificado o null si no está definido.
Map< String, List<String> > getHeaderFields()	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.
URL getURL()	Devuelve la dirección URL.

Ejercicio 4 URLConnection

- Crear una clase Ejemplo1urlCon que contiene el método main.
- Crear un objeto URL a la dirección http://www.elaltozano.es , e invocar al método openConnection() del objeto para crear una conexión y obtener una instancia de URLConnection.
- Abrir un stream de entrada sobre esa conexión mediante el método getInputStream().

La salida que se obtiene es la misma que en el ejemplo anterior, con la diferencia de que en este ejemplo se crea una conexión con el recurso representado por la URL y el anterior abre directamente un stream desde la URL.

Formularios

Gran cantidad de páginas HTML contienen formularios a través de los cuales podemos solicitar información a un servidor rellenando los campos requeridos y pulsando el botón de envío. El servidor recibe la petición, la procesa y envía los datos solicitados al cliente normalmente en formato HTML. Por ejemplo, tenemos una página HTML que contiene un formulario con dos campos de entrada y un botón. En el atributo "action" se indica el tipo de acción que va a realizar el formulario, en este caso los datos se envían a un script PHP de nombre "vernombre.php". Con (method=post) indicamos la forma en que se envía el formulario.

```
<html>
<body>
<form action="vernombre.php" method="post">
<p> Escribe tu nombre:
<input name="nombre" type="text" size="15"> </p>
<p> Escribe tus apellidos:
<input name="apellidos" type="text" size="15"></p>
<input type="submit" name="ver" value="Ver">
</form>
</body>
</html>
```

El script PHP que recibe los datos del formulario es el siguiente:

```
<?php
$nom=$_POST["nombre"];
$ape=$_POST["apellidos"];
echo "El nombre recibido es: $nom, y ";
echo "los apellidos son: $ape ";
?>
```

En él se reciben los valores introducidos en los campos nombre y apellidos del formulario, mediante la instrucción $_POST["nombrecampo"], y se visualiza en la pantalla del navegador mediante la orden "echo".
La URL para enviar datos al formulario, suponiendo que los ficheros ".html" y ".php" estén en la carpeta 2021 del servidor web local sería similar a la siguiente:

http://localhost/2021/vernombre.php?nombre=Juan&apellidos=Martínez&ver=Ver

Desde Java, usando la clase URLConnection podemos interactuar con scripts del lado del servidor y podemos enviar valores a los campos del script sin necesidad de abrir un formulario HTML, será necesario escribir en la URL para dar los datos al script.

¿Qué tiene que realizar nuestro programa?

- Crear el objeto URL al script con el que va a interactuar. Por ejemplo, en nuestra máquina local tenemos instalado un servidor web Apache y dentro de htdocs tenemos la carpeta 2021 con el script PHP "vernombre.php", la URL sería la siguiente:
  URL url = new URL(http://localhost/2021/vernombre.php***)***

- Abrir una conexión con la URL, es decir obtener el objeto URLConnection:
  URLConnection conexión = url.openConnection()

- Configurar la conexión para que se puedan enviar datos usando el método setDoOutput():
  conexión.setDoOutput(true)

- Obtener un stream de salida sobre la conexión
  PrintWriter output = new PrintWriter(conexion.getOutputStream())

- Escribir en el stream de salida, en este caso mandamos una cadena con los datos que necesita el script
  output.write(cadena)

  La cadena tiene el siguiente formato:

  parámetro=valor

  Si el script recibe varios parámetros sería:

  parámetro1=valor1&parámetro2=valor2&parámetro3=valor3 , y así sucesivamente

- Cerrar el stream de salida
  output.close()

Nota: *Normalmente cuando se pasa información a algún script PHP, éste realiza alguna acción y después envía la información de vuelta por la misma URL. Por tanto si queremos ver lo que devuelve será necesario leer desde la URL. Para ello se abre un stream de entrada sobre esa conexión mediante el método getInputStream()*

BufferedReader reader = new BufferedReader(new InputStreamReader(conexion.getInputStream())))

Después se realiza la lectura para obtener los resultados devueltos por el script.

Ejercicio 5 URLConnection

Crea la clase "Ejemplo2urlCon" que contendrá solo el método main para realizar todos los puntos explicado anteriormente.

Nota: Para este ejercicio debemos de tener instalado en nuestra máquina el servidor web Apache.

```
import java.io.*;
import java.net.*;

public class Ejemplo2urlCon {
  public static void main(String[] args) {
    try {
       // Crear objeto URL con "http://localhost/2018/vernombre.php"


      String cadena ="nombre=Juan&apellidos=Ramos Martín";

      //ESCRIBIR EN LA URL - stream de salida

       //cerrar flujo

      //LEER DE LA URL - stream de entrada


       //cerrar flujo


    } catch (MalformedURLException me) {

    } catch (IOException ioe) {

    }
  }//main
}
```

Ejercicio 6: Prueba de métodos de la clase URLConnection

Creamos la clase Ejemplo3urlCon para probar los métodos de la clase URL Connection

```
import java.net.*;
import java.io.*;
import java.util.*;

public class Ejemplo3urlCon {
  @SuppressWarnings("rawtypes")
  public static void main(String[] args) throws Exception {
   String cadena;
   URL url = new URL("http://localhost/2018/vernombre.html");
   URLConnection conexion = url.openConnection();

   System.out.println("Direccion [getURL()]:" + conexion.getURL());

   Date fecha = new Date(conexion.getLastModified());
   System.out.println("Fecha ultima modificacion [getLastModified()]: " + fecha);
   System.out.println("Tipo de Contenido [getContentType()]: "
                        + conexion.getContentType());

   System.out.println("========================================== ");
   System.out.println("TODOS LOS CAMPOS DE CABECERA CON getHeaderFields(): ");

   //USAMOS UNA ESTRUCTURA Map PARA RECUPERAR CABECERAS
```

```
   Map camposcabecera = conexion.getHeaderFields();
   Iterator it = camposcabecera.entrySet().iterator();
   while (it.hasNext()) {
       Map.Entry map = (Map.Entry) it.next();
       System.out.println(map.getKey() + " : " + map.getValue());
   }

   System.out.println("========================================== ");
   System.out.println("CAMPOS 1 Y 4 DE CABECERA:");
   System.out.println("getHeaderField(1)=> "+
conexion.getHeaderField(1));
   System.out.println("getHeaderField(4)=> " +
                           conexion.getHeaderField(4));
   System.out.println("==========================================");

   System.out.println("CONTENIDO DE [url.getFile()]:"+url.getFile());
   BufferedReader pagina = new BufferedReader
             (new InputStreamReader(url.openStream()));

   while ((cadena = pagina.readLine()) != null) {
             System.out.println(cadena);
   }
  }
}//
```

NOTA: Para recorrer una estructura Map podemos usar la interfaz "Iterator". Para obtener un iterador sobre el map se invoca a los métodos entrySet() e iterator().
El método entrySet() devuelve un Set de objetos Map.Entry que contiene los pares. Para mover el iterador utilizaremos el método next() y para comprobar si ha llegado al final usamos el método hasNext() . De la estructura recuperaremos los valores mediante getKey(), para la clave y getValue(), para el valor.