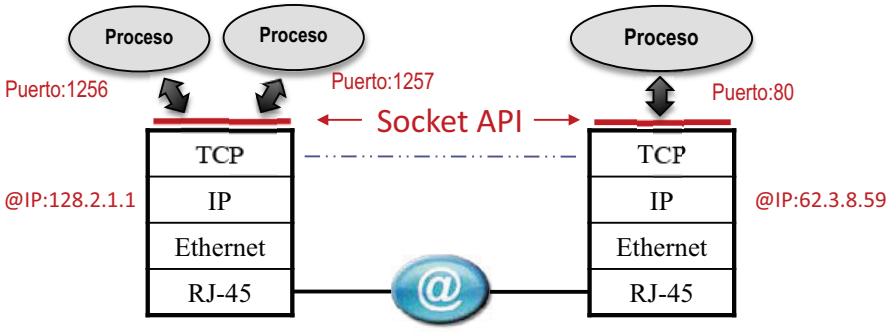
PROGRAMACION DISTRIBUIDA

Mecanismo básico de comunicación: Sockets en Java

Héctor Pérez

Introducción: Concepto de Socket

- Es una abstracción software proporcionada por el sistema operativo
  - representa la interfaz entre la aplicación y la red de comunicaciones
  - las aplicaciones los crean, los utilizan y los cierran cuando ya no son necesarios
  - su funcionamiento está controlado por el sistema operativo
- Habilita la comunicación entre procesos
  - los procesos envían/reciben mensajes a través de su socket
  - la comunicación se realiza de socket a socket
  - permite la **localización** de aplicaciones y el **intercambio de información** entre ellas
    - Por ejemplo, un socket TCP/IP contiene la @IP del computador y el número de puerto



Introducción: Aplicaciones cliente/servidor

- *La comunicación a través del protocolo TCP/IP está estrechamente relacionada con la arquitectura **cliente/servidor***
  - El **servidor** utiliza sockets para esperar la llegada de peticiones del *cliente*
  - El **cliente** utiliza sockets para realizar las peticiones al *servidor*

Cliente

Inicia la comunicación

Solicita un servicio al servidor

Ejemplos:

- ✓ Un cliente web solicita la página www.istr.unican.es\index.html

Servidor

Espera peticiones

Proporciona el servicio solicitado

Ejemplo:

- ✓ El servidor web envía la página solicitada por el cliente

La clase Java InetAddress: Nombres en Internet

- Una dirección IP se representa mediante un identificador numérico de 32 bits
  - también se puede hacer corresponder con un nombre de dominio simbólico
  - el Servicio de Nombres de Dominio (***DNS***) establece la correspondencia entre nombres de dominio y direcciones IP
- La clase Java **InetAddress** representa una dirección IP, y proporciona métodos para la gestión de direcciones y nombres de dominio:

byte[] getAddress()	Devuelve la dirección IP de un objeto InetAddress.
static InetAdress getByName (String host)	Devuelve un objeto InetAddress representando el host que se le pasa como parámetro.
static InetAdress[] getAllByName (String host)	Devuelve un array de objetos InetAddress que se puede utilizar para determinar todas las direcciones IP asignadas al host.
static InetAddress getByAddress (**byte[]** addr)	Devuelve un objeto InetAdress a partir de una dirección IP.
static InetAddress getByAddress (String host, **byte[]** addr)	Devuelve un objeto InetAddress a partir del host y la dirección IP dada.
static InetAdress getLocalHost()	Devuelve un objeto InetAddress representando el ordenador local en el que se ejecuta la aplicación.

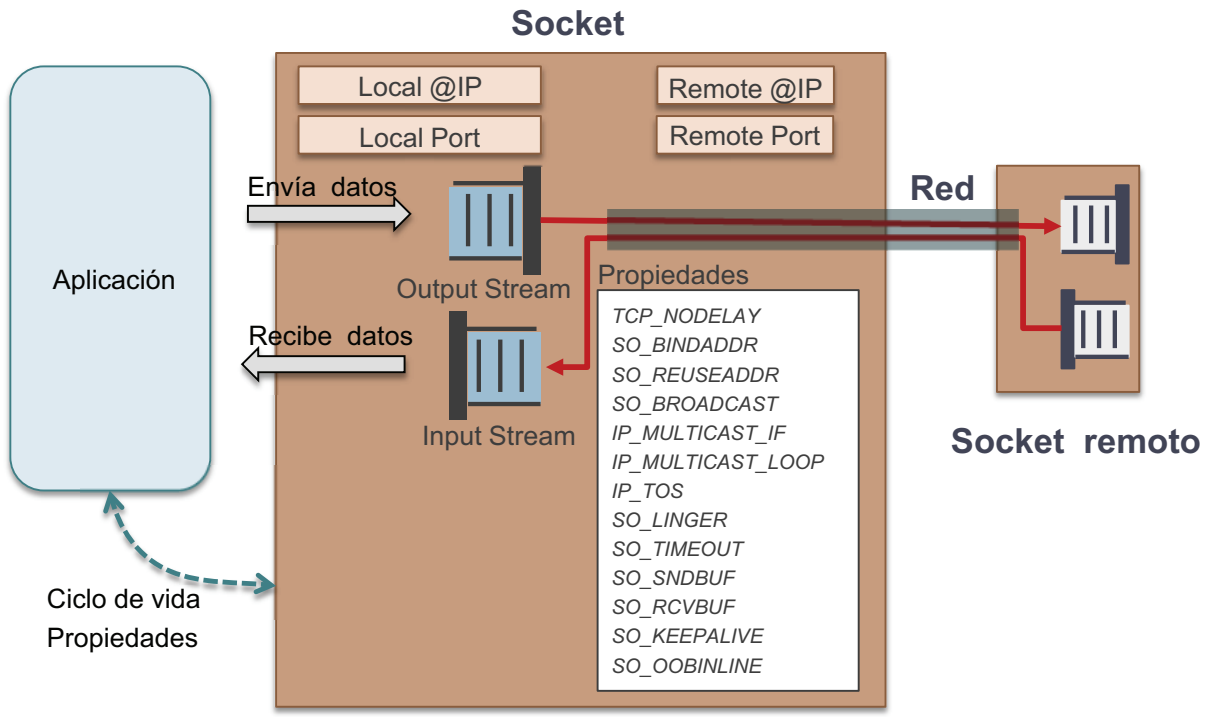
Ejemplo de uso de InetAddress

```
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");
    System.out.println(address);                                          print => www.google.com/173.194.34.242
    System.out.println("-> Nombre a partir de la direccion");
    int temp = address.toString().indexOf('/');
    address = InetAddress.getByName(address.toString().substring(temp + 1));
    System.out.println(address);                                          print => 173.194.34.242
    System.out.println("-> Direccion IP actual de LocalHost");
    address = InetAddress.getLocalHost();
    System.out.println(address);                                          print => SERVICI-1QBPB7N/192.168.0.193
    System.out.println("-> Nombre de LocalHost a partir de la direccion");
    temp = address.toString().indexOf('/');
    address = InetAddress.getByName(address.toString().substring(temp + 1));
    System.out.println(address);                                          print => 192.168.0.193
    System.out.println("-> Nombre actual de LocalHost");
    System.out.println(address.getHostName());                            print => SERVICI-1QBPB7N
 } catch (UnknownHostException e) {System.out.println("Error de conexión");}
 }
}
```


Sockets Java

- Representan los extremos o puntos lógicos de la comunicación
- La comunicación entre sockets es full-duplex
- Hay dos tipos:
  - Sockets TCP:
    - Los datos son transmitidos sin ser empaquetados en paquetes (*streams* o flujo de bytes).
    - La comunicación empezaría una vez que los dos sockets estén conectados.
    - Para crear aplicaciones con este socket, Java proporciona dos clases: **Socket** y **ServerSocket**.
  - Sockets UDP:
    - Los datos son enviados y recibidos en paquetes denominados *datagramas*.
    - Aunque se deben enviar datos adicionales, este socket es más eficiente que el socket TCP, aunque la comunicación es menos fiable.
    - Para crear aplicaciones con este socket, Java proporciona las clases **DatagramSocket** y **DatagramPacket**.
- La utilización de los sockets es muy similar a la utilización de ficheros

Sockets TCP: Elementos de Socket java



Sockets TCP: Funcionalidad del Socket Java

- Sobre un *Socket Java* se puede realizar las siguientes operaciones básicas:
  - Conectar con una máquina remota
  - Enviar datos
  - Recibir datos
  - Cerrar una conexión

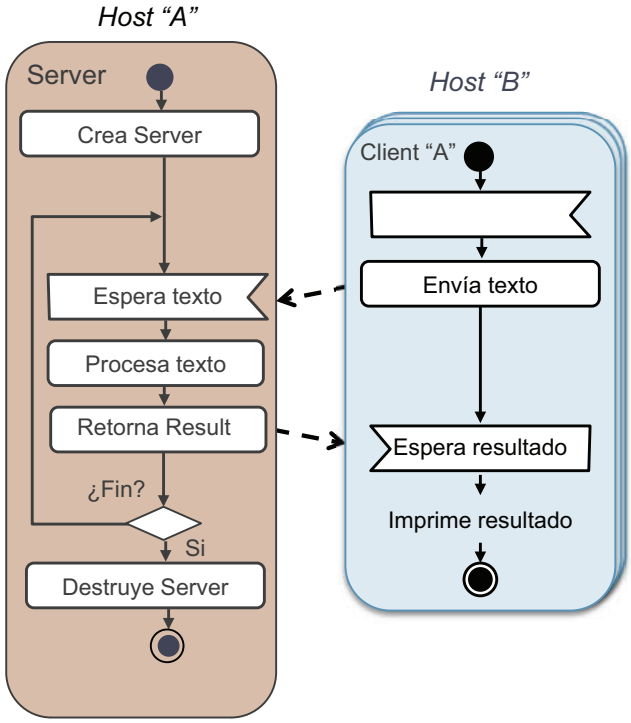
  Propias de clientes y servidores
  Clase Java **Socket**

  - Asociarse a un puerto
  - Esperar la llegada de conexiones
  - Aceptar la conexión de una máquina remota a un puerto asociado

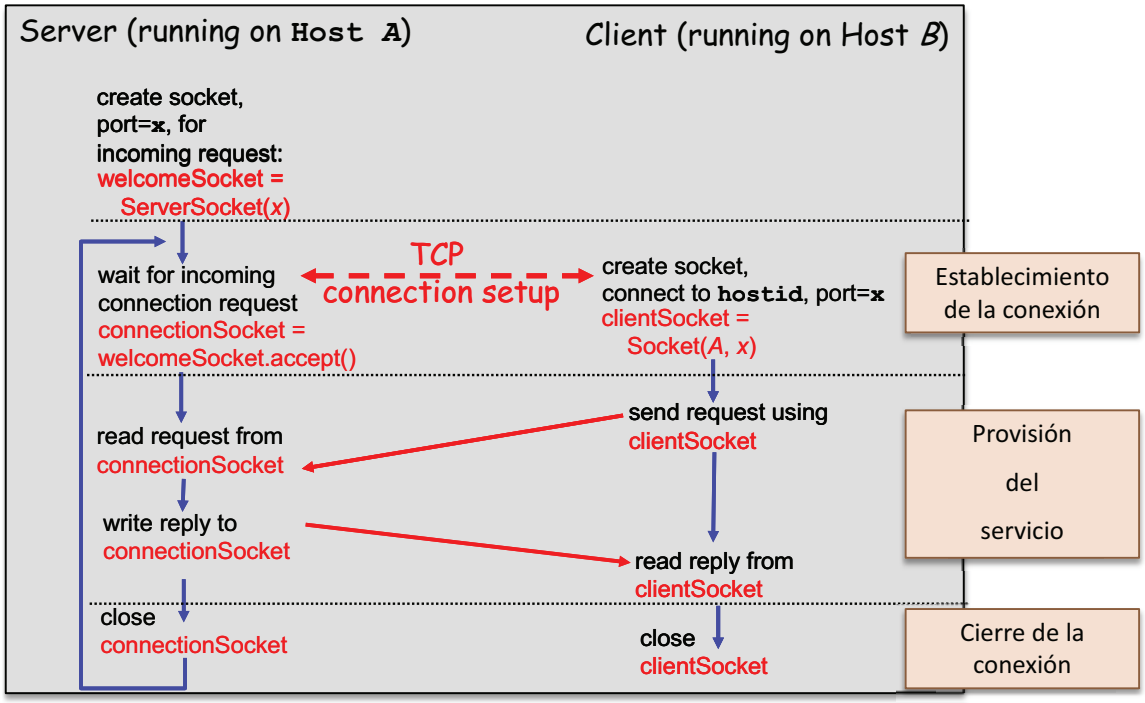
  Específica de servidores
  Clase Java **ServerSocket**

Sockets TCP: Ejemplo de aplicación cliente/servidor

- *Cliente*: Lee una línea de texto del teclado. Los envía al servidor vía el socket
- *Servidor*: Permanece a la espera de que algún cliente envía un texto. Cuando llega, lo lee, lo transforma sustituyendo las letras mayúsculas por minúsculas, y lo devuelve por el socket por el que lo recibió.
- *Cliente* espera en el socket a recibir la respuesta, lo lee y lo imprime.



Sockets TCP: Cliente/servidor en Java

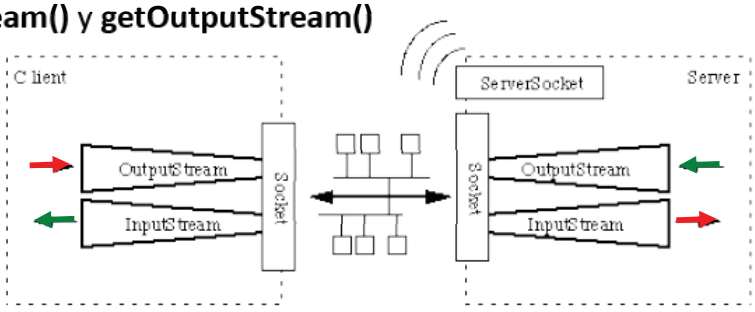


Sockets TCP: La clase Socket

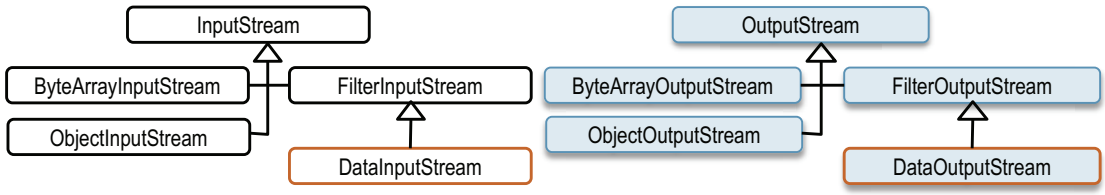
Socket (InetAddress address, **int** port)	*Creates a stream socket and connects it to the specified port number at the specified IP address*
Socket (String host, **int** port)	*Creates a stream socket and connects it to the specified port number on the named host*
void close()	*Closes this socket*
void shutdownInput()	*Places the input stream for this socket at "end of stream"*
void shutdownOutput()	*Disables the output stream for 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*
String toString()	*Converts this socket to a String*
OutputStream getOutputStream()	*Returns an output stream for this socket*
InputStream getInputStream()	*Returns an input stream for this socket*

Sockets TCP: Gestión de los flujos

- La lectura y la escritura sobre ***sockets*** TCP se realiza por medio de objetos **InputStream** y **OutputStream,** que se obtienen respectivamente por los métodos **getInputStream()** y **getOutputStream()**

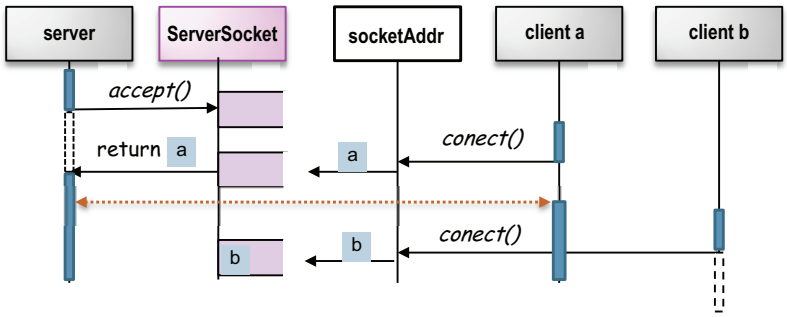


- Manejar la información en secuencias de bytes es complejo, por lo que conviene utilizar otros streams mas especializados que permiten manejar información compleja mediante ***serialización***



Sockets TCP: La clase ServerSocket

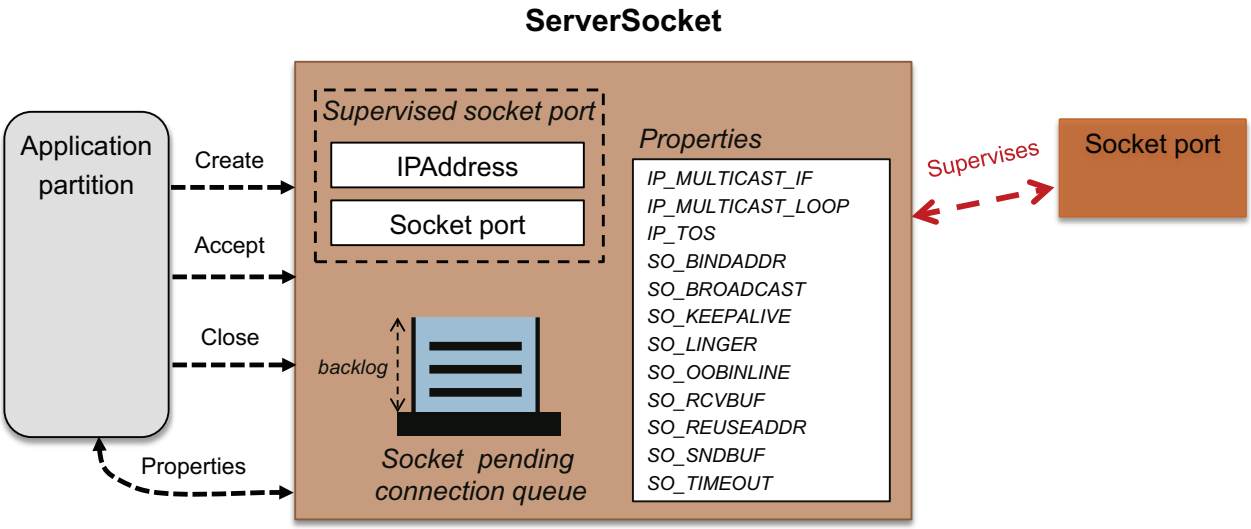
- Permite gestionar la petición de conexiones desde el lado del servidor
  - una instancia de *Socket* requiere un puerto y una dirección IP para la comunicación
  - en el lado servidor no se conoce ni con **quién** ni **cuando** se realizará la comunicación
    - es necesario obtener dicha información para el intercambio de datos
- Proporciona el método *accept* () para atender las peticiones de conexión recibidas en una dirección concreta (@IP, puerto)
  - si hay una petición de conexión pendiente, se atiende la conexión. En caso contrario, el thread invocante queda suspendido hasta que se reciba una petición de conexión



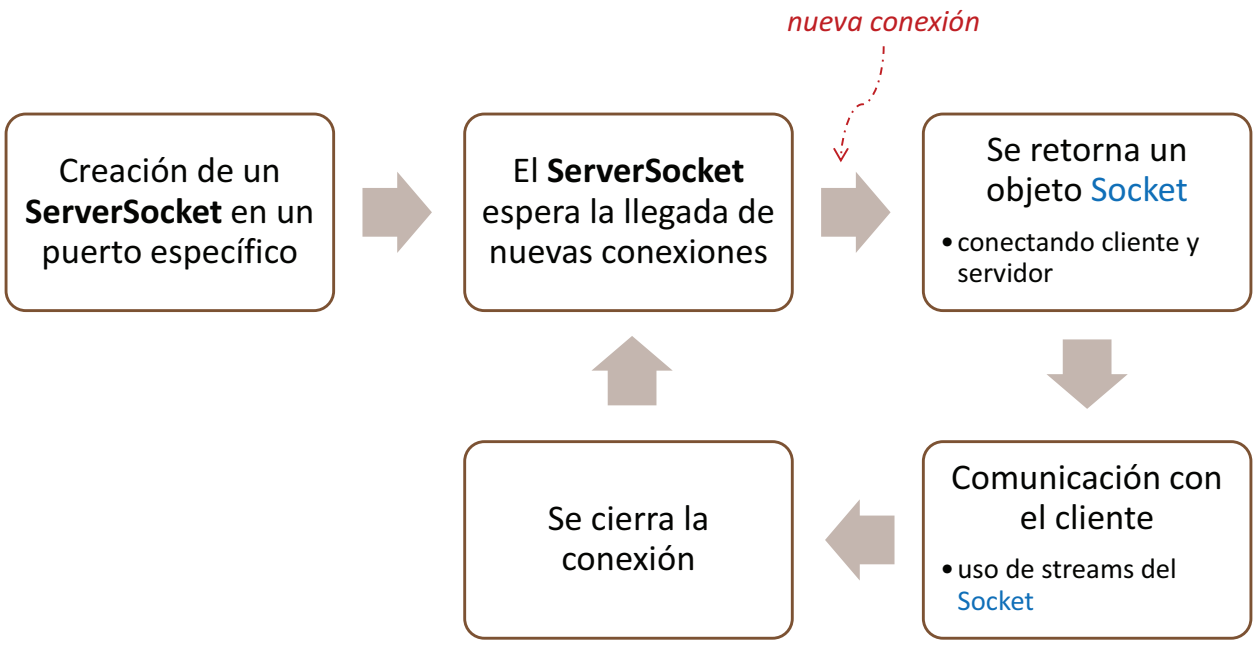
Sockets TCP: Principales métodos de ServerSocket

ServerSocket (**int** port, **int** backlog)	Creates a server socket and binds it to the specified local port number, with the specified backlog ➢ *Backlog es el número máximo de peticiones de conexión que se pueden mantener en cola*
Socket accept()	Listens for a connection to be made to this socket and accepts it
void close()	Closes this socket

Sockets TCP: Elementos de ServerSocket

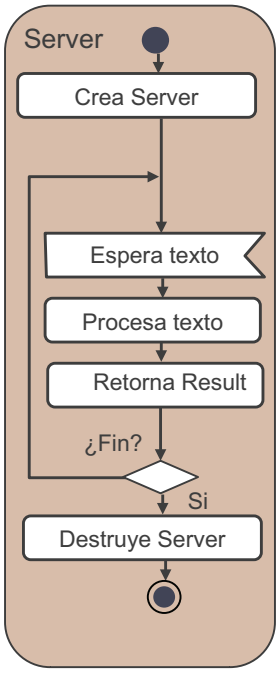


Sockets TCP: Etapas habituales en un servidor



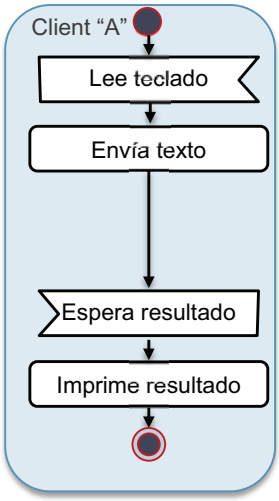
Sockets TCP: Ejemplo de servidor

```
import java.io.*; import java.net.*;
public class Server {
 public static void main(String[] args) {
   Socket elSocket;
   String texto="";
   String result;
   try {
    ServerSocket elServerSocket=new ServerSocket(5000);
    while (!texto.equals("FIN")){          // Repite servicio hasta que texto sea "FIN"
       elSocket=elServerSocket.accept();  // Espera a la conexión de un cliente
       DataInputStream dis=new DataInputStream(elSocket.getInputStream());
       texto=dis.readUTF();               // Lee el texto recibido
       result=texto.toLowerCase();  // Procesa el texto
       DataOutputStream dos=new DataOutputStream(elSocket.getOutputStream());
       dos.writeUTF(result);              // Retorna el resultado al cliente
       elSocket.close();  //Cierra el socket
   }
    elServerSocket.close();                // Cierra el server socket
  } catch (IOException e) {System.err.println("ERROR: En acceso al socket");}
 }
}
```



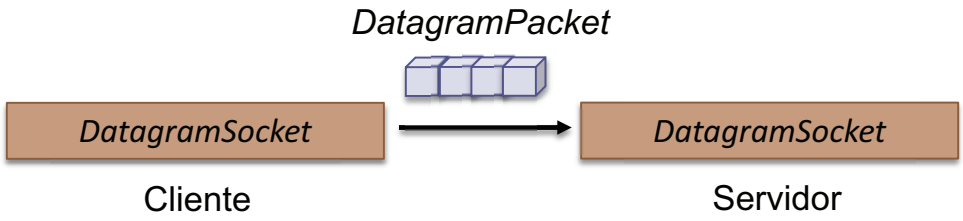
Sockets TCP: Ejemplo de cliente

```
import java.io.*; import java.net.*;
public class Client {
   public static void main(String[] args) {
     Socket elSocket;
     String texto="";
     try {                  // Se lee un texto desde el teclado
       texto = (new BufferedReader(new InputStreamReader(System.in))).readLine();
     } catch (IOException e1) {System.out.println("ERROR con teclado");}
     String result="";      // Se utiliza el servidor para procesar texto
     try{                   // Se crea el socket y se establece la conexión
       elSocket = new Socket(InetAddress.getLocalHost(), 5000);
       DataOutputStream dos = new DataOutputStream(elSocket.getOutputStream());
       dos.writeUTF(texto); // Se envía el texto al servidor
       DataInputStream dis = new DataInputStream(elSocket.getInputStream());
       result=dis.readUTF(); // Se lee el resultado retornado por el servidor
       elSocket.close();       // Se cierra el socket
     } catch (IOException e) {System.out.println("ERROR con socket");}
      // Se imprime el texto antes y después de la transformación
     System.out.println("Texto: "+ texto+"   Result: "+result);
  }
}
```

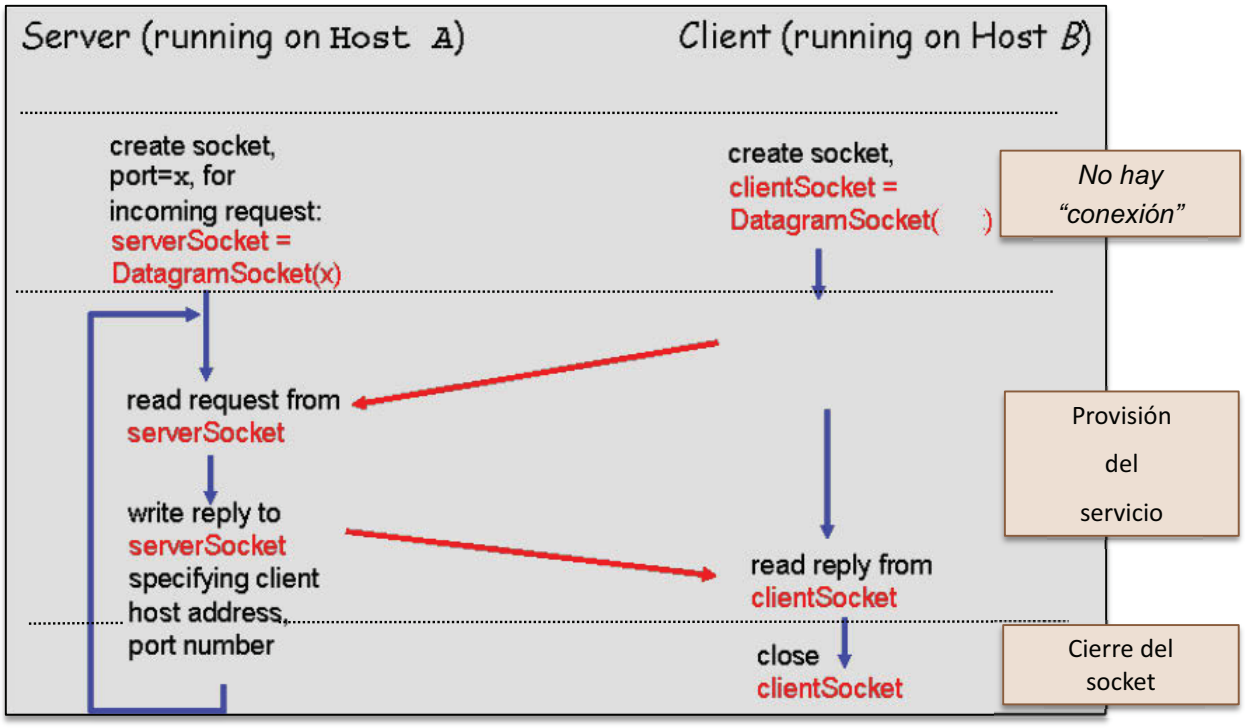


Sockets UDP

- Permiten la transferencia de paquetes de información utilizando el protocolo UDP/IP
  - no requiere que exista una conexión entre sockets previa
- Java gestiona los socket UDP a través de dos clases principales:
  - **DatagramSocket**: para enviar o recibir datagramas.
  - **DatagramPacket**: paquete de datos o datagrama.

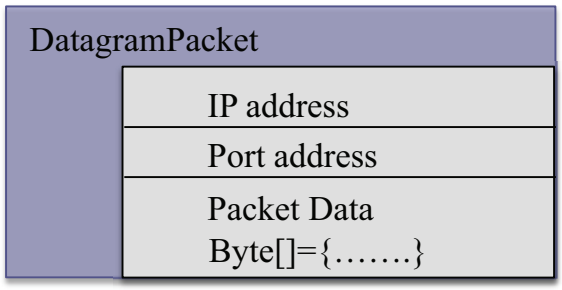


Sockets UDP: Cliente/servidor en Java



Sockets UDP: La clase *DatagramPacket*

- *DatagramPacket* representa un paquete de datos concebido para la transmisión vía UDP
- Los paquetes son contenedores de una pequeña secuencia de bytes que incluye información de direccionamiento (p.ej, una @IP y un puerto)
  - El significado de esa información se determina por su contexto
    - En la recepción, la dirección IP de un socket UDP representa la dirección del remitente
    - En el envío, la dirección IP representa la dirección del destinatario
- Un datagrama consta de:
  - *Cabecera*: dirección de origen/destino del paquete, el puerto, la longitud del paquete, un checksum, etc.
  - *Cuerpo*: datos del paquete.



Sockets UDP: Constructores de *DatagramPacket*

- Para **enviar** datos a una máquina remota usando UDP

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, 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 ➢ *Offset indica la posición absoluta en buf del primer byte transmitido*

- Para **recibir** datos de una máquina remota usando UDP

*DatagramPacket (**byte[]** buf, **int** length)*	Constructs a DatagramPacket for receiving packets of length *length*.
*DatagramPacket (**byte[]** buf, **int** offset, **int** length)*	Constructs a DatagramPacket for receiving packets of length *length*, specifying an offset into the buffer. ➢ *Offset indica la posición absoluta en buf del primer byte recibido*

Sockets UDP: Principales métodos de *DatagramPacket*

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.
SocketAddress getSocketAddress()	Gets the SocketAddress (e.g., 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 of the remote host to which this datagram is being sent.

Sockets UDP: La clase *DatagramSocket*

- Proporciona acceso a los sockets UDP, que permiten enviar y recibir *DatagramPacket*
  - un mismo socket puede ser usado para recibir y enviar paquetes
  - las operaciones de lectura son **bloqueantes** (el thread se suspende hasta que el paquete llegue)
- Constructores:

DatagramSocket()	Constructs a datagram socket and binds it to *any* available port on the local host machine.
DatagramSocket(**int** port)	Constructs a datagram socket and binds it to the specified port on the *wildcard* address.
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.
protected DatagramSocket (DatagramSocketImpl impl)	Creates an *unbound* datagram socket with the specified DatagramSocketImpl.

Sockets UDP: Principales métodos *DatagramSocket*

void receive(DatagramPacket p)	Receives a datagram packet from this socket.
void send(DatagramPacket p)	Sends a datagram packet from this socket.
void close()	Closes this datagram socket.

Sockets UDP: Ejemplo de servidor

```
import java.io.*;
import java.net.*;
public class ServerUDP {
   public static void main(String[] args) {
      byte[] buff=new byte[256];
      String texto="";
      String result;
      try {                                          //Se crea un datagram socket asociado al puerto 5000 local
        DatagramSocket elDatagramSocket= new DatagramSocket(5000,InetAddress.getLocalHost());

        do {                                         // Se crean un datagram packet con capacidad de hasta 256 bytes
         DatagramPacket elPaquete = new DatagramPacket(buff,256);
         elDatagramSocket.receive(elPaquete);        // Espera el paquete procedente de un cliente con el texto
         texto=new String(elPaquete.getData(),0,elPaquete.getLength());
         result=texto.toLowerCase();                 // Procesa el texto recibido (conversión a minúsculas)
         elPaquete.setData(result.getBytes());       // Incluye el resultado en el paquete y lo retorna al cliente
         elDatagramSocket.send(elPaquete);
        } while (!texto.equals("FIN"));              // Repite el servicio hasta que texto sea "FIN"
        elDatagramSocket.close();                    // Cierra el datagram socket
      } catch (IOException e) {System.err.println("ERROR: En acceso al socket");}
   }
}
```

RCSD: Laura Barros, José M. Drake y Héctor Pérez

Sockets UDP: Ejemplo de cliente

```
import java.io.*;
import java.net.*;
public class ClientUDP {
 public static void main(String[] args) {
   String texto="";
   String result="";
   try {                                                    // Se lee un texto desde el teclado
       texto = (new BufferedReader(new InputStreamReader(System.in))).readLine();
    } catch (IOException e1) {System.out.println("ERROR con teclado");}
   try{                                                     // Se utiliza el servidor para procesar texto
       DatagramSocket elDatagramSocket = new DatagramSocket(); // Se crea el socket y después el paquete datagram
       byte[] buff=texto.getBytes();
       DatagramPacket elPaquete=new DatagramPacket(buff,buff.length,InetAddress.getLocalHost(),5000);
       elDatagramSocket.send(elPaquete);                    // Se envía el paquete con el texto al servidor
       elDatagramSocket.receive(elPaquete);                 // Se espera a que el servidor retorne el resultado
       result=new String(elPaquete.getData());

       elDatagramSocket.close();                            // Se cierra el socket
    } catch (IOException e){System.out.println("ERROR con socket");}

   System.out.println("Texto: "+ texto+"   Result: "+result);          // Se imprime antes y después de la transformación
  }
}
```