PROGRAMACIÓN CONCURRENTE Y DISTRIBUIDA

VI.1: Socket Java

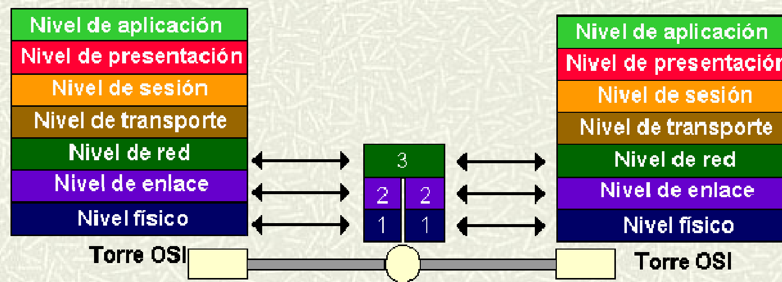
Laura Barros

Notas:

Posibilidades que ofrece Java para la comunicación en red: Socket,RMI y URL.

Modelo OSI (Modelo de Referencia de Interconexión de Sistemas Abiertos)

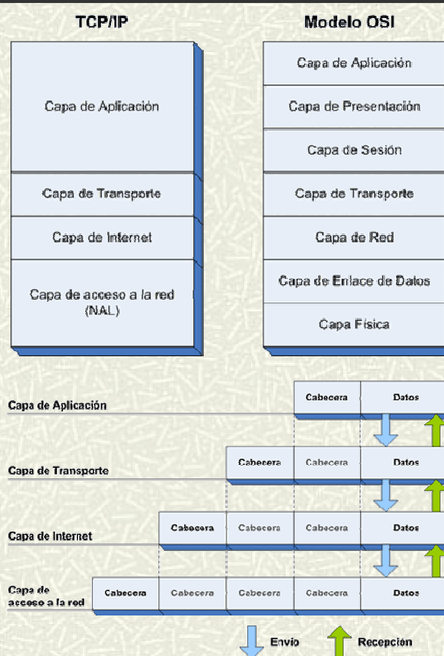
- El molelo OSI está formado por 7 niveles. Cada nivel llama a los servicios del nivel que está justo por debajo. Los niveles paritarios de las dos máquinas que comunican lo hacen virtualmente, a través de los niveles inferiores, sólo el nivel físico comunica realmente con la otra máquina.
- Entre dos niveles vecinos se establece un interfaz para el intercambio de unidades de información conocidas como PDU (unidad de datos de protocolo).



Notas:

Modelo TCP/IP

- Modelo de Internet.
- Agrupa las capas del modelo OSI.
- Es modelo software
  (no define capa física).



- Se añade cabeceras (información de control) a medida que descendemos por la pila.

Notas:

La mayoría de los programadores, no quieren saber cuando programan, de los detalles de bajo nivel de las aplicaciones, necesarios para comunicar un computador con otro.Los programadores prefieren manejar abstracciones de algo nivel que son más fáciles de entender.

El programador no se preocupa por los detalles de la transmisión de “1s y 0s” que se produce entre dos computadores que quieren compartir información. Son los protocolos los que se preocupan por ello. Nos permiten manejar las aplicaciones a nivel de aplicación sin tener que preocuparnos por los detalles de red de bajo nivel. Estos conjuntos de protocolos se llaman pilas . La mas común es la pila TCP/IP. El primero que se estableció fue el **modelo de referencia de Interconexión de Sistemas Abiertos** (OSI, Open System Interconnection) . El modelo en sí mismo no puede ser considerado una arquitectura, ya que no especifica el protocolo que debe ser usado en cada capa, sino que, suele hablarse de modelo de referencia.

Los sockets residen en la capa de sesión del modelo OSI. La capa de sesión se encuentra entre las capas de aplicación y las de comunicación de datos de bajo nivel. La capa de sesión provee servicios de manejo y control del flujo de datos entre dos ordenadores.

Podemos realizar un símil con una llamada telefónica.El teléfono es un interfaz de red y el usuario no requiere conocer los detalles de cómo la voz es transportada. Del mismo modo, un socket actúa como una interfaz de alto nivel que esconde la complejidad de transmitir 1s y 0s a través de los canales desconocidos.

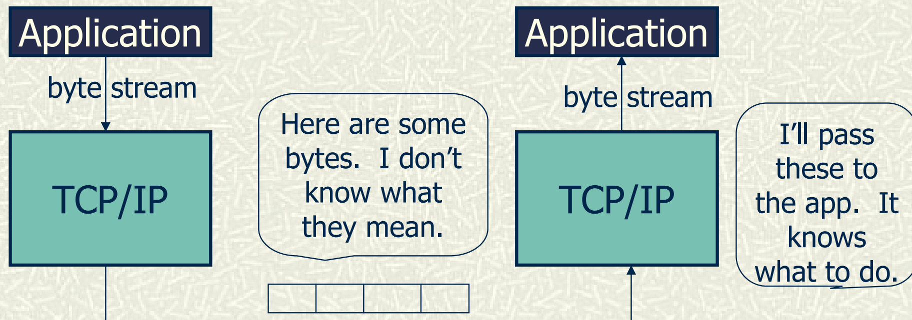
Protocolo de comunicaciones

- La interconexión entre dos o más ordenadores a través de una red es dirigida o supervisada por un protocolo de comunicaciones, el cual, debe ser aceptado por los distintos ordenadores que intervengan en la conexión.
- Define las reglas que se deben seguir en la comunicación.
- Hay muchos protocolos disponibles (ejemplos de protocolos de aplicación):
  - HTTP: define como se van a comunicar los servidores y navegadores web.
  - SMTP: define la forma de transferencia del correo electrónico.
- Hay otros protocolos que actúan por debajo del nivel de aplicación.
- El programador de Java no necesita conocerlos.
- Las redes están separadas lógicamente en capas (layers).
- La comunicación a través de cada capa es establecida por el protocolo correspondiente.

Notas:

Protocolo de comunicaciones II

- El protocolo TCP/IP transporta bytes.
- El protocolo de Aplicación proporciona la semántica.



Notas:

TCP/IP opera sólo en los niveles superiores de red, resultándole indiferente el conjunto de protocolos que se encuentren por debajo de dicha capa.

Aplicaciones Cliente-Servidor I

- Servidor: parte que está escuchando la petición de algún servicio por parte del Cliente.
- Cliente: parte que realiza las peticiones al Servidor.

Cliente:
Inicia la comunicación
Solicita un servicio al servidor
Ejemplo:
-Un cliente web solicita una página
-Un proceso P2P solicita un fichero a otro proceso P2P

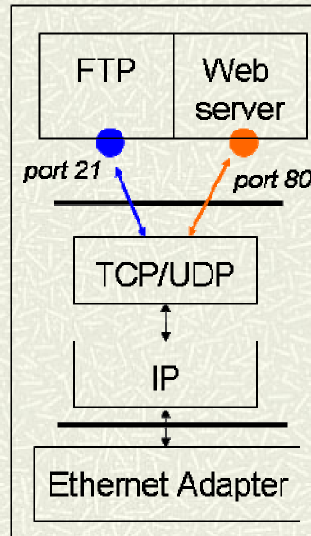
Servidor:
Espera peticiones
Proporciona el servicio solicitado
Ejemplo:
-El servidor web envía la página solicitada por el cliente
-El proceso P2P envía el fichero solicitado por otro proceso P2P

Notas:

Medio de conexión: Internet, Ethernet...

Aplicaciones Cliente-Servidor II. Puerto

- Un puerto es una dirección numérica de la cual se procesa un servicio.
- Son direcciones lógicas proporcionadas por el SO.



Procodis'08: VI- Sockets Laura Barros 7

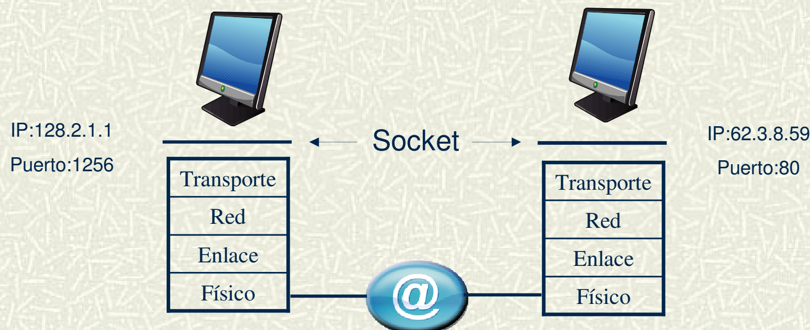
Notas:

Los puertos son direcciones lógicas proporcionadas por el sistema operativo para poder responder (distinguir de los puertos hardware).

Aplicaciones Cliente-Servidor III.Socket

- Es una abstracción del sistema operativo (no Hw)
  - Las aplicaciones los crean, los utilizan y los cierran cuando ya no son necesarios
  - Su funcionamiento está controlado por el sistema operativo
- Comunicación entre procesos
  - Los procesos envían/reciben mensajes a través de su socket
  - Los socket se comunican entre ellos
- La comunicación en Internet es de socket a socket
  - El proceso que está comunicándose se identifica por medio de su socket
  - El socket tiene un identificador

    Identificador = dir. IP del computador + núm. Puerto

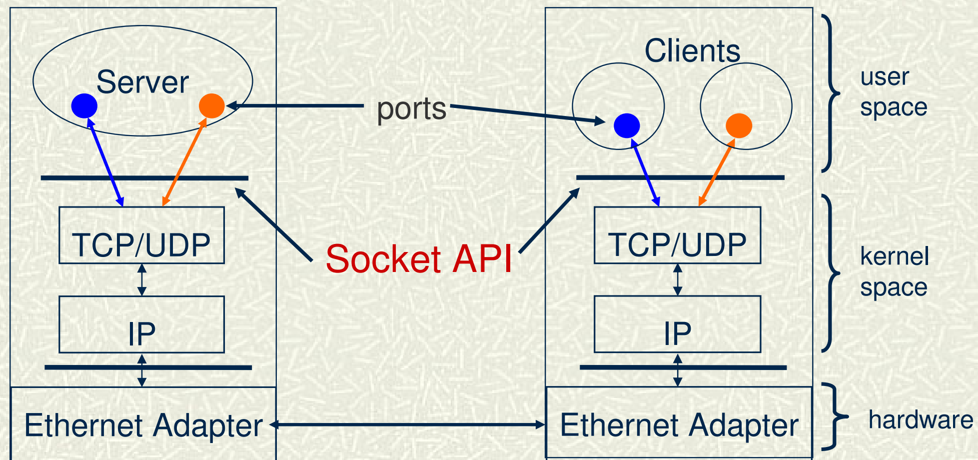


Notas:

The socket is the software abstraction used to represent the "terminals" of a connection between two machines. For a given connection, there's a socket on each machine, and you can imagine a hypothetical "cable" running between the two machines with each end of the "cable" plugged into a socket. Of course, the physical hardware and cabling between machines is completely unknown. The whole point of the abstraction is that we don't have to know more than is necessary. Bruce Eckel ,Thinking in Java.

Aplicaciones Cliente-Servidor IV.Socket API

- Servidor y Cliente intercambian mensajes a través de la red mediante la API Socket.



Notas:

Los procesos de las aplicaciones residen en el espacio de **usuario.**

Los procesos de los protocolos de transporte forman parte del **S.O.**

Se necesita un mecanismo para ponerlos en contacto: API (Application Programming Interface):

-Permite a las aplicaciones utilizar los protocolos de la pila TCP/IP.

-Define las operaciones permitidas y sus argumentos:

Similar a la forma de acceder a los ficheros en Unix

- Operaciones: open, read,write, close
- Cuando se abre un fichero obtenemos un descriptor

IP,Internet Protocol

- Protocolo usado para la comunicación entre dos aplicaciones que usan como medio de comunicación Internet.
- Se encarga de mover datos en forma de paquetes entre un origen y un destino.
- Todo dispositivo conectado a Internet posee al menos un identificador (una dirección IP de 4 bytes=32 bits) que lo define unívocamente.
- Origen y destino, basan su comunicación en la utilización de direcciones IP.

Notas:

TCP, Transfer Control Protocol

- Protocolo incorporado al protocolo IP para dar seguridad a la comunicación realizada a través del protocolo IP.
- Se utiliza para corroborar 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.
- Puede pedir la retransmisión de los paquetes que hubiesen llegado en mal estado o se hubiesen perdido.
- El funcionamiento de este protocolo, es similar al de una llamada de teléfono.

Notas:

Llamada de teléfono: en primer lugar, el equipo local solicita al remoto el establecimiento de un canal de comunicación; y solamente cuando el canal ha sido creado, y ambas máquinas están preparadas para la transmisión, empieza la transferencia de datos real.

Analogía de TCP con llamada telefónica

TCP	Llamada telefónica
Garantía de llegada	Garantía de llegada
Byte stream – llegada en orden	Llegada en orden
Orientado a la conexión – un socket por conexión	Orientado a la conexión
Se establece la conexión y se envían los datos	Se establece la conexión y se establece la conversación

Notas:

UDP, User Datagram Protocol

- Protocolo que se utiliza combinado con el protocolo IP, en aquellos tipos de comunicaciones en los que no resulta tan importante que lleguen todos los mensajes a un destinatario, o que lleguen en el orden en que se han enviado.
- Es preferible que cada paquete llegue lo más rápidamente posible, incluso a costa de perder algunos paquetes.
- Desventajas respecto al protocolo TCP:
  - Es un protocolo menos fiable.
  - Los procesos que hagan uso de UDP han de implementar, si es necesario, sus propias rutinas de verificación de envío y sincronización.
- El funcionamiento de este protocolo, es similar al del envío de una carta.

Notas:

Analogía de UDP con envío postal

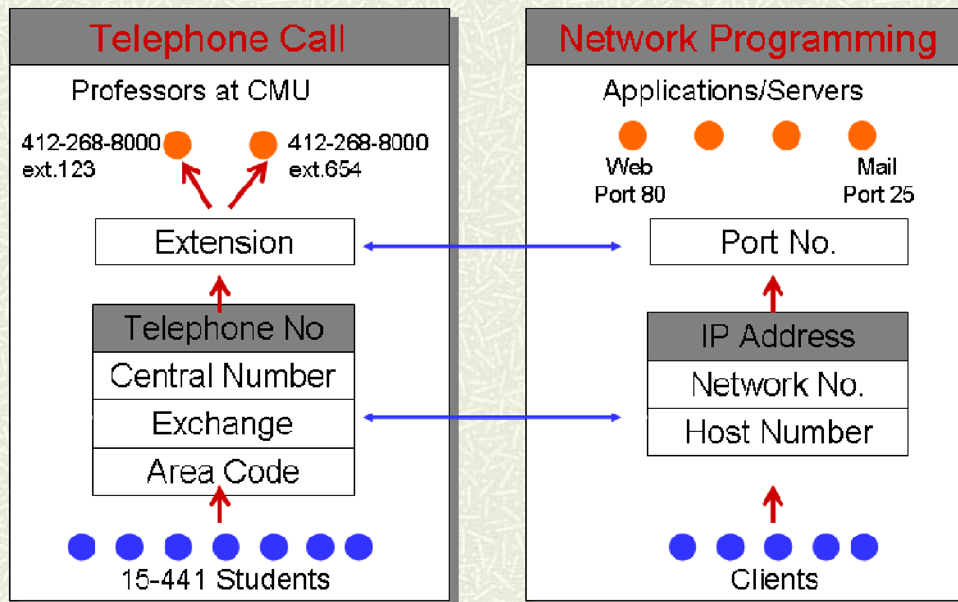
UDP

- Un socket para recibir los mensajes
- No hay garantía de llegada
- La llegada no es necesariamente en orden
- Datagram – paquetes independientes
- Muchas direcciones cada paquete

Envío Postal

- Un buzón para recibir las cartas
- No hay garantía de llegada
- La llegada no es necesariamente en orden
- Las cartas son independientes
- Una respuesta puede ir a varias direcciones

Notas:



Notas:

Nombres en Internet. La clase InetAddress

- Todas las direcciones en Internet son a través de una cantidad numérica o IP de 4 cifras.
- Una dirección se corresponde con un nombre (dominio), constituido por una cadena de caracteres.
- El Sistema de Nombres de Dominio (DNS) fue desarrollado para realizar la conversión entre los dominios y las direcciones IP.
- Java proporciona la clase InetAddress para la manipulación y conocimiento de direcciones y dominios. Representa una dirección IP.
- Métodos:
  - byte[] getAddress() =>Devuelve la dirreción IP de un objeto InetAddress.
  - InetAdress getByName(String host)=>Devuelve un objeto InetAddress representando el host que se le pasa como parámetro, bien como nombre o bien como parámetro.
  - InetAdress[] getAllByName(String host)=>Devuelve un array de objetos InetAddress que se puede utilizar para determinar todas las direcciones IP asignadas al host.
  - static InetAddressgetByAddress(byte[] addr) =>Devuelve un objeto InetAdress, dada una dirección IP.
  - static InetAddressgetByAddress(String host, byte[] addr) =>Devuelve un objeto InetAddress a partir del host y la dirección IP dada.
  - InetAdress getLocalHost()=>Devuelve un objeto InetAddress representando el ordenador local en el que se ejecuta la aplicación.

Notas:

Java proporciona la clase InetAddress para la manipulación y conocimiento de direcciones y dominios:Permite encontrar un nombre de dominio a partir de su dirección IP y viceversa.

Ejemplo de uso de InetAddress

```
import java.net.*;
public class ExtraePropiedades {
public static void main(String[] args) {
		try {
				System.out.println("-> Direccion IP de una URL, por nombre");
				InetAddress address = InetAddress.getByName("www.google.com");
				// Se imprime el objeto InetAddress obtenido
				System.out.println(address);
				System.out.println("-> Nombre a partir de la direccion");
				int temp = address.toString().indexOf('/');
				address = InetAddress.getByName(address.toString().substring(0, temp));
				System.out.println(address);
				System.out.println("-> Direccion IP actual de LocalHost");
				address = InetAddress.getLocalHost();
				System.out.println(address);
				System.out.println("-> Nombre de LocalHost a partir de la direccion");
				temp = address.toString().indexOf('/');
				address = InetAddress.getByName(address.toString().substring(0, temp));
				System.out.println(address);
				System.out.println("-> Nombre actual de LocalHost");
				System.out.println(address.getHostName());
				System.out.println();
		} catch (UnknownHostException e) {
				System.out.println(e);
				System.out.println("Debes estar conectado para que esto funcione bien.");
		}
	}
}
```


Notas:

InetAddress Example

-> Direccion IP de una URL, por nombre

www.google.com/74.125.77.99

-> Nombre a partir de la direccion

www.google.com/74.125.77.99

-> Direccion IP actual de LocalHost

MYHOUSE/192.168.2.3

-> Nombre de LocalHost a partir de la direccion

MYHOUSE/192.168.2.3

-> Nombre actual de LocalHost

MYHOUSE

Notas:

Sockets Java

- Representan los extremos del canal de comunicación.
- La conexión entre sockets es full-duplex.
- Hay dos tipos:
  - Sockets Stream (Sockets TCP):
    - Los datos son transmitidos en bytes (no son empaquetados en registros o paquetes).
    - Para establecer la comunicación, utilizan el protocolo TCP.
    - La conexión empezaría una vez que los dos sockets estén conectados.
    - Para crear aplicaciones con este socket, Java proporciona dos clases, Socket y ServerSocket.
  - Sockets Datagrama (Sockets UDP):
    - Los datos son enviados y recibidos en paquetes denominados datagramas.
    - Para establecer la comunicación entre estos sockets se usará el protocolo UDP.
    - Aunque se deben enviar datos adicionales, este socket es más eficiente que el anterior, aunque menos seguro.
    - Para crear aplicaciones con este socket, Java proporciona la clase DatagramSocket.
- La utilización de los sockets es muy similar a la utilización de ficheros.

Notas:

Una conexión a través de una red, implica que como mínimo intervengan dos partes, entre las cuales se abre un canal de comunicación.

Full-duplex: en ambos sentidos a la vez, entre las dos partes que intervienen.

Para establecer la comunicación, utilizan el protocolo TCP, de tal forma que si re rompe la conexión entre las partes integrantes de la comunicación, éstas serán informadas de tal evento para que reaccionen.

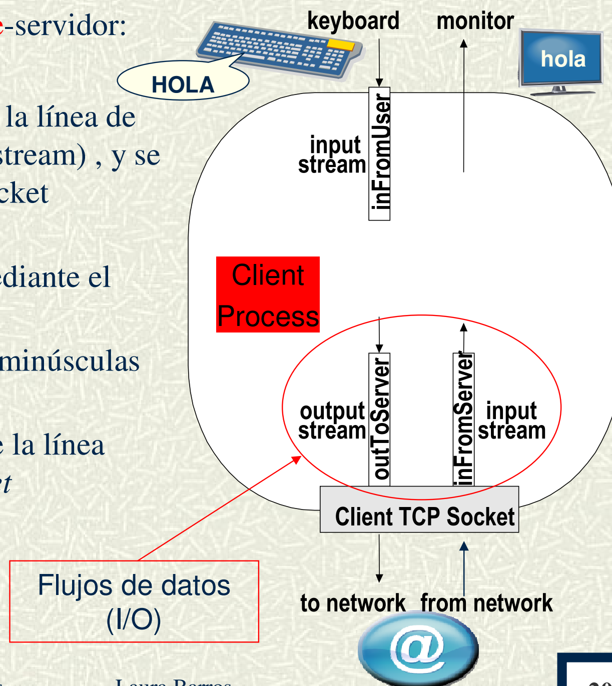
La conexión empezaría una vez que los dos sockets estén conectados; en ese momento se puede realizar la transmisión de datos en ambas direcciones.

Para establecer la comunicación entre sockets datagrama, se usará el protocolo UDP, por lo tanto no hay seguridad de que los paquetes lleguen a su destino.

Programando Socket con TCP

Ejemplo de aplicación cliente-servidor:

- El cliente lee una línea de la línea de comandos (**inFromUser** stream) , y se la envía al servidor via socket (**outToServer** stream)
- El servidor lee la línea mediante el *socket*
- El servidor la convierte a minúsculas y se la envía al cliente
- El cliente la lee e imprime la línea modificada desde el *socket* (**inFromServer** stream)



Procodis'08: VI- Sockets Laura Barros 20

Notas:

InFromUser, outputStream...: implica la existencia de un registro o buffer (espacio de memoria) para guardar el flujo.

Sockets Cliente/Servidor en Java: TCP

Server (running on **Host A**)

Client (running on Host *B*)

create socket,
port=**x**, for
incoming request:
welcomeSocket =
ServerSocket(*x*)

TCP
connection setup

wait for incoming
connection request
connectionSocket =
welcomeSocket.accept()

create socket,
connect to **hostid**, port=**x**
clientSocket =
Socket(*A*, *x*)

Establecimiento de la conexión

send request using
clientSocket

read request from
connectionSocket

write reply to
connectionSocket

read reply from
clientSocket

Provisión del servicio

close
connectionSocket

close
clientSocket

Cierre de la conexión

Notas:

La clase Socket

- Pasos de utilización:
  - Instanciar el socket.
  - Abrir el canal de E/S.
  - Cerrar el socket.
- Constructores:
  - Socket()=>Creates an unconnected socket, with the system-default type of SocketImpl.
  - Socket(InetAddress address, int port) =>Creates a stream socket and connects it to the specified port number at the specified IP address.
  - Socket(InetAddress address, int port, InetAddress localAddr, int localPort) =>Creates a socket and connects it to the specified remote address on the specified remote port.
  - Socket(SocketImpl impl) =>Creates an unconnected Socket with a user-specified SocketImpl.
    - Para la confección de diferentes protocolos o variantes de los mismos Java proporciona un interfaz denominado **SocketImplFactory**.
    - **SocketImpl createSocketImpl()=>**Crea una instancia de la superclase de sockets.
      - **SocketImpl** es la superclase de todos las implementaciones de sockets. Un objeto de esta clase es usado como argumento para la creación de sockets.
  - Socket(String host, int port)=> Creates a stream socket and connects it to the specified port number on the named host.
  - Socket(String host, int port, InetAddress localAddr, int localPort) =>Creates a socket and connects it to the specified remote host on the specified remote port.

Notas:

La clase Socket I

- Métodos:
  - void bind(SocketAddress bindpoint) =>Binds the socket to a local address.
  - void close() => Closes this socket.
  - void connect(SocketAddress endpoint)=> Connects this socket to the server.
  - void connect(SocketAddress endpoint, int timeout)=>Connects this socket to the server with a specified timeout value.
  - SocketChannel getChannel()=> Returns the unique SocketChannel object associated with this socket, if any.
  - InetAddress getInetAddress()=>Returns the address to which the socket is connected.
  - InputStream getInputStream()=>Returns an input stream for this socket.
  - boolean getKeepAlive()=>Tests if SO_KEEPALIVE is enabled.
  - InetAddress getLocalAddress()=>Gets the local address to which the socket is bound.
  - int getLocalPort()=>Returns the local port to which this socket is bound.
  - SocketAddress getLocalSocketAddress() =>Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
  - boolean getOOBInline()=>Tests if OOBINLINE is enabled.

Notas:

La clase Socket II

- OutputStream getOutputStream()=>Returns an output stream for this socket.
- int getPort() =>Returns the remote port to which this socket is connected.
- int getReceiveBufferSize()=> Gets the value of the SO_RCVBUF option for this Socket, that is the buffer size used by the platform for input on this Socket.
- SocketAddress getRemoteSocketAddress() =>Returns the address of the endpoint this socket is connected to, or null if it is unconnected.
- boolean getReuseAddress()=> Tests if SO_REUSEADDR is enabled.
- int getSendBufferSize()=> Get value of the SO_SNDBUF option for this Socket, that is the buffer size used by the platform for output on this Socket.
- int getSoLinger() =>Returns setting for SO_LINGER.
- int getSoTimeout() =>Returns setting for SO_TIMEOUT.
- boolean getTcpNoDelay() => Tests if TCP_NODELAY is enabled.
- int getTrafficClass() => Gets traffic class or type-of-service in the IP header for packets sent from this Socket.
  - Tipo de servicio (TOS): octeto del paquete IP que indica parámetros de la QoS (retardos, ancho de banda…).
- boolean isBound() =>Returns the binding state of the socket.
- boolean isClosed() =>Returns the closed state of the socket.

Notas:

La clase Socket III

- boolean isConnected() =>Returns the connection state of the socket.
- boolean isInputShutdown() =>Returns wether the read-half of the socket connection is closed.
- boolean isOutputShutdown()=>Returns wether the write-half of the socket connection is closed.
- void sendUrgentData(int data) => Send one byte of urgent data on the socket.
- void setKeepAlive(boolean on)=>Enable/disable SO_KEEPALIVE.
- void setOOBInline(boolean on) =>Enable/disable OOBINLINE (receipt of TCP urgent data) By default, this option is disabled and TCP urgent data received on a socket is silently discarded.
- void setReceiveBufferSize(int size) =>Sets the SO_RCVBUF option to the specified value for this Socket.
- void setReuseAddress(boolean on) => Enable/disable the SO_REUSEADDR socket option.
- void setSendBufferSize(int size)=> Sets the SO_SNDBUF option to the specified value for this Socket.
- static void setSocketImplFactory(SocketImplFactory fac)=> Sets the client socket implementation factory for the application.

Notas:

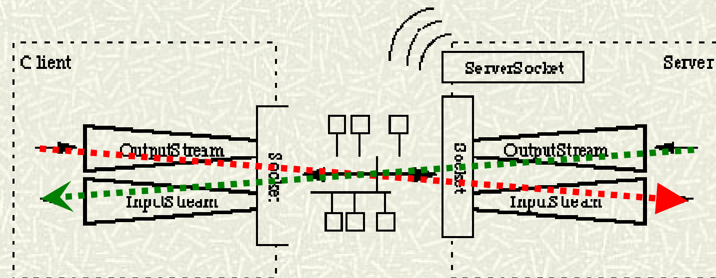
La clase Socket IV

- void setSoLinger(boolean on, int linger)=>Enable/disable SO_LINGER with the specified linger time in seconds.
- void setSoTimeout(int timeout) =>Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
- void setTcpNoDelay(boolean on) => Enable/disable TCP_NODELAY (disable/enable Nagle's algorithm).
  - El Algoritmo de Nagle se trata de un procedimiento que supone una mejora y aumento de eficiencia de las redes de comunicación basadas en Transmission Control Protocol (TCP). El algoritmo de Nagle es un método heurístico para evitar enviar paquetes IP particularmente pequeños). El algoritmo de Nagle intenta evitar la congestión que estos paquetes pueden ocasionar en la red reteniendo por poco tiempo la transmisión de datos TCP en algunas circunstancias.
- void setTrafficClass(int tc) => Sets traffic class or type-of-service octet in the IP header for packets sent from this Socket.
- void shutdownInput()=>Places the input stream for this socket at "end of stream".
- void shutdownOutput() =>Disables the output stream for this socket.
- String toString() => Converts this socket to a String.

Notas:

Gestión de los flujos

- La lectura y la escritura sobre sockets TCP se realiza por medio de objetos derivados de las clases de Stream (en concreto subclases de **InputStream** y **OutputStream**).
  - **InputStream getInputStream()**
    Obtiene el *stream* de lectura.
  - **OutputStream getOutputStream()**
    Obtiene el *stream* de escritura.
- Los objetos devueltos son transformados a la subclase apropiada para manejarlo (por ejemplo **DataInputStream**).



Notas:

Ejemplo de uso de Socket

```
import java.net.*;
import java.io.*;
import java.util.*;
class LeeFicheroSocket {
		public static void main(String[] args) {
				try {
						// Instanciamos el socket con el constructor de la clase
						Socket socket = new Socket("barros.servidor.es", 80);
						// Abrimos los canales de E/S
						BufferedReader entrada = new BufferedReader(new InputStreamReader(
										socket.getInputStream()));
						PrintWriter salida = new PrintWriter(new OutputStreamWriter(socket
										.getOutputStream()), true);
						// Utilizamos los métodos de las clases Reader y Writer.
						System.out.println("GET/samples/default.htm");
						String linea = null;
						while ((linea = entrada.readLine()) != null)
								System.out.println(linea);
						// Se cierra el canal de entrada
						entrada.close();
						// Se cierra el socket
						socket.close();
				} catch (UnknownHostException e) {
						System.out.println("Problemas de red");
				} catch (IOException e) {}
		}
}
```


Notas:

La clase ServerSocket I

- Se utiliza para construir servidores que están a la espera de peticiones por parte de los clientes.
- Constructor:
  - ServerSocket() =>Creates an unbound server socket.
  - ServerSocket(int port)=>Creates a server socket, bound to the specified port.
  - ServerSocket(int port, int backlog) =>Creates a server socket and binds it to the specified local port number, with the specified backlog.
  - ServerSocket(int port, int backlog, InetAddress bindAddr)=>Create a server with the specified port, listen backlog, and local IP address to bind to.

Notas:

La clase ServerSocket II

- Socket accept() =>Listens for a connection to be made to this socket and accepts it.
- void bind(SocketAddress endpoint)=>Binds the ServerSocket to a specific address (IP address and port number).
- void bind(SocketAddress endpoint, int backlog) =>Binds the ServerSocket to a specific address (IP address and port number).
- void close() =>Closes this socket.
- ServerSocket Channel getChannel() =>Returns the unique ServerSocketChannel object associated with this socket, if any.
- InetAddress getInetAddress()=>Returns the local address of this server socket.
- int getLocalPort() =>Returns the port on which this socket is listening.
- SocketAddress getLocalSocketAddress()=>Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
- int getReceiveBufferSize() =>Gets the value of the SO_RCVBUF option for this ServerSocket, that is the proposed buffer size that will be used for Sockets accepted from this ServerSocket.
- boolean getReuseAddress() => Tests if SO_REUSEADDR is enabled.
- int getSoTimeout() =>Retrive setting for SO_TIMEOUT.

Notas:

La clase ServerSocket III

- protected void implAccept(Socket s)=>Subclasses of ServerSocket use this method to override accept() to return their own subclass of socket.
- boolean isBound() =>Returns the binding state of the ServerSocket.
- boolean isClosed()=> Returns the closed state of the ServerSocket.
- void setReceiveBufferSize(int size) =>Sets a default proposed value for the SO_RCVBUF option for sockets accepted from this ServerSocket.
- void setReuseAddress(boolean on) =>Enable/disable the SO_REUSEADDR socket option.
- static voidsetSocketFactory(SocketImplFactory fac)=>Sets the server socket implementation factory for the application.
- void setSoTimeout(int timeout) => Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
- String toString() =>Returns the implementation address and implementation port of this socket as a String.

Notas:

Ejemplo de uso de ServerSocket. Server

```
import java.io.*;
import java.net.*;
public class Server {
    public static void main(String args[]) {
            ServerSocket server;
            Socket connection;
            DataOutputStream output;
            DataInputStream input;
            int counter = 1, SIZE = 100;
            try {
                        // step 1 Creamos un ServerSocket
                        server = new ServerSocket(5000, 100);
                        while (true) {
                                    // step 2 Esperamos a la conexión
                                    connection = server.accept();
                                    System.out.println("Connection " + counter + "received from:"
                                                        + connection.getInetAddress().getHostName());
                                    // step 3 Abrimos los canales de E/S
                                    input = new DataInputStream(connection.getInputStream());
                                    output = new DataOutputStream(connection.getOutputStream());
                                    // step 4 : Utilizamos los métods de Write y de Read
                                    output.writeUTF("connection sucessful ");
                                    System.out.println("Client Message " + input.readUTF());
                                    // step 5 Cerramos la conexión
                                    connection.close();
                                    ++counter;
                        }
            } catch (IOException e) {}
    }
}
```

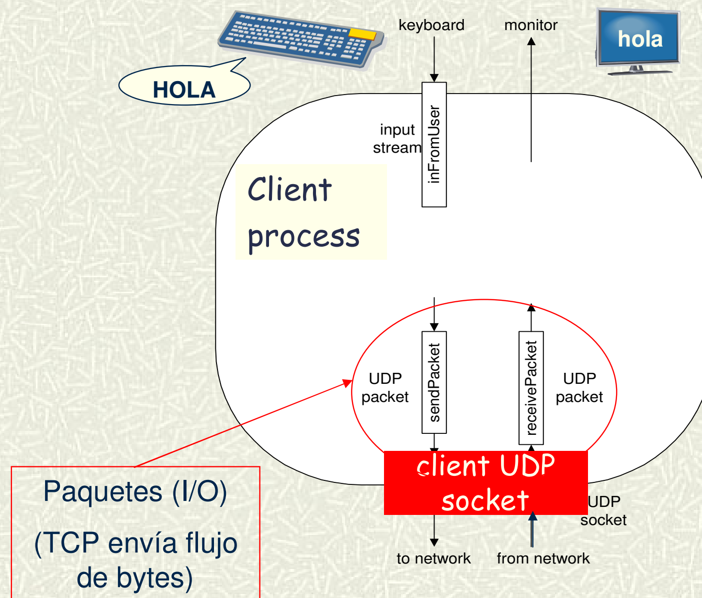

Notas:

Ejemplo de uso de ServerSocket.Client

```
import java.io.*;
import java.net.*;
public class Client {
    public static void main(String args[]) {
          Socket client;
          DataOutputStream output;
          DataInputStream input;
          try {
                    // step 1 Creamos el socket para establecer la conexión
                    client = new Socket(InetAddress.getLocalHost(), 5000);
                    // step 2 Abrimos los canales de E/S
                    input = new DataInputStream(client.getInputStream());
                    output = new DataOutputStream(client.getOutputStream());
                    // step 3 : Utilizamos los métodos de las clases Read y Write
                    System.out.println("Server Message " + input.readUTF());
                    output.writeUTF("Thank You ");
                    // step 4 Cerramos la conexión
                    client.close();
          } catch (IOException e) {
          }
    }
}
```


Notas:

Programando Socket con UDP



Paquetes (I/O)

(TCP envía flujo de bytes)

Notas:

Sockets Cliente/Servidor en Java: UDP

Server (running on Host A)

Client (running on Host B)

create socket, port=x, for incoming request: serverSocket = DatagramSocket(x)

create socket, clientSocket = DatagramSocket( )

No hay "conexión:
Client:especifica en cada paquete la IP de destino.
Server:debe extraer la IP y el puerto del cliente de cada paquete recibido.

read request from serverSocket

write reply to serverSocket specifying client host address, port number

read reply from clientSocket

Provisión del servicio

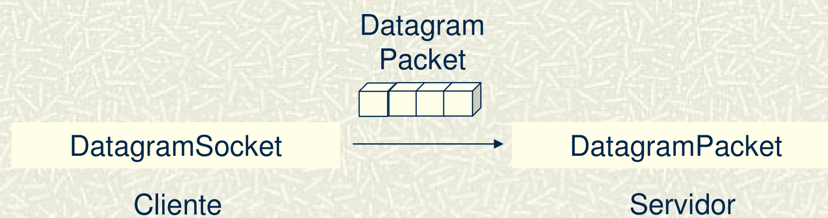
close clientSocket

Cierre de la conexión

Notas:

Datagram Socket y Datagram Packet

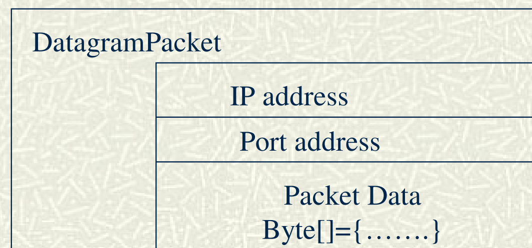
- Dos clases:
  - **DatagramSocket**: para enviar o recibir datagramas.
  - **DatagramPacket**: paquete de datos o datagrama.



Notas:

Datagram Packet

- DatagramPacket representa una paquete de datos propuesto para la transmisión UDP.
- Los paquetes son contenedores para una pequeña secuencia de bytes que incluye información de direccionamiento tal como , una dirección IP y un puerto.
- El significado de los datos almacenados en un DatagramPacket se determina por su contexto. En la recepción, la dirección IP de una socket UDP, representa la dirección del remitente, en tanto que para el envío, la dirección IP representa la dirección del destinatario.
- Un datagrama consta de:
  - Una cabecera: dirección de origen y destino del paquete, el puerto, la longitud del paquete, un checksum, etc.
  - Cuerpo: datos del paquete.



Notas:

La Clase DatagramPacket

- Hay dos razones para crear una instancia de DatagramPacket:
  - Para enviar datos a una máquina remota usando UDP
  - Para recibir datos de una máquina remota usando UDP.
- Constructores:
  - DatagramPacket(byte[] buf, int length) =>Constructs a DatagramPacket for receiving packets of length length.
  - DatagramPacket(byte[] buf, int length, InetAddress address, int port)=>Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.
  - DatagramPacket(byte[] buf, int offset, int length) =>Constructs a DatagramPacket for receiving packets of length length, specifying an offset into the buffer.
  - DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port) =>Constructs a datagram packet for sending packets of length length with offset offset to the specified port number on the specified host.
  - DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)=>Constructs a datagram packet for sending packets of length length with offset ioffset to the specified port number on the specified host.
  - DatagramPacket(byte[] buf, int length, SocketAddress address) =>Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

Notas:

La Clase DatagramPacket II

- Métodos:
  - InetAddress getAddress()=> Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
  - byte[] getData()=>Returns the data buffer.
  - int getLength() =>Returns the length of the data to be sent or the length of the data received.
  - int getOffset()=>Returns the offset of the data to be sent or the offset of the data received.
  - int getPort() =>Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.
  - SocketAddressgetSocketAddress()=>Gets the SocketAddress (usually IP address + port number) of the remote host that this packet is being sent to or is coming from.
  - void setAddress(InetAddress iaddr) =>Sets the IP address of the machine to which this datagram is being sent.
  - void setData(byte[] buf) =>Set the data buffer for this packet.
  - void setData(byte[] buf, int offset, int length)=>Set the data buffer for this packet.
  - void setLength(int length)=>Set the length for this packet.
  - void setPort(int iport)=>Sets the port number on the remote host to which this datagram is being sent.
  - void setSocketAddress(SocketAddress address) =>Sets the SocketAddress (usually IP address + port number) of the remote host to which this datagram is being sent.

Notas:

La clase DatagramSocket I

- Ofrece el acceso a los sockets UDP, que permiten enviar y recibir paquetes UDP. El mismo socket que puede ser usado para recibir paquetes, puede usarse para enviarlos. Las operaciones de lectura son bloqueantes (la aplicación esperará hasta que el paquete llegue).
- Constructores:
  - DatagramSocket()=>Constructs a datagram socket and binds it to any available port on the local host machine.
  - protected DatagramSocket(DatagramSocketImpl impl)=>Creates an unbound datagram socket with the specified DatagramSocketImpl.
  - DatagramSocket(int port) => Constructs a datagram socket and binds it to the specified port on the local host machine.
  - DatagramSocket(int port, InetAddress laddr)=>Creates a datagram socket, bound to the specified local address.
  - DatagramSocket(SocketAddress bindaddr) =>Creates a datagram socket, bound to the specified local socket address.

Notas:

La clase DatagramSocket II

- void bind(SocketAddress addr)=>Binds this DatagramSocket to a specific address & port.
- void close() => Closes this datagram socket.
- void connect(InetAddress address, int port) => Connects the socket to a remote address for this socket.
- void connect(SocketAddress addr)=>Connects this socket to a remote socket address (IP address + port number).
- void disconnect() =>Disconnects the socket.
- boolean getBroadcast() => Tests if SO_BROADCAST is enabled.
- Datagram ChannelgetChannel() =>Returns the unique DatagramChannel object associated with this datagram socket, if any.
- InetAddress getInetAddress()=>Returns the address to which this socket is connected.

Notas:

La clase DatagramSocket III

- InetAddress getLocalAddress() =>Gets the local address to which the socket is bound.
- Int getLocalPort() =>Returns the port number on the local host to which this socket is bound.
- SocketAddress getLocalSocketAddress() =>Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
- int getPort()=>Returns the port for this socket.
- int getReceiveBufferSize() =>Get value of the SO_RCVBUF option for this DatagramSocket, that is the buffer size used by the platform for input on this DatagramSocket.
- SocketAddress getRemoteSocketAddress() =>Returns the address of the endpoint this socket is connected to, or null if it is unconnected.
- boolean getReuseAddress() =>Tests if SO_REUSEADDR is enabled.
- int getSendBufferSize()=> Get value of the SO_SNDBUF option for this DatagramSocket, that is the buffer size used by the platform for output on this DatagramSocket.

Notas:

La clase DatagramSocket IV

- void receive(DatagramPacket p) => Receives a datagram packet from this socket.
- void send(DatagramPacket p)=> Sends a datagram packet from this socket.
- Void setBroadcast(boolean on) => Enable/disable SO_BROADCAST.
- static void setDatagramSocketImplFactory(DatagramSocketImplFactory fac)

=>Sets the datagram socket implementation factory for the application.

- void setReceiveBufferSize(int size) =>Sets the SO_RCVBUF option to the specified value for this DatagramSocket.
- void setReuseAddress(boolean on) =>Enable/disable the SO_REUSEADDR socket option.
- void setSendBufferSize(int size)=> Sets the SO_SNDBUF option to the specified value for this DatagramSocket.
- void setSoTimeout(int timeout) =>Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
- void setTrafficClass(int tc) => Sets traffic class or type-of-service octet in the IP datagram header for datagrams sent from this DatagramSocket.

Notas:

Ejemplo de uso de DatagramPacket/Server

```
/* A single threaded UDP socket server *  UDPServer.java */
public class UDPServer {
        static boolean debug = true;
        static void dbg(String str) {
                if (debug) {
                        System.out.println(str);
                        System.out.flush();
                }
        }
public static void main(String args[]) {
        DatagramSocket socket = null;
        DatagramPacket sendPkt, recPkt;
        String recStr = "                ";
        try {
                // step create server socket
                socket = new DatagramSocket(42424);
                while (true) {
                        dbg("Starting Server on port = " + socket.getLocalPort());
                        byte buf[] = new byte[100];
                        recPkt = new DatagramPacket(buf, 100);
                        socket.receive(recPkt);
                        recStr = new String(recPkt.getData());
                        dbg("Received from Inetaddress: " + recPkt.getAddress()+ " port number: " + recPkt.getPort() + " Length = "+
                        recPkt.getLength() + " Containing : " + recStr);
                        // Echo information from packet back to client
                        sendPkt = new DatagramPacket(recPkt.getData(), recPkt.getLength(), recPkt.getAddress(), recPkt.getPort());
                        socket.send(sendPkt);
                        }
                } catch (IOException e) {}}}
```


Notas:

Ejemplo de uso de DatagramPacket/Server.UDPClient I

```
/* A single threaded UDP socket server **  UDPServer.java */
public class UDPClient {
	static boolean debug = true;
	private static DatagramSocket socket = null;
	private static DatagramPacket sendPkt, recPkt;
	static void dbg(String str) {
		if (debug) {
			System.out.println(str);
			System.out.flush();
		}
	}
	public UDPClient() {
		try {
			socket = new DatagramSocket();
		} catch (SocketException se) {}
	}
```


Notas:

Ejemplo de uso de DatagramPacket/Server.UDPClient II

```
    public void sendrecv() {
        try {
            for (int j = 0; j < 3; j++) {
                String s[] = { "Sending Packet  1 ", "Sending Packet  2 ","LAST" };
                byte data[] = s[j].getBytes();
                sendPkt = new DatagramPacket(data, data.length, InetAddress
                                .getLocalHost(), 42424);
                socket.send(sendPkt);
                dbg(s[j]);
                byte recData[] = new byte[100];
                recPkt = new DatagramPacket(recData, recData.length);
                // wait for packet
                socket.receive(recPkt);
                dbg(" Received Packet is : " + " From Host "
                        + recPkt.getAddress() + " HostPort = "
                        + recPkt.getPort() + " Length = " + recPkt.getLength()
                        + " Containing " + new String(recPkt.getData()));
            }
        } catch (IOException e) {}
    }
    public static void main(String args[]) {
        UDPClient c = new UDPClient();
        c.sendrecv();
    }
}
```


Notas: