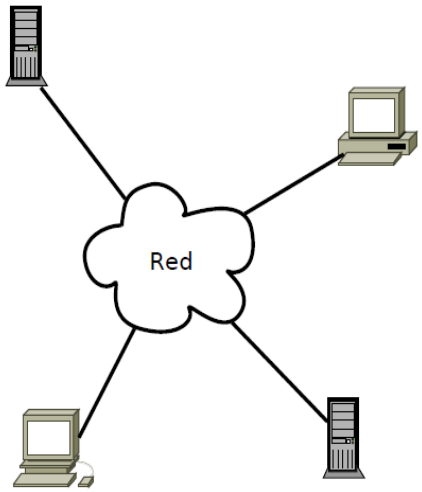
PROGRAMACIÓN DE COMUNICACIONES EN RED.

Índice de contenido

1. INTRODUCCIÓN

En los orígenes de la informática, la computación estaba concebida como grandes ordenadores centrales localizados en lugares específicos, como universidades, laboratorios, etc., y aislados entre sí. Con la proliferación de los ordenadores personales y la red Internet, décadas después, surgió una nueva forma de concebir la computación, en la que múltiples computadores colaboran entre sí, comunicándose a través de una red.



Antiguamente la programación de aplicaciones que comunican diferentes máquinas era dificil, compleja y fuente de muchos errores; el programador tenía que conocer detalles sobre las capas del protocolo de red, incluso sobre el hardware de la máquina.

El uso de las librerías Java hacen que la programación en red para comunicar distintas máquinas no sea una tarea tan compleja. Java dispone de clases para establecer conexiones, crear servidores, enviar y recibir datos, y para el resto de 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.

2. PROTOCOLOS DE COMUNICACIONES.

TCP/IP es una familia 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 4 capas o niveles de abstracción:

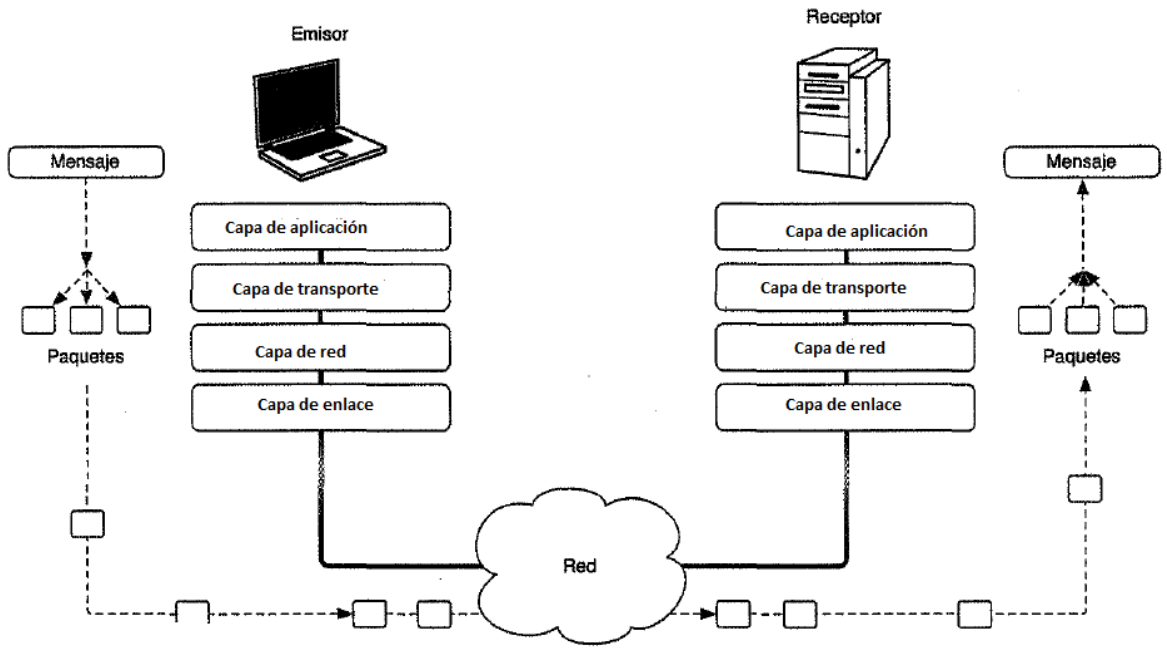
- **Capa de aplicación:** en este nivel se encuentran las aplicaciones disponibles para los usuarios. Por ejemplo FTP, SMTP, Telnet, HTTP, etc.
- **Capa de transporte:** suministra a las aplicaciones servicio comunicaciones extremo a extremo utilizando dos tipos de protocolos: TCP (Transmission Control Protocol) y UDP (User Datagram Protocol). Su función es crear el canal de comunicación, descomponer el mensaje en paquetes y gestionar su transmisión entre el emisor y el receptor. Se encarga de que la información se envíe a la aplicación adecuada (mediante un determinado puerto).

- **Capa de red:** tiene como proposito 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, capa host-red:** es la interfaz con la red real. Recibe los datagramas de la capa de red y los transmite al hardware de la red. Permite comunicar el ordenador con el medio que conecta el equipo a la red. Para ello primero debe permitir convertir la información en impulsos físicos (p.ej. eléctricos, magnéticos, luminosos) y además, debe permitir las conexiones entre los ordenadores de la red. En esta capa se realiza un direccionamiento físico utilizando las direcciones MAC.

Las aplicaciones de un sistema distribuido se sitúan en el nivel superior (nivel de aplicación). Cuando una aplicación emisora decide enviar un mensaje se produce la siguiente secuencia de operaciones:

1. La aplicación del emisor entrega el mensaje al nivel inmediatamente inferior, es decir, la capa de transporte.
2. El protocolo del nivel de transporte descompone el mensaje en paquetes, y los pasa la capa inferior (capa de red).
3. El nivel de red localiza al receptor del mensaje y calcula la ruta que deben seguir los paquetes para llegar a su destino. Una vez hecho esto, entrega los paquetes al nivel inferior (capa de enlace).
4. El nivel de enlace transmite los paquetes hasta el receptor.
5. Una vez los paquetes van llegando al receptor, el nivel de enlace los recibe y los pasa a su nivel superior (capa de red).
6. El nivel de red comprueba que los paquetes recibidos han llegado al destinatario (receptor) correcto. Si es así, los envía al nivel superior (capa de transporte).
7. El nivel de transporte agrupa los paquetes recibidos para formar el mensaje. Una vez el mensaje ha sido reconstruido, lo envía al nivel superior (capa de aplicación). Por último, la aplicación receptora recibe el mensaje.

La capa de transporte cumple la función de establecer las reglas necesarias para establecer una conexión entre dos dispositivos.



Desde la capa anterior, la capa de red, la información se recibe en forma de paquetes desordenados y la capa de transporte debe ser capaz de manejar dichos paquetes y obtener un único flujo de datos. La capa de red en la arquitectura TCP/IP no se preocupa del orden de los paquetes ni de los errores, es en esta capa donde se deben cuidar estos detalles.

La capa de transporte del modelo TCP/IP es equivalente a la capa de transporte del modelo OSI, por lo que es el encargado de la transferencia libre de errores de los datos entre el emisor y el receptor, aunque no estén directamente conectados, así como de mantener el flujo de la red. La tarea de este nivel es proporcionar un transporte de datos confiable de la máquina de origen a la máquina destino, independientemente de la red física.

Existen dos tipos de conexiones:

TCP (Transmission Control Protocol). Es un protocolo *orientado a la conexión* que permite que un flujo de bytes originado en una máquina se entregue **sin errores** en cualquier máquina destino. Este protocolo fragmenta el flujo entrante de bytes en mensajes y pasa cada uno a la capa de red. En el destino, el proceso TCP receptor reensambla los mensajes recibidos para formar el flujo de salida. TCP también se encarga del control de flujo para asegurar que un emisor rápido no pueda saturar a un receptor lento con más mensajes de los que pueda gestionar. Garantiza que los datos enviados desde un extremo de la conexión llega al otro extremo y en el mismo orden en que fueron enviados. De lo contrario, se notifica un error.

UDP (User Datagram Protocol). Es un protocolo *sin conexión*, para aplicaciones que no necesitan la asignación de secuencia ni el control de flujo TCP y que desean utilizar los suyos propios. 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.Este protocolo también se utilizan para las consultas de petición y respuesta del tipo cliente-servidor, y en aplicaciones en las que la velocidad es más importante que la entrega precisa, como las transmisiones de voz o de vídeo. Uno de sus usos es en la transmisión de audio y vídeo en tiempo real, donde no es posible realizar retransmisiones por los estrictos requisitos de retardo que se tiene en estos casos.

De esta forma, a la hora de programar nuestra aplicación deberemos elegir el protocolo que queremos utilizar según nuestras necesidades: **TCP** o **UDP**.

Con la capa de red se consigue que la información vaya de un equipo origen a un equipo destino a través de su dirección IP. Pero para que una aplicación pueda comunicarse con otra aplicación es necesario establecer a qué aplicación se conectará. El método que se emplea es el de definir direcciones de transporte en las que los procesos pueden estar a la escucha de solicitudes de conexión. Estos puntos terminales se llaman **puertos**.

Aunque muchos de los puertos se asignan de manera arbitraria, ciertos puertos se asignan, por convenio, a ciertas aplicaciones particulares o servicios de carácter universal. De hecho, la IANA[1] (Internet Assigned Numbers Authority) determina, las asignaciones de todos los puertos. Existen tres rangos de puertos establecidos:

- **Puertos conocidos [0, 1023].** Son puertos reservados a aplicaciones de uso estándar como: 21 – FTP (File Transfer Protocol), 22 – SSH (Secure SHell), 53 – DNS (Servicio de nombres de dominio), 80 – HTTP (Hypertext Transfer Protocol), etc.
- **Puertos registrados [1024, 49151].** Estos puertos son asignados por IANA para un servicio específico o aplicaciones. Estos puertos pueden ser utilizados por los usuarios libremente.
- **Puertos dinámicos [49152, 65535].** Este rango de puertos no puede ser registrado y su uso se establece para conexiones temporales entre aplicaciones.

1 Internet Assigned Numbers Authority

2.1. Nombres en internet.

Los equipos informáticos se comunican entre sí mediante una **dirección IP** como 193.144.43.238. Sin embargo nosotros preferimos utilizar nombres como www.iessanclemente.net porque son más fáciles de recordar y porque ofrecen la flexibilidad de poder cambiar la máquina en la que están alojados (cambiaría entonces la dirección IP) sin necesidad de cambiar las referencias a él.

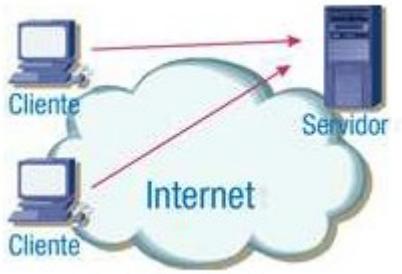
El servidor DNS utiliza una base de datos distribuida y jerárquica que almacena información asociada a nombres de dominio en redes como Internet.

A la hora de comunicarse con un equipo, puedes hacerlo directamente a través de su dirección IP o puede poner su entrada DNS (p.ej. servidor.miempresa.com). En el caso de utilizar la entrada DNS el equipo resuelve automáticamente su dirección IP a través del servidor de nombres que utilice en su conexión a Internet.

2.2. Modelos de comunicaciones.

Sin duda alguna, el modelo de comunicación que ha revolucionado los sistemas informáticos es el modelo **cliente/servidor**. El modelo cliente/servidor esta compuesto por un servidor que ofrece una serie de servicios y unos clientes que acceden a dichos servicios a través de la red.

Por ejemplo, el servicio más conocido de Internet, el WWW , utiliza el modelo cliente/servidor. Por un lado tenemos el servidor que alojan las páginas web y por otro lado los clientes que solicitan al servidor una determinada página web.



Modelo Cliente/Servidor

Modelo Sistema de Información Distribuido

Otro modelo ampliamente utilizado son los **Sistemas de Información Distribuidos**. Un Sistema de Información Distribuido esta compuesto por un conjunto de equipos que interactúan entre sí y pueden trabajar a la vez como cliente y servidor. Desde el punto de vista externo es igual que un sistema cliente/servidor ya que el cliente ve al Sistema de Información Distribuido como una entidad.

Internamente, los equipos del Sistema de Información Distribuido interactúan entre sí (actuando como servidores y clientes de forma simultánea) para compartir información, recursos, realizar tareas, etc.

3. CLASES JAVA PARA COMUNICACIONES EN RED.

Los equipos conectados a Internet se comunican entre sí utilizando el protocolo TCP o UDP.

Cuando se escriben programas 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; en su lugar, se pueden utilizar las clases del paquete **java.net**. Sin embargo es necesario saber las diferencias entre una y otra que conviene saber para decidir qué clases usar en los programas.

Resumiendo lo que vimos anteriormente, vemos que principalmente se diferencian:

- **TCP:** protocolo basado en la conexión, garantiza que los datos enviados desde un extremo de la conexión llega 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** proporciona las clases para la implementación de aplicaciones de red.

3.1. LA CLASE InetAddress

La clase InetAddress es la abstracción que representa una dirección IP (Internet Protocol).

Tiene dos subclases: Inet4Address para direcciones IPv4 e Inet6Address para direcciones IPv6;pero en la mayoría de los casos InetAddress aporta la funcionalidad necesaria y no es necesario recurrir a ellas. En la siguiente tabla se muestran algunos métodos importantes de esta clase:

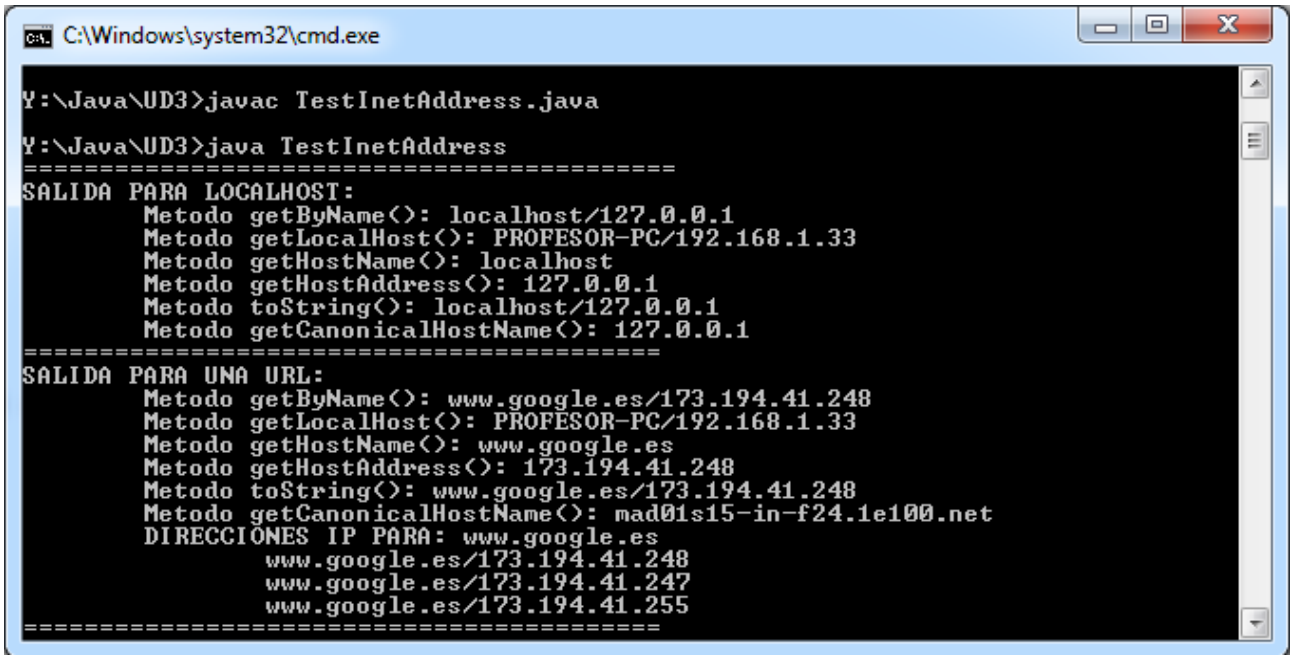
MÉTODOS	MISIÓN
InetAdress getLocalHost()	Devuelve un objeto **InetAddress** que representa la dirección IP de la máquina donde se está ejecutando el programa
InetAdress getByName(String host)	Devuelve un objeto **InetAddress** que representa la dirección IP de la máquina que se especifica como parámetro (*host*). Este parámetro puede ser el nombre de la máquina, un nombre de dominio o una dirección IP
InetAdress[] 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 getHostAdress()	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[2] completo (suele ser la dirección real del host) de un objeto *InetAddress*

Los 3 primeros métodos pueden lanzar la excepción **UnknownHostException**. En el siguiente ejemplo se define un objeto **InetAddress** de nombre *dir.* En primer lugar lo utilizamos para obtener la dirección IP de la máquina local en la que se ejecuta el programa. A continuación llamamos al método *pruebaMetodos()* llevando el objeto creado. En dicho método se prueban los métodos de la clase **InetAddress**. Después utilizamos el objeto para obtener la dirección IP de la URL *www.google.es* y volvernos a invocar a *pruebaMetodos()* (para que funcione en este segundo caso necesitamos estar conectados a Internet). Por último utilizarnos el método *getAllByName()* para ver todas las direcciones IP asignadas a la máquina representada por *www.google.es*. Se encierra todo en un bloque try-catch:

2 Se usa para crear nombres de servidores de alojamiento adicionales, o alias, para los servidores de alojamiento de un dominio. Es usado cuando se están corriendo múltiples servicios (como ftp y servidor web) en un servidor con una sola dirección ip. Cada servicio tiene su propia entrada de DNS (como ftp.ejemplo.com. y www.ejemplo.com.). esto también es usado cuando corres múltiples servidores http, con diferente nombres, sobre el mismo host. Se escribe primero el alias y luego el nombre real.

```java
import java.net.*;
public class TestInetAddress {
    public static void main(String[] args) {
        InetAddress dir = null;
        System.out.println("=============================================");
        System.out.println("SALIDA PARA LOCALHOST: ");
        try {
            //LOCALHOST
            dir = InetAddress.getByName("localhost")            ;
            pruebaMetodos(dir) ;//

            //URL www.google.es
            System.out.println("=============================================");
            System.out.println("SALIDA PARA UNA URL:");
            dir = InetAddress.getByName("www.google.es");
            pruebaMetodos(dir) ;

            // Array de tipo InetAddress con todas las direcciones IP
            //asignadas a google.es
            System.out.println (" \tDIRECCIONES IP PARA: " + dir.getHostName()) ;
            InetAddress[] direcciones =
                InetAddress.getAllByName(dir.getHostName());
            for (int i = 0; i < direcciones.length; i++)
                System.out.println("\t\t"+direcciones[i].toString());
            System.out.println("=============================================");
        } catch (UnknownHostException el) {el.printStackTrace();}
    }// main
    //
    private static void pruebaMetodos(InetAddress dir) {
        System.out.println("\tMetodo getByName(): " + dir);
        InetAddress dir2;
        try {
            dir2 = InetAddress.getLocalHost();
            System.out.println("\tMetodo getLocalHost(): " + dir2);
            }
        catch (UnknownHostException e) {e.printStackTrace();}
        // USAMOS METODOS DE LA CLASE
        System.out.println("\tMetodo getHostName(): "+dir.getHostName());
        System.out.println("\tMetodo getHostAddress(): "+
        dir.getHostAddress()) ;
        System.out.println("\tMetodo toString(): " + dir.toString());
        System.out.println("\tMetodo getCanonicalHostName(): " +
        dir.getCanonicalHostName());
    }//pruebaMetodos
}//fin
```

La salida generada debería ser algo así:



3.2. LA CLASE URL

La clase **URL** *(Uniform Resource Locator) r*epresenta 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 se divide en varias partes. Por ejemplo en la siguiente URL:

http://www.iessanclemente.net/wp-content/uploads/2013/04/BRIC.pdf.

encontramos el protocolo (*http*), el nombre de máquina (*www.iessanclemente.net*) y el fichero (*BRIC.pdf*) que está en un directorio dentro del servidor (*wp-content/uploads/2013/04*).

Una URL puede especificar opcionalmente un puerto (punto de destino para la comunicación dentro de una máquina) para realizar la conexión TCP. Por defecto para el protocolo HTTP es el 80, la URL anterior con el puerto sería:

http://www.iessanclemente.net:80/wp-content/uploads/2013/04/BRIC.pdf

La clase **URL** contiene varios constructores, algunos son:

CONSTRUCTOR	MISIÓN
URL (String url)	Crea un objeto URL a partir del String *url*
URL(String protocolo, String host, String fichero)	Crea un objeto URL a partir de los parámetros *protocolo, host* y *fichero*
URL(String protocolo, String host, int puerto, String fichero)	Crea un objeto URL en el que se especifica el protocolo, host, puerto y fichero representados mediante String
URL(URL context, String url)	Crea un objeto URL a partir de la dirección del host dada por *URL context* y una URL relativa dada en el String (un directorio)

Estos 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 de los métodos de la clase URL son los siguientes:

MÉTODOS	MISIÓN
String getAuthority ()	Obtiene la autoridad[3] 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 getUserInfo()	Devuelve la parte con los datos del usuario de la dirección URL o nulo si no existe
InputStream openStream()	Abre una conexión al objeto URL y devuelve un **InputStream** para la lectura de esa conexión
URLConnection openConnection()	Devuelve un objeto **URLConnection** que representa la conexión a un objeto remoto referenciado por la URL

3 Autoridad: elemento jerárquico que identifica la autoridad de nombres (por ejemplo //www.example.com).

El siguiente ejemplo muestra el uso de los constructores definidos anteriormente; el método Visualizar() muestra información de la URL usando los métodos de la tabla anterior:

```
import java.net.*;
public class Ejemplo1URL {
    public static void main(String[] args) {
        URL url;
        try {
            System.out.println("Constructor simple para una URL:");
            url = new URL("http://docs.oracle.com/") ;
            Visualizar(url) ;

            System.out.println("Otro constructor simple para una URL:");
            url = new URL("http://www.circuitodejerez.com/index.php?id=196");
            Visualizar (url) ;

            System.out.println("Const. para protocolo +URL +directorio:");
            url = new URL("http", "docs.oracle.com", "/javase/7");
            Visualizar (url) ;

            System.out.println("Constructor para protocolo + URL + puerto + directorio:");
            url = new URL("http", "docs.oracle.com", 80, "/javase/7");
            Visualizar (url) ;

            System.out.println("Constructor para un objeto URL y un directorio:");
            URL urlBase = new URL("http://docs.oracle.com/");
            url = new URL(urlBase, "/javase/7/docs/api/java/net/URL.html");
            Visualizar (url) ;

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

    private static void Visualizar(URL url) {
        System.out.println("\tURL completa: " + url.toString());
        System.out.println("\tgetProtocol(): "+ url.getProtocol());
        System. out.println ("\tgetHost (): "+ url.getHost () ) ;
        System.out.println("\tgetPort(): " + url.getPort());
        System.out.println("\tgetFile(): " + url.getFile());
        System.out.println("\tgetUserInfo(): " + url.getUserInfo());
        System.out.println("\tgetPath(): " + url.getPath());
        System. out. println ("\tgetAuthority (): " + url.getAuthority () ) ;
        System.out.println("\tgetQuery(): " + url.getQuery());
        System.out.println("=================================================");
    }//
}// Ejemplo1URL
```

La salida generada será algo así:

```
Y:\Java\UD3>javac Ejemplo1URL.java

Y:\Java\UD3>java Ejemplo1URL
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
=================================================
Otro constructor simple para una URL:
        URL completa: http://www.circuitodejerez.com/index.php?id=196
        getProtocol(): http
        getHost (): www.circuitodejerez.com
        getPort(): -1
        getFile(): /index.php?id=196
        getUserInfo(): null
        getPath(): /index.php
        getAuthority (): www.circuitodejerez.com
        getQuery(): id=196
=================================================
Const. para protocolo +URL +directorio:
        URL completa: http://docs.oracle.com/javase/7
        getProtocol(): http
        getHost (): docs.oracle.com
        getPort(): -1
        getFile(): /javase/7
        getUserInfo(): null
        getPath(): /javase/7
        getAuthority (): docs.oracle.com
        getQuery(): null
=================================================
Constructor para protocolo + URL + puerto +     directorio:
        URL completa: http://docs.oracle.com:80/javase/7
        getProtocol(): http
        getHost (): docs.oracle.com
        getPort(): 80
        getFile(): /javase/7
        getUserInfo(): null
        getPath(): /javase/7
        getAuthority (): docs.oracle.com:80
        getQuery(): null
=================================================
Constructor para un objeto URL y un directorio:
        URL completa: http://docs.oracle.com/javase/7/docs/api/java/net/URL.html

        getProtocol(): http
        getHost (): docs.oracle.com
        getPort(): -1
        getFile(): /javase/7/docs/api/java/net/URL.html
        getUserInfo(): null
        getPath(): /javase/7/docs/api/java/net/URL.html
        getAuthority (): docs.oracle.com
        getQuery(): null
=================================================
```

El siguiente ejemplo crea un objeto URL a la dirección http://www.iessanclemente.net, abre una conexión con él creando un objeto InputStream y lo utiliza 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:

```java
import java.net.*;
import java.io.*;
public class Ejemplo2URL{
    public static void main(String[] args){
        URL url=null;
        try {
            url = new URL("http://www.iessanclemente.net");
        } catch (MalformedURLException e) { e.printStackTrace();}
        BufferedReader in;
        try {
            InputStream inputStream = url.openStream() ;
            in = new BufferedReader(new InputStreamReader(inputStream));
            String inputLine;
            while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);
            in.close() ;
        }catch (IOException e) {e.printStackTrace();}
    }//
}//Ejemplo2URL
```

3.3. 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.iessanclemente.net");

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:

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 establecido a 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

MÉTODOS	MISIÓN
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.

El siguiente ejemplo crea un objeto URL a la dirección http://www.iessanclemente.net, se invoca al método **openConnection()** del objeto para crear una conexión y se obtiene un **URLConnection**. Después se abre un stream de entrada sobre esa conexión mediante el método **getInputStream()**. Al ejecutar el programa se muestra la misma salida que en el ejemplo anterior; sin embargo, este programa crea una conexión con la URL y el anterior abre directamente un stream desde la URL:

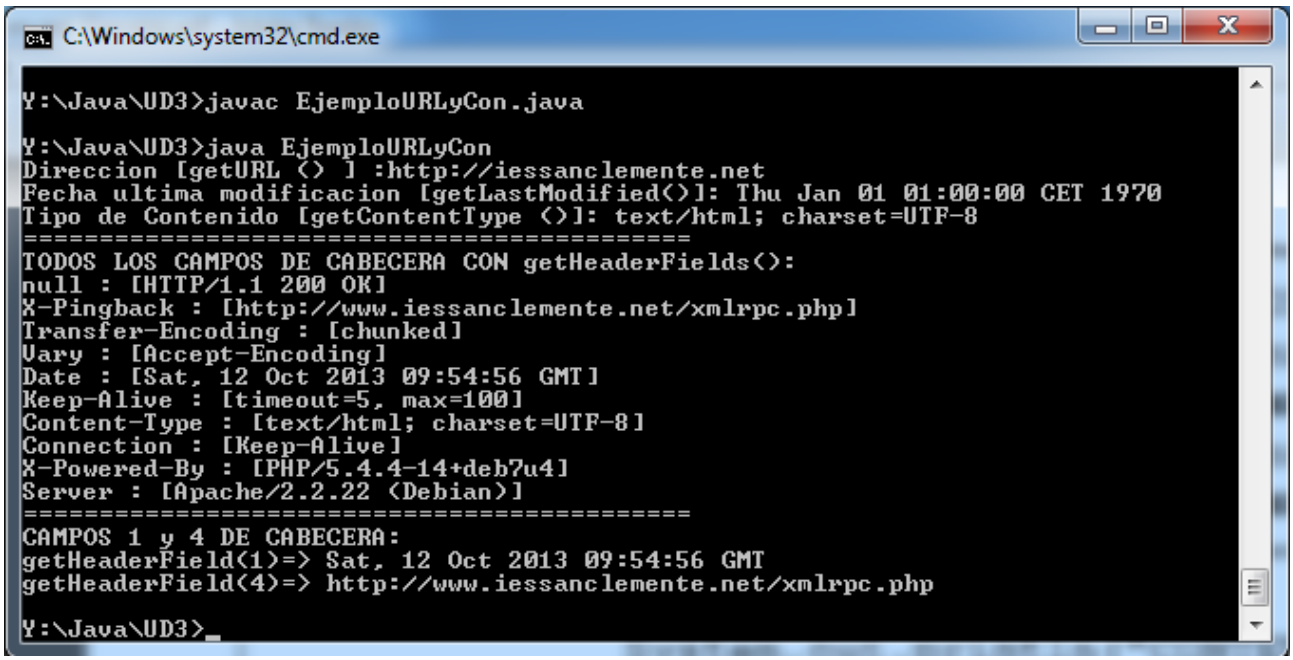
```
import java.net.*;
import java.io.*;

public class EjemplolurlCon {
    public static void main(String[] args) {
        URL url=null;
        URLConnection urlCon=null;
        try {
            url = new URL("http://www.iessanclemente.net");
            urlCon= url.openConnection() ;

            BufferedReader in;
            InputStream inputStream = urlCon.getInputStream() ;
            in = new BufferedReader(new
            InputStreamReader(inputStream)) ;

            String inputLine;
            while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);
            in.close () ;
        }
        catch (MalformedURLException e) {e.printStackTrace();}
        catch (IOException e) {e.printStackTrace();}
    }//
}//EjemplolurlCon
```

En el siguiente ejemplo se prueban algunos de los métodos de la clase URLConnection:

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

public class EjemploURLyCon {
    @SuppressWarnings("rawtypes") //deshabilitar las advertencias de compilación
    //en relación con los tipos de prima (clase genérica o interfaz sin ningún
    //argumento de tipo), ocurre en el Map<String,List<String>>

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

        URL url = new URL("http://iessanclemente.net");
        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)) ;

    }
}//
```

NOTA: Para recorrer una estructura **Map** podemos usar una estructura **Iterator**. Para obtener un iterador sobre el map se invoca a los métodos *entrySet()* e *iterator()*. 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.

La salida será la siguiente:


C:\Windows\system32\cmd.exe
Y:\Java\UD3>javac EjemploURLyCon.java
Y:\Java\UD3>java EjemploURLyCon
Direccion [getURL ()] :http://iessanclemente.net
Fecha ultima modificacion [getLastModified()]: Thu Jan 01 01:00:00 CET 1970
Tipo de Contenido [getContentType ()]: text/html; charset=UTF-8
==========
TODOS LOS CAMPOS DE CABECERA CON getHeaderFields():
null : [HTTP/1.1 200 OK]
X-Pingback : [http://www.iessanclemente.net/xmlrpc.php]
Transfer-Encoding : [chunked]
Vary : [Accept-Encoding]
Date : [Sat, 12 Oct 2013 09:54:56 GMT]
Keep-Alive : [timeout=5, max=100]
Content-Type : [text/html; charset=UTF-8]
Connection : [Keep-Alive]
X-Powered-By : [PHP/5.4.4-14+deb7u4]
Server : [Apache/2.2.22 (Debian)]
==========
CAMPOS 1 y 4 DE CABECERA:
getHeaderField(1)=> Sat, 12 Oct 2013 09:54:56 GMT
getHeaderField(4)=> http://www.iessanclemente.net/xmlrpc.php
Y:\Java\UD3>