Programación de comunicaciones en red parte I

1. Conceptos básicos

1.1 Protocolo de Comunicaciones

Para que dos o más ordenadores puedan conectarse a través de una red y ser capaces de intercambiar datos de una forma ordenada, deben seguir un protocolo de comunicaciones que sea aceptado por todos ellos. El protocolo define las reglas que se deben seguir en la comunicación, por ejemplo, enseñar a los niños a decir *por favor* y *gracias* es una forma de indicarles un protocolo de educación, y si alguna vez se olvidan de dar las gracias por algo, seguro que reciben una reprimenda de sus mayores.

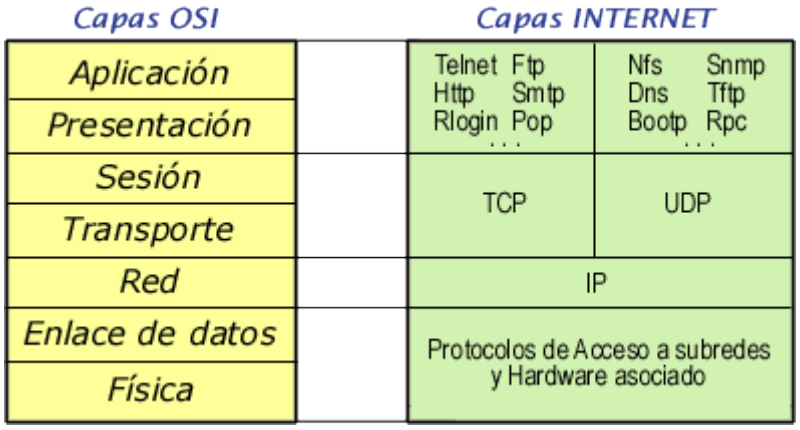
Hay muchos protocolos disponibles para ser utilizados; por ejemplo, el protocolo **HTTP** define como se van a comunicar los servidores y navegadores Web y el protocolo **SMTP** define la forma de transferencia del correo electrónico. Estos protocolos, son protocolos de aplicación que actúan al nivel de superficie, pero también hay otros protocolos de bajo nivel que actúan por debajo del nivel de aplicación y que son más complicados, aunque, afortunadamente, como programadores Java, no será necesario tener excesivo conocimiento de los protocolos de bajo nivel; nada que vaya más allá del conocimiento de su existencia.

1.2 Capas de Red

Las redes están separadas lógicamente en *capas* o *niveles*; desde el nivel de aplicación en la parte más alta hasta el *nivel físico* en la parte más baja. Los detalles técnicos de la división en capas o niveles de la red no es un conocimiento imprescindible para desarrollar programas Java que se comuniquen a través de la red, ya que la gente de JavaSoft se ha encargado de ocultar toda la parafernalia que involucra el manejo de los protocolos de redes de bajo nivel y las capas de más bajo nivel del modelo de comunicaciones.

La única capa interesante para el usuario y el programador es el ***Nivel de Aplicación***, que es el que se encarga de tomar los datos en una máquina desde esta capa y soltarlos en la otra máquina en esta misma capa, los pasos intermedios y los saltos de capas que se hayan producido por el camino, no resultan de interés, ya que su uso está oculto en el lenguaje Java.

A continuación se explica un poco el funcionamiento de las capas, la figura más abajo muestra la correlación existente entre el modelo teórico de capas o niveles de red propuestos por la Organización de Estándares Internacional (ISO, *International Standards Organization*) y el modelo empleado por las redes TCP/IP. Cuando se presenta un problema de tamaño considerable, la solución más óptima comienza por dividirlo en pequeñas secciones, para posteriormente proceder a solventar cada una de ellas independientemente. Pues el mismo principio de *divide y vencerás* es el que se sigue a la hora de diseñar redes, es decir, separar en un buen número de niveles el hecho de la transmisión de un sistema a otro. Como referencia, la ISO, creó un modelo de interconexión de sistemas abiertos, conocido como ***OSI***. Ese modelo divide en siete capas el proceso de transmisión de información entre equipos informáticos, desde el hardware físico, hasta las aplicaciones de red que maneja el usuario. Estas capas son las que se pueden ver en la figura siguiente: física, de enlace de datos, de red, de transporte, de sesión, de presentación y, por último, de aplicación. Cada nuevo protocolo de red que se define se suele asociar a uno (o a varios) niveles del estándar OSI. Internet dispone de un modelo más sencillo; no define nada en cuanto al aspecto físico de los enlaces, o a la topología o clase de red de sus subredes y, por lo tanto, dentro del modelo OSI, sólo existe una correlación con los niveles superiores.



Las aplicaciones que trabajan a un cierto nivel o capa, sólo se comunican con sus iguales en los sistemas remotos; es decir, a nivel de aplicación, un navegador sólo se entiende con un servidor Web, sin importarle para nada cómo le llega la información. Este mismo principio es el que se emplea para el resto de las capas. Para ilustrar este concepto de capas o niveles, puede resultar explicativo ver qué sucede cuando se solicita una página Web. En este caso, el navegador realiza una *petición HTTP*, petición que se incluye en un *paquete TCP*, que a su vez es encapsulado y fragmentado en uno o varios *datagramas IP*, que es la unidad de datos a nivel de red. Dichos datagramas son de nuevo encapsulados en unidades de datos *PPP*, o *frames*, que se envían al proveedor de Internet a través del módem, que transforma esas unidades digitales de datos en señales acústicas de acuerdo a una determinada norma, *V.34bis* o *V.90*, por ejemplo. El proveedor de Internet ensamblará los paquetes PPP para convertirlos de nuevo en datagramas IP, que son llevados a su destino, donde serán decodificados en sentido inverso al realizado en el equipo originador de la petición, hasta que alcancen *el nivel de aplicación*, que supone el servidor web.

De todo esto, se pueden sacar tres ideas fundamentales. En primer lugar, que TCP/IP opera sólo en los niveles superiores de red, resultándole indiferente el conjunto de protocolos que se entienden con los adaptadores de red Token Ring, Ethernet, ATM, etc., que se encuentren por debajo. En segundo lugar, que IP es un protocolo de datagramas que proporciona una interfaz estándar a protocolos superiores. Y, en tercer lugar, que dentro de estos protocolos superiores se incluyen *TCP* y *UDP*, los cuales ofrecen prestaciones adicionales que ciertas aplicaciones de red necesitan.

1.3 Cliente y Servidor

Servidor es aquel que escucha y está siempre a la espera de que el *Cliente* se conecte y comenzar la conversación entre ambos.

1.4 IP, Internet Protocol

Es el protocolo que se utiliza por debajo del *Nivel de Aplicación* para traspasar datos entre cliente y servidor. Sólo se necesita saber que es un protocolo de red encargado de mover datos en forma de paquetes entre un origen y un destino y que, como bien indica su nombre, es el protocolo que normalmente se utiliza en Internet.

IP es un protocolo simple, fácilmente implementable, de pequeñas unidades de datos o datagramas, que proporciona un interfaz estándar a partir del cual el resto de los protocolos y servicios pueden ser construidos, sin tener que preocuparse de las diferencias que existan entre las distintas subredes por la cuales circulen los datos.

Todo dispositivo conectado a Internet o a cualquier red basada en TCP/IP, posee al menos una dirección IP, un identificador que define unívocamente al dispositivo que lo tiene asignado en la red.

Un **datagrama IP** se encuentra dividido en dos partes: cabecera y datos. Dentro de la cabecera se encuentran, entre otros campos, la dirección IP del equipo origen y la del destino, el tamaño y un número de orden. El estudio más en detalle de la estructura interna del datagrama, se escapa del alcance de este tema.

IP opera entre un sistema local conectado a Internet y su *router* más próximo, así como entre los distintos routers que forman la red. Cuando un datagrama llega a un router, éste determina, a partir de su dirección IP de destino, hacia cuál de sus conexiones de salida ha de dirigir el datagrama que acaba de recibir. Por desgracia, en cuanto al transporte, IP provee un servicio que *intenta* entregar los datos al equipo destino, pero no puede garantizar la integridad, e incluso la recepción de esos datos. Por ello, la mayoría de las aplicaciones hacen uso de un protocolo de más alto nivel que ofrezca el grado de fiabilidad necesario.

Cada datagrama IP es independiente del resto, por lo que cada uno de ellos es llevado a su destino por separado. La longitud del datagrama es variable, pudiendo almacenar hasta 65 Kbytes de datos; si el paquete de datos (TCP o UDP) sobrepasa ese límite, o el tamaño de la unidad de datos de la red que se encuentra por debajo es más pequeño que el datagrama IP, el mismo protocolo IP lo fragmenta, asignándole un número de orden, y distribuye empleando el número de datagramas que sea necesario.

1.5 TCP, Transmission Control Protocol (Orientado a la conexión)

Hay veces en que resulta de vital importancia tener la seguridad de que todos los paquetes que constituyen un mensaje llegan a su destino y en el orden correcto para la recomposición del mensaje original por parte del destinatario. El protocolo ***TCP*** se incorporó al protocolo **IP** para proporcionar a éste la posibilidad de dar reconocimiento de la recepción de paquetes y poder pedir la retransmisión de los paquetes que hubiesen llegado mal o se hubiesen perdido. Además, TCP hace posible que todos los paquetes lleguen al destinatario, juntos y en el mismo orden en que fueron enviados.

Por lo tanto, es habitual la utilización de los dos acrónimos juntos, ***TCP/IP***, ya que los dos protocolos constituyen un método más fiable de encapsular un mensaje en paquetes, de enviar los paquetes a un destinatario, y de reconstruir el mensaje original a partir de los paquetes recibidos.

TCP, en resumen, ofrece un servicio de transporte de datos fiable, que garantiza la integridad y entrega de los datos entre dos procesos o aplicaciones de máquinas remotas. Es un protocolo orientado a la conexión, es decir, funciona más o menos como una llamada de teléfono. En primer lugar, el equipo local solicita al remoto el establecimiento de un canal de comunicación; y solamente cuando ese canal ha sido creado, y ambas máquinas están preparadas para la transmisión, empieza la transferencia de datos real.

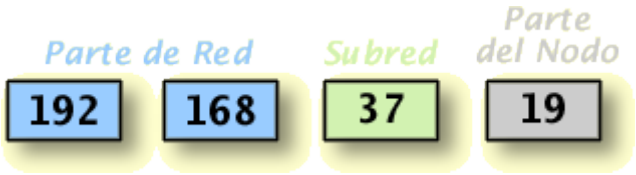
1.6 UDP, User Datagram Protocol (no orientado a la conexión)

Hay veces en que no resulta tan importante el que lleguen todos los mensajes a un destinatario, o que no lleguen en el orden en que se han enviado. Hay ocasiones en las que no se quiere incurrir

en una sobrecarga del sistema o en la introducción de retrasos por causa de cumplir esas garantías. Por ejemplo, si un ordenador está enviando la fecha y la hora a otro ordenador cada 100 milisegundos para que la presente en un reloj digital, es preferible que cada paquete llegue lo más rápidamente posible, incluso aunque ello signifique la pérdida de algunos de los paquetes. El protocolo ***UDP*** está diseñado para soportar este tipo de operaciones. UDP es, por tanto, un protocolo menos fiable que el TCP, ya que no garantiza que una serie de paquetes lleguen en el orden correcto, e incluso no garantiza que todos esos paquetes lleguen a su destino. Los procesos que hagan uso de UDP han de implementar, si es necesario, sus propias rutinas de verificación de envío y sincronización. Como programador Java, está en tus manos la elección del protocolo que va a utilizar un programa en sus comunicaciones, en función de las características de velocidad y seguridad que requiera la comunicación que se desea establecer.

1.7 Dirección IP

La verdad es que no se necesita saber mucho sobre el protocolo IP para poder utilizarlo, pero sí que es necesario conocer el esquema de direccionamiento que utiliza este protocolo. Cada ordenador conectado a una red TCP/IP dispone de una dirección IP única de 4 bytes (32 bits), en donde, según la clase de red que se tenga y la máscara, parte de los 4 bytes representan a la red, parte a la subred (donde proceda) y parte al dispositivo final o nodo específico de la red. La figura siguiente muestra la representación de los distintos números de una dirección IP de un nodo perteneciente a una subred de clase B (máscara 255.255.0.0). Con 32 bits se puede definir una gran cantidad de direcciones únicas, pero la forma en que se asignaban estas direcciones estaba un poco descontrolada, por lo que hay muchas de esas direcciones que a pesar de estar asignadas no se están utilizando.



Por razones administrativas, en los primeros tiempos del desarrollo del protocolo IP, se establecieron cinco rangos de direcciones, dentro del rango total de 32 bits de direcciones IP disponibles, denominando a esos subrangos, *clases*. Cuando una determinada organización requiere conectarse a Internet, solicita una clase, de acuerdo al número de nodos que precise tener conectados a la Red. La administración referente a la cesión de rangos la efectúa InterNIC (*Internet Network Information Center*), aunque existen autoridades que, según las zonas, gestionan dominios locales; por ejemplo, el dominio correspondiente a España lo gestiona *Red Iris*.

Los subrangos se definen en orden ascendente de direcciones IP, por lo cual, a partir de una dirección IP es fácil averiguar el tipo de clase de Internet con la que se ha conectado. El tipo de clase bajo la que se encuentra una dirección IP concreta viene determinado por el valor del primer byte de los cuatro que la componen o, lo que es igual, el primer número que aparece en la dirección IP. Las clases toman nombre de la A a la E, aunque las más conocidas son las A, B y C. En Internet, las redes de clase A son las comienzan con un número entre el 1 y el 126, que permiten otorgar el mayor número de direcciones IP (16,7 millones), por lo que se asignan a grandes instituciones educativas o gubernamentales. Las clases B (65536 direcciones por clase), suelen concederse a grandes empresas o corporaciones y, en general, a cualquier organización que precise un importante número de nodos. Las redes de clase C (256 direcciones) son las más comunes y habitualmente se asignan sin demasiados problemas a cualquier empresa u organización que lo solicite. La clase D se reserva a la transmisión de mensajes de difusión múltiple (multicast), mientras que la clase E es la destinada a investigación y desarrollo. La tabla siguiente resume estos datos.

Clase	Nodos por Clase	Máscara asociada	Direccion de comienzo	Dirección Final
A	$2^{24} = 16,777,216$	255.0.0.0	0.0.0.0	127.255.255.255
B	$2^{16} = 65,536$	255.255.0.0	128.0.0.0	191.255.255.255
C	$2^{8} = 256$	255.255.255.0	192.0.0.0	223.255.255.255
D	-	-	224.0.0.0	239.255.255.255
E	-	-	240.0.0.0	255.255.255.255

Todo lo dicho antes solamente implica a la asignación de direcciones dentro de Internet. Si se diseña una red TCP/IP que no vaya a estar conectada a la Red, se puede hacer uso de cualquier conjunto de direcciones IP. Solamente existen cuatro limitaciones, intrínsecas al protocolo, a la hora de escoger direcciones IP, pero que reducen en cierta medida el número de nodos disponibles por clase que se indicaban en la tabla anterior.

1. No se pueden asignar direcciones que comiencen por 0; dichas direcciones hacen referencia a nodos dentro de la red actual.
2. La red 127 se reserva para los procesos de resolución de problemas y diagnosis de la red; de especial interés resulta la dirección 127.0.0.1, bucle interno (*loopback*) de la estación de trabajo local.
3. Las direcciones IP de nodos no pueden terminar en 0, o en cualquier otro valor base del rango de una subred; porque es así como concluyen las redes.
4. Cuando se asignan direcciones a nodos, no se pueden emplear el valor 255, o cualquier otro valor final del rango de una subred. Este valor se utiliza para enviar mensajes a todos los elementos de una red (broadcast); por ejemplo, si se envía un mensaje a la dirección 192.168.37.255, se estaría enviando en realidad a todos los nodos de la red de clase C 192.168.37.xx.

Ahora bien, si se quiere que una red local tenga acceso exterior, hay una serie de restricciones adicionales, por lo que hay una serie de direcciones reservadas que, a fin de que pudiesen ser usadas en la confección de redes locales, fueron excluidas de Internet. Estas direcciones se muestran en la siguiente tabla.

Clase	Máscara asociada	Dirección de comienzo	Dirección Final
A	255.0.0.0	10.0.0.0	10.255.255.255
B	255.255.0.0	172.16.0.0	172.31.255.255
C	255.255.255.0	192.168.0.0	192.168.255.255

Infovía, por ejemplo, al ser una especie de gran Intranet española, utiliza el rango de direcciones 10.xx.xx.xx. Si estás considerando la creación de una intranet, deberías escoger direcciones IP para tu red dentro de alguno de los rangos reservados de la tabla anterior, y emplear un servidor proxy, o cualquier otro mecanismo que enmascare las direcciones IP de esa intranet, de forma que todos los puestos de la red local utilicen una única dirección IP a la hora de salir a la Red.

Actualmente, se intenta expandir el número de direcciones únicas a un número mucho mayor, utilizando 128 bits. *E.R. Harold*, en su libro *Java Network Programming*, dice que el número de direcciones únicas que se podría alcanzar representando las direcciones con 128 bits es 1.6043703E32. La verdad es que las direcciones indicadas de esta forma son difíciles de recordar, así que lo que se hace es convertir el valor de los cuatro bytes en un número decimal y separarlos por puntos, de forma que sea mucho más sencillo el recordarlos; así, por ejemplo, la dirección única asignada a **java.sun.com** es **137.254.16.112**.

1.8 Dominios

Y ahora surge la pregunta de qué es lo que significa **java.sun.com**. Como a pesar de que la dirección única asignada a un ordenador se indique con cuatro cifras pequeñas, resulta muy difícil recordar las direcciones de varias máquinas a la vez; muchas de estas direcciones se han hecho

corresponder con un nombre, o *dominio*, constituido por una cadena de caracteres, que es mucho más fácil de recordar para los humanos. Así, el dominio para la dirección IP **137.254.16.112** es **java.sun.com**.

El Sistema de Nombres de Dominio (***DNS***, Domain Name System) fue desarrollado para realizar la conversión entre los dominios y las direcciones IP. De este modo, cuando el lector entra en Internet a través de su navegador e intenta conectarse con un dominio determinado, el navegador se comunica en primer lugar con un servidor DNS para conocer la dirección IP numérica correspondiente a ese dominio. Esta dirección numérica IP, y no el nombre del dominio, es la que va encapsulada en los paquetes y es la que utiliza el protocolo Internet para enrutar estos paquetes desde el ordenador del lector hasta su destino.

Java proporciona clases para la manipulación y conocimiento de direcciones y dominios, concretamente la clase **InetAddress** permite encontrar un nombre de dominio a partir de su dirección IP; y viceversa, encontrar la dirección IP que corresponde a un dominio determinado.

Si deseas saber cuál es tu dirección IP, pues lo cierto es que probablemente no lo sepas. Si utilizas un proveedor de acceso a Internet, realmente no tendrás una dirección IP fija o un dominio específico. Cada proveedor de acceso a Internet dispone de un bloque de direcciones reservadas; cuando se conecta con él y se accede a Internet, el proveedor asigna una dirección de ese bloque, que durará solamente el tiempo que dure la conexión. Si desconectas y vuelves a conectar, casi seguro que la dirección IP para esta nueva sesión será diferente de la utilizada en la anterior conexión.

1.9 Puertos y Servicios

Un *servicio* es una facilidad que proporciona el sistema, y cada uno de estos servicios está asociado a un *puerto*. Un puerto es una dirección numérica a través de la cual se procesa el servicio, es decir, no son puertos físicos semejantes al puerto paralelo para conectar la impresora en la parte trasera del ordenador, sino que son direcciones lógicas proporcionadas por el sistema operativo para poder responder.

Los puertos asignados a las aplicaciones más conocidas son los siguientes:

Número de puerto	Aplicación servidora
21	FTP (transferencia de ficheros)
23	Telnet (conexión a terminal remoto)
25	SMTP (correo)
80	HTTP (web)
119	NNTP (news)

Las comunicaciones de información relacionada con Web tienen lugar a través del puerto 80 mediante protocolo TCP. Para emular esto en Java, se utiliza la clase **Socket**.

Teóricamente hay 65535 puertos disponibles, aunque los puertos del 1 al 1023 están reservados al uso de servicios estándar proporcionados por el sistema, quedando el resto libre para utilización por las aplicaciones de usuario. De no existir los puertos, solamente se podría ofrecer un servicio por máquina. Nótese que el protocolo IP no sabe nada al respecto de los números de puerto, al igual que TCP y UDP no se preocupan en absoluto por las direcciones IP. Se puede decir que IP pone en contacto las máquinas, TCP y UDP establecen un canal de comunicación entre

determinados procesos que se ejecutan en tales equipos y, los números de puerto se pueden entender como números de oficinas dentro de un gran edificio. El edificio (equipo), tendrá una única dirección IP, pero dentro de él, cada tipo de negocio, en este caso HTTP, FTP, etc., dispone de una oficina individual.

Los puertos son de gran importancia en las aplicaciones cliente/servidor, constituyen la arquitectura más común de las aplicaciones o servicios de Internet.

Es importante permitir o denegar el acceso a los diferentes puertos porque las aplicaciones servidoras "escuchan" en un puerto conocido de antemano para que un cliente pueda conectarse. Esto significa que cuando el sistema operativo recibe una petición a ese puerto, éste la traslada a la aplicación que escucha en él.

1.10 Cortafuegos

Seguramente ya sabrás algo sobre los cortafuegos, o ***firewalls***. Un firewall es el nombre que se da a un equipo y su software asociado que permite aislar la red interna de una empresa del resto de Internet. Normalmente se utiliza para restringir el grado de acceso de los ordenadores de la red interna de una empresa a Internet, por razones de seguridad o cualquier otra.

1.11 Servidores Proxy

También es posible que hayas leído cosas sobre los ***servidores proxy***. Un servidor proxy actúa como interfaz entre los ordenadores de la red interna de una empresa e Internet. Frecuentemente, el servidor proxy tiene posibilidad de ir almacenando un cierto número de páginas web temporalmente en caché, para un acceso más rápido. Por ejemplo, si diez personas dentro de la empresa intentan conectarse a un mismo servidor Internet y descargar la misma página en un período corto de tiempo, esa página puede ser almacenada por el servidor proxy la primera vez que se accede a ella y proporcionarla él, sin necesidad de acceder a Internet, a las otras nueve personas que la han solicitado. Esto reduce en gran medida el tiempo de espera por la descarga de la página y el tráfico, tanto dentro como fuera de la empresa, aunque a veces puede también hacer que la información de la página se quede sin actualizar, al no descargarse de su sitio original.

1.12 URL, Uniform Resource Locator

Una ***URL***, o dirección, es en realidad un puntero a un determinado recurso de un determinado sitio de Internet. Al especificar una URL, se está indicando:

- El protocolo utilizado para acceder al servidor (http, por ejemplo)
- El nombre del servidor
- El puerto de conexión (opcional)
- El camino, y
- El nombre de un fichero determinado en el servidor (opcional a veces)
- Un punto de referencia dentro del fichero (opcional)

A veces el nombre del fichero se puede omitir, ya que el navegador incorporará automáticamente el nombre de fichero ***index.html*** cuando no se indique ninguno, e intentará descargar ese fichero.

Además de indicar el fichero o página a la que se desea acceder, también es posible indicar una referencia (*anchor*), que se haya establecido dentro de esa página.

La sintaxis general, resumiendo pues, para una dirección URL, sería:

```
protocolo://nombre_servidor[:puerto]/directorio/fichero#referencia
```

El *puerto* es opcional y normalmente no es necesario especificarlo si se está accediendo a un servidor que proporcione sus servicios a través de los puertos estándar; tanto el navegador como cualquier otra herramienta que se utilice en la conexión conocen perfectamente los puertos por los cuales se proporciona cada uno de los servicios e intentan conectarse directamente a ellos por defecto.

2. El paquete *java.net*

Las aplicaciones utilizan los servicios de red para comunicarse, pero no conocen cómo deben mandar y recibir la información. Desde el punto de vista de una aplicación, es como si estuviera leyendo y escribiendo sobre un fichero o la salida/entrada estándar. Java dispone del paquete *java.net*, que contiene definiciones de clases e interfaces que implementan estos servicios de red.

Interfaces	Clases	Excepciones
ContentHandlerFactory	Authenticator	BindException
DatagramSocketImplFactory	ContentHandler	ConnectException
FileNameMap	DatagramPacket	MalformedURLException
SocketImplFactory	DatagramSocket	NoRouteToHostException
SocketOptions	DatagramSocketImpl	PortUnreachableException
URLStreamHandlerFactory	HttpURLConnection	ProtocolException
	Inet4Address	SocketException
	Inet6Address	SocketTimeoutException
	InetAddress	UnknownHostException
	InetSocketAddress	UnknownServiceException
	JarURLConnection	URISyntaxException
	MulticastSocket	
	NetPermission	
	NetworkInterface	
	PasswordAuthentication	
	ServerSocket	
	Socket	
	SocketAddress	
	SocketImpl	
	SocketPermission	
	URI	
	URL	
	URLClassLoader	
	URLConnection	
	URLDecoder	
	URLEncoder	
	URLStreamHandler	

2.1 Dirección. Clase *InetAdress*

Java trabaja en Internet; para poder comunicarse con procesos que se encuentran en distintos ordenadores es necesario que cada uno de ellos disponga de una identificación única dentro de la red. Esta identificación se contempla en la norma IP (*Internet Protocol*), estudiada anteriormente.

En Java manejaremos direcciones IP mediante la clase *InetAddress*. Para esta clase no existe un constructor, aunque sí posee métodos estáticos para iniciar un objeto *InetAddress*, como por ejemplo *getByName*().

Ejemplo 1

```
import java.net.*;
public class Direccion{
     public static void main(String args[]){
            if(args.length<1){
                  System.out.println("Indica el nombre del ordenador");
                  return;
            }
            try{
                  InetAdress dirección = InetAddress.getByName(args[0]);
                  System.out.println(direccion);
            }catch(UnknownHostException e){
                  System.out.println("Imposible encontrar la dirección:   "
                  + args[0]);
            }
     }
}
-------------------------------------
```

El resultado de ejecutar este código es:

```
>java Direccion www.google.es
www.google.es/66.249.87.104
```

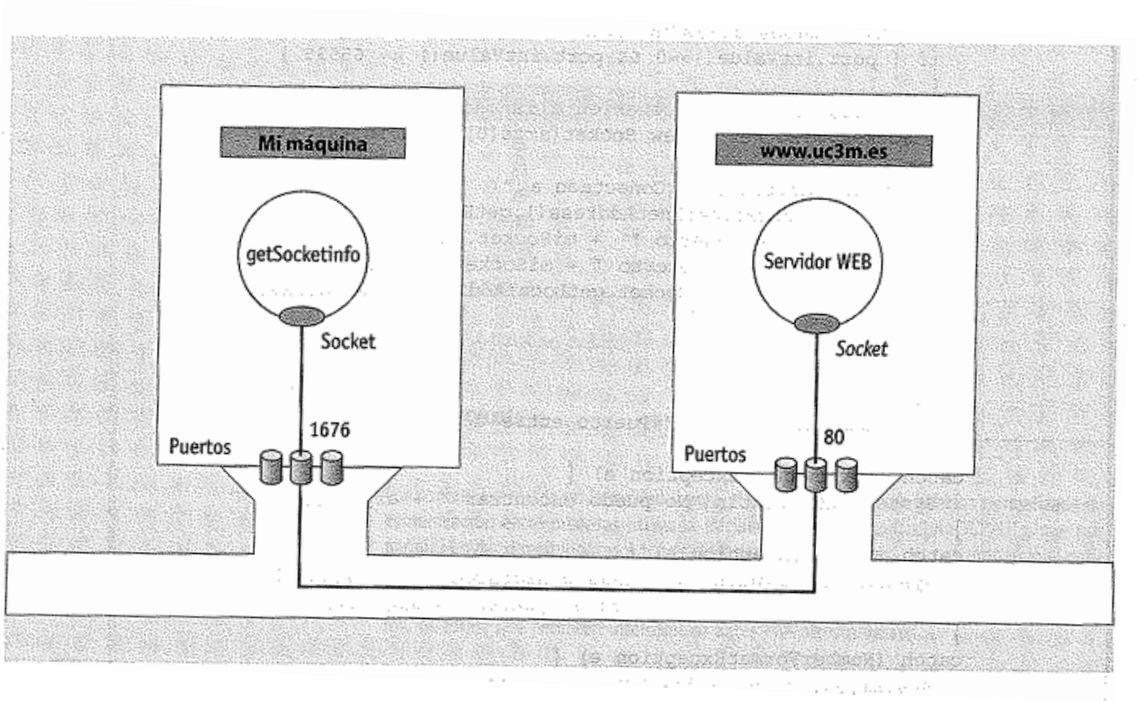
2.2 Sockets

Java proporciona tres formas diferentes de atacar la programación de comunicaciones a través de red, al menos en lo que a la comunicación web concierne. Por un lado están las clases *Socket*, *DatagramSocket*, *MulticastSocket* y *ServerSocket*, y por otro lado están las clases *URL*, *URLEncoder* y *URLConnection*.

Los sockets son puntos finales de enlaces de comunicaciones entre procesos. Los procesos los tratan como descriptores de ficheros, de forma que se pueden intercambiar datos con otros procesos transmitiendo y recibiendo a través de sockets. **El tipo de sockets describe la forma en la que se transfiere información a través de ese socket**.

Ejemplo 2

Vamos a programar en Java lo siguiente.



El cliente “Mi máquina” utiliza un socket para conectarse al servidor que se encuentra escuchando, mediante otro Socket, en el puerto 80. El socket del cliente utiliza el puerto 1676 para conectarse con el servidor. Al cliente se le asigna el primer puerto libre (no es necesario que sea uno fijo).

En Java esto sería del siguiente modo:

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

public class GetSocketInfo{

  //al llamar a esta clase hay que pasarle de entrada dirección y puerto
  public static void main(String args[]){
     if(args.length>1){
        Integer puerto;
        try{
           puerto=new Integer(args[1]);
           //Comprobamos si es un número de puerto válido
           if(puerto.intValue()>=0 && puerto.intValue()<=65535){
            //Solicitamos conexión a la máquina
            Socket miSocket = new Socket(args[0],puerto.intValue());

           System.out.println("Conectado a "
            + miSocket.getInetAddress().getHostName()
            + " en el puerto " + miSocket.getPort()
            + " desde el puerto " + miSocket.getLocalPort()
            + " en " +miSocket.getLocalAddress().getHostName());
```

```
          miSocket.close();
          }else{
           System.out.println("#Puerto entre 0 y 65535");
          }
       }catch(UnknownHostException e){
          System.out.println("No puedo encontrar "+args[0]);
       }catch(SocketException e){
          System.out.println("No puedo conectar a "+args[0]
          + " en el puerto " + args[1]);
       }catch(NumberFormatException e){
          System.out.println("#Puerto valor entero");
       }catch(IOException e){
          System.out.println(e);
       }
    }else{
       System.out.println("Usar la siguiente sintaxis:\n"
       + ">java getSocketInfo <direccion IP> <Puerto>" );
    }
  }
}
```

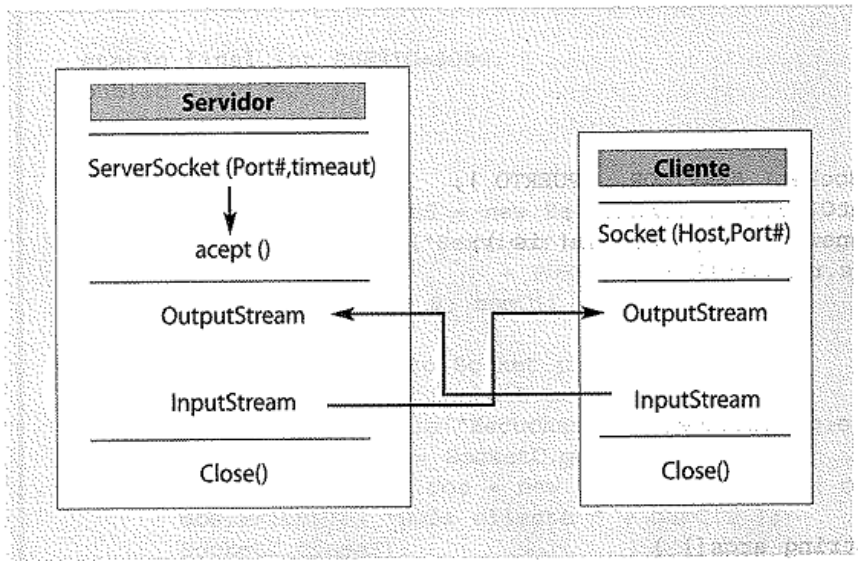
El resultado de ejecutar esto es:

```
>java GetSocketInfo www.iessantiagohernandez.com 80
Conectado a www.iessantiagohernandez.com en el puerto 80 desde el
puerto 49282 en Win7-base.localdomain

>java GetSocketInfo www.iessantiagohernandez.com 1000
No puedo conectar a www.iessantiagohernandez.com en el puerto 1000

>java GetSocketInfo www.iessantiagohernandez.es 80
No puedo encontrar www.iessantiagohernandez.es
```

A continuación se muestra todos los pasos seguidos durante todas las fases del proceso de comunicación cliente/servidor.



En Java, los sockets, para poder llevar a cabo la comunicación entre dos procesos, utilizan un objeto InputStream o OutputStream, según el sentido en que fluye la información. Existen dos tipos de sockets que utilizan los mecanismos de IOStream para comunicar procesos:

- La clase ServerSocket: utilizada por la aplicación servidora para escuchar en un determinado puerto la petición de conexión de los clientes.
- La clase Socket: utilizada por los clientes para iniciar una conexión con el servidor.

2.2.1 Aplicación cliente/servidor con *sockets* (*Socket* y *ServerSocket*)

Veamos una aplicación con una arquitectura cliente/servidor en la que los clientes intercambian con el servidor unas cadenas de información.

Ejemplo 3

Nos hacen falta dos aplicaciones, el servidor y el cliente.

Aplicación cliente

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

public class Cliente{
  static final String SERVIDOR = "localhost";
  static final int PUERTO = 6000;

  //Constructor
  public Cliente(){
     try{
      Socket SocketC = new Socket(SERVIDOR, PUERTO);
      InputStream is = SocketC.getInputStream();
      DataInputStream dis = new DataInputStream(is);
      System.out.println(dis.readUTF());
      SocketC.close();
     }catch(Exception e){
      System.out.println(e.getMessage());
     }
  }

  public static void main(String args[]){
     for (int i = 0; i <= 3; i++)
       new Cliente();
  }
}
```

Esta clase declara un objeto de tipo socket que utilizará para la comunicación con el servidor. El cliente debe especificar la máquina en la que se encuentra el servidor y el puerto por el que escucha. En este ejemplo, tanto la aplicación cliente como la servidora se encuentran en la misma máquina (localhost).

Una vez creado el socket, lo asocia a un objeto *InputStream* mediante el método *getInputStream()*, ya que la información irá del servidor al cliente. El objeto *InputStream* se asocia a un objeto *DataInputStream* para poder utilizar su método *readUTF()*, que obtendrá la información enviada

por el servidor como si fuera un fichero. Una vez recibida la información, el cliente la imprime en la salida estándar.
Finalmente, el cliente cierra el objeto *socket* mediante su método *close()*.

Aplicación servidor

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

public class Servidor{
  static final int PUERTO = 6000;

  //Constructor
  public Servidor(){
     try{
      ServerSocket SocketS = new ServerSocket(PUERTO);
      System.out.println("Servidor escuchando por el puerto "
      + PUERTO);
     for(int i=0; i<3;i++){
      //Crea el objeto socket cliente
      Socket SocketC = SocketS.accept();
      System.out.println("Escuchando al cliente " + i);
      OutputStream os = SocketC.getOutputStream();
      DataOutputStream dos = new DataOutputStream(os);
      dos.writeUTF("Hola cliente " + i);
      SocketC.close();
     }
     System.out.println("El servidor termina");
     SocketS.close();
     }catch(Exception e){
      System.out.println(e.getMessage());
     }
  }

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

El servidor utiliza un objeto *ServerSocket* para escuchar las peticiones por el puerto 6000. Para atender estas peticiones utiliza el método *accept()*. Este método crea un objeto *socket* cuya referencia guarda en *SocketC*, y es el que se utiliza para realizar la comunicación individual con cada cliente.

Para ejecutar la aplicación, debe ejecutarse primero la aplicación servidor, que se queda bloqueado a la espera de recibir peticiones de clientes.
A continuación se ejecuta una instancia de cada cliente, crearemos cuatro, y como el servidor sólo atiende a 3 peticiones, el cuarto recibirá una excepción al no poder conectarse con el servidor.

El resultado de ejecutar el servidor será:

```
Servidor escuchando por el puerto 6000
Escuchando al cliente 0
Escuchando al cliente 1
Escuchando al cliente 2
```

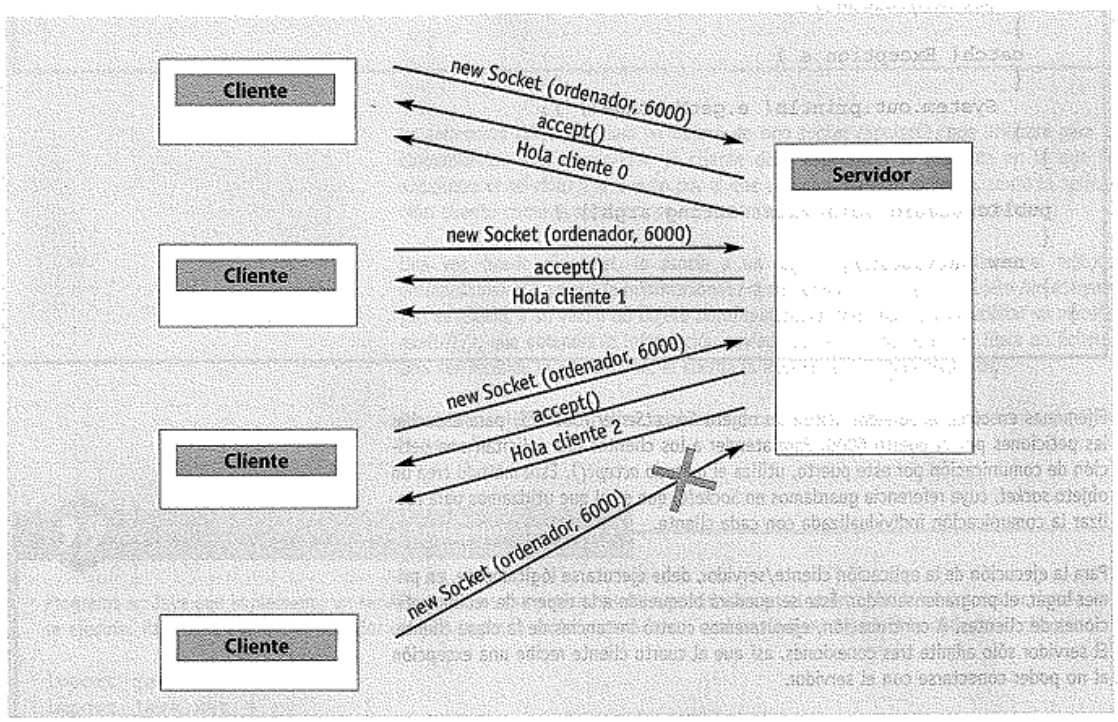
```
El Servidor termina.
```
El resultado de ejecutar los clientes será:

Para cliente 1:
```
Hola cliente 0
```

Para cliente 2:
```
Hola cliente 1
```

Para cliente 3:
```
Hola cliente 2
```

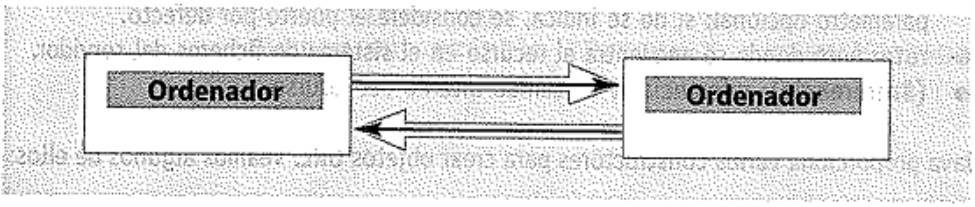
Para cliente 4:
```
Connection refused: connect.
```



2.2.1 Comunicación no orientada a conexión (*DatagramPacket* y *DatagramSocket*)

Antes, no hemos hablado de los tipos de sockets que se pueden utilizar. El tipo de *socket* define un protocolo de transporte distinto y viceversa. En Internet podemos tener dos tipos de comunicación.

- **Orientada a conexión**
  Son comunicaciones en las que es necesario que los procesos establezcan una conexión antes de intercambiar información. Como se ha visto anteriormente la norma de Internet que define este tipo de conexiones es el *Transport Connection Protocol o* TCP, que proporciona un canal de comunicación fiable punto a punto, de modo que si se rompe la conexión o se detecta alguna pérdida de información existen mecanismos para su recuperación y, en su defecto, informar a los procesos implicados de lo ocurrido.
  Java incorpora las clases *Socket* y *SocketServer* para este tipo de conexiones. El ejemplo anterior es una comunicación orientada a conexión.



- **No orientada a conexión**
  En estas comunicaciones no es necesario establecer una conexión. El protocolo que define esta norma es el UDP (User Datagram Protocol). No es necesario establecer una comunicación punto a punto y no existe ningún flujo de control en el intercambio de información. Por lo tanto, en su utilización no está garantizada la fiabilidad. Los datos se envían en paquetes o datagramas cuya entrega no está garantizada. Los paquetes pueden estar duplicados, perdidos o llegar en un orden diferente en el que se enviaron. En Java la comunicación con UDP se implementa mediante dos clases: *DatagramPacket* y *DatagramSocket*.

2.3 La clase URL

El paquete *java.net* ofrece un conjunto de clases para trabajar con las URL.

Java proporciona varios constructores para objetos URL. Veamos algunos de ellos:

- URL(String spec):

```
URL direccion = new URL("http://www.iessantiagohernandez.com");
```

- URL(String protocol, String host, int port, String file):

```
URL dirección = new URL("http","www.iessantiagohernandez.com",
                        "index.html");
```

Todos los constructores de la clase URL pueden provocar la aparición de la excepción *MalformedURLException* debido a que en sus argumentos podemos haber introducido un protocolo desconocido. Por lo tanto, hay que tratar al menos siempre esta excepción, e introducir el constructor de la clase URL en un bloque *try{} catch{}*.

Una vez que creamos una instancia de la clase URL, no la podemos modificar. Sin embargo, la clase URL cuenta con varios métodos que nos suministran información sobre los atributos de la instancia URL.

Ejemplo 4

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

public class URLDemo{
   public static void main(String args[]){
      if(args.length!=1){
         System.out.println("Utilizar : java URLDemo <URL>");
         System.exit(0);
      }
      try{
         URL u = new URL(args[0]);
         System.out.println("protocolo = " +u.getProtocol());
         System.out.println("ordenador = " +u.getHost());
         System.out.println("fichero = " +u.getFile());
         System.out.println("puerto = " +u.getPort());
         System.out.println("ref = " +u.getRef());
      }catch(MalformedURLException e){
         System.out.println("URL errónea: " +args[0]);
      }catch(Exception e){
         System.out.println("Error: "  + e.getMessage());
      }
   }
}

El resultado de ejecutar
> java URLDemo http://www.oracle.com:80/technetwork/es/oem/tutorial-de-oracle-bpm-1704890-esa.pdf#online
debe ser:
protocolo = http
ordenador = www.oracle.com
fichero = /technetwork/es/oem/tutorial-de-oracle-bpm-1704890-esa.pdf
puerto = 80
ref = online
```

Ejemplo 5

Programa en Java que recibe como argumento una URL y escribe en su salida estándar el contenido de dicha URL.

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

public class GetURL{
   public static void main(String args[]){
      if(args.length!=1){
         System.out.println("Utilizar : java getURL <URL>");
         System.exit(0);
      }
      try{
       URL u = new URL(args[0]);
       InputStreamReader isr = new InputStreamReader(u.openStream());
```

```
        BufferedReader in = new BufferedReader(isr);

        String linea;

        while((linea = in.readLine()) != null){
           System.out.println(linea);
        }
        in.close();
       }catch(MalformedURLException e){
         System.out.println("URL errónea: " +args[0]);
       }catch(Exception e){
         System.out.println("Error: " + e.getMessage());
       }
    }
}
```

El programa utiliza el método *openStream()* para crear un objeto *inputStream* con el que poder leer el contenido de la URL.

Al ejecutar el programa, aparece en pantalla el contenido y las etiquetas de la página solicitada, esto es precisamente lo que hace un navegador web, ahora ya sólo faltaría interpretar el contenido de la página y mostrarlo de forma gráfica en la pantalla.

2.4 La clase URLConnection

Hay veces que interesa crear una conexión web y obtener más información sobre ella, para esto se usa la clase URLConnection. Esta clase permite establecer una conexión con un servidor (generalmente web). Cuando se pregunta a un servidor por una URL, además del contenido de esa URL solicitada, también envía información relacionada con el tipo de contenido que envía, la fecha en que envía la información, la fecha de la última modificación de la información,…

El servidor web, inserta esta información al principio del contenido de la URL; por esa razón, a esa información nos referimos como cabecera.

Ejemplo 6

Programa que recibe como argumento una URL y escribe la información de cabecera de esta URL en la salida estándar.

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

public class GetCabecera{
   public static void main(String args[]){
      if(args.length!=1){
         System.out.println("Utilizar : java getCabecera <URL>");
         System.exit(0);
      }
      try{
       URL u = new URL(args[0]);
       URLConnection uc =  u.openConnection();
```

```
        System.out.println("Tipo de contenido [Content-type]: "
         + u.getContentType());
        System.out.println("Codificación [Content-encoding]: "
         + u.getContentEncoding());
        System.out.println("Fecha: "
         + new java.util.Date(uc.getDate()));
        System.out.println("Fecha modificación [Last modified]: "
         + new java.util.Date(uc.getLastModified()));
        System.out.println("Fecha expiración [Expiration date]: "
         + new java.util.Date(uc.getExpiration()));
        System.out.println("Tamaño [Content-length]: "
        + u.getContentLength());

      }catch(MalformedURLException e){
        System.out.println("URL errónea: " +args[0]);
      }catch(Exception e){
        System.out.println("Error: " + e.getMessage());
      }
    }
  }
```

A continuación ejecuta este código para las siguientes URLs:

- http://www.mcgraw-hill.es
- http://www.mcgraw-hill.es/imagenes/general/mhe-logo.gif

Ejemplo 7

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

public class GetObjeto{
  public static void main(String args[]){
    if(args.length!=1){
      System.out.println("Utilizar : java getObjeto <URL>");
      System.exit(0);
    }
    try{
     URL u = new URL(args[0]);
     URLConnection uc =  u.openConnection();

     String ct = uc.getContentType();
     int cl = uc.getContentLength;

     if(ct.starsWith("text/") || cl==-1){
      System.out.println("Esta URL es de tipo texto");
      return;
     }

     InputStream is = new uc.getInputStream();
     BufferedInputStream bin = new BufferedInputStream(is);

     String fich = u.getFile();
     fich = fich.substring(fich.lastIndexOf('/')+1);
```

```
        FileOutPutStream fout = new FileOutPutStream(fich);
        BufferedOutputStream out = new BufferedOutputStream(fout);

        int i;
        while((i = bin.read()) != -1){
            out.write(i);
        }
        out.flush();
        in.close();
        out.close();

       }catch(MalformedURLException e){
         System.out.println("URL errónea: " +args[0]);
       }catch(Exception e){
         System.out.println("Error: " + e.getMessage());
       }
    }
 }
```

En este ejemplo, a partir de una URL se abre una conexión URL llamando al método *openConnection()*. Una vez abierta, comprobamos que el contenido de la URL no es un objeto de tipo texto consultando el método *getContentType()*.

Mediante la llamada *uc.getInputStream()* obtenemos un objeto de *inputStream* que nos permitirá leer el contenido de la URL

Para facilitar las labores de lectura transformamos el objeto *InputStream* en *BufferedInputStream*. Ahora sólo hay que ir leyendo del canal creado en la conexión e ir escribiendo en un fichero.

Ejecuta el ejemplo anterior con la URL que quieras.