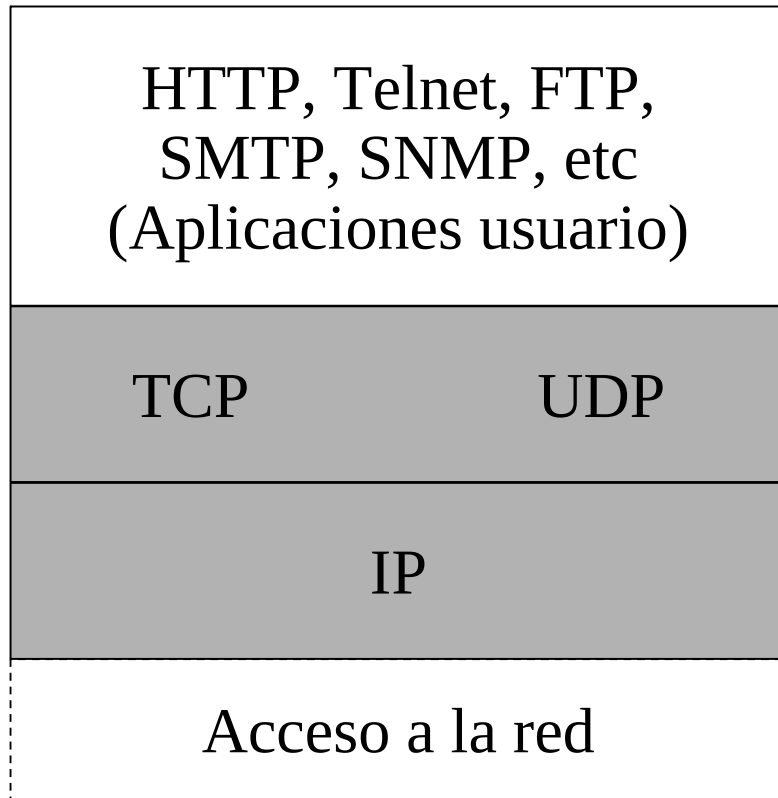


Redes en Java

TCP/IP



- TCP: Transmission Control Protocol
- UDP: User Datagram Protocol
- IP: Internet Protocol

TCP

- Protocolo orientado a conexión que maneja un canal de comunicación confiable entre dos computadoras
- Garantiza que los mensajes llegarán al destino en forma correcta y ordenada
- Verificación de errores, reconocimientos (ACK), retransmisión, secuenciación de paquetes
- HTTP, FTP, Telnet

UDP

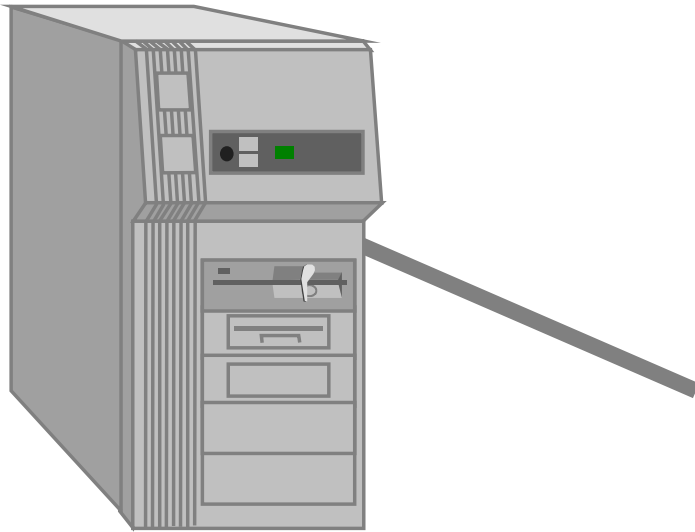
- Protocolo orientado a no-conexión que envía paquetes de datos independientes (datagramas), sin garantía de la llegada o del orden de los paquetes
- No incurre en el gasto extra de mantener una conexión y hacer la verificación de cada paquete por medio de *ACKs* y *timeouts*
- Diseñado para aplicaciones donde la pérdida parcial de datos no es importante

Direcciones IP y puertos

- Cuando enviamos información en Internet necesitamos identificar la máquina destino y el proceso que queremos maneje los datos enviados
- Una dirección IP identifica una máquina en Internet, son de 32 bits: **131.178.14.14**, **192.9.48.9**
- Puertos identifican a un proceso dentro de una máquina son números de 16 bits

Direcciones IP y puertos

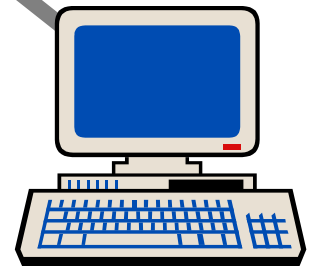
192.9.48.9



<u>Aplicación</u>	<u>Puerto</u>
Servidor WEB	80
Servidor FTP	21
:	:
Miprograma	5999
:	:

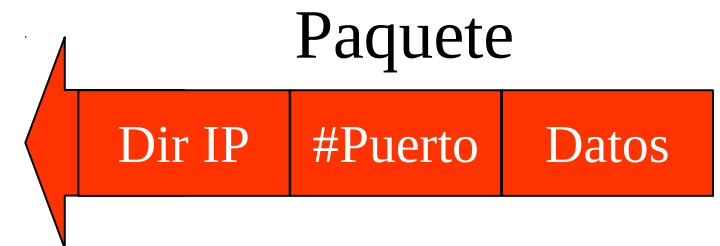
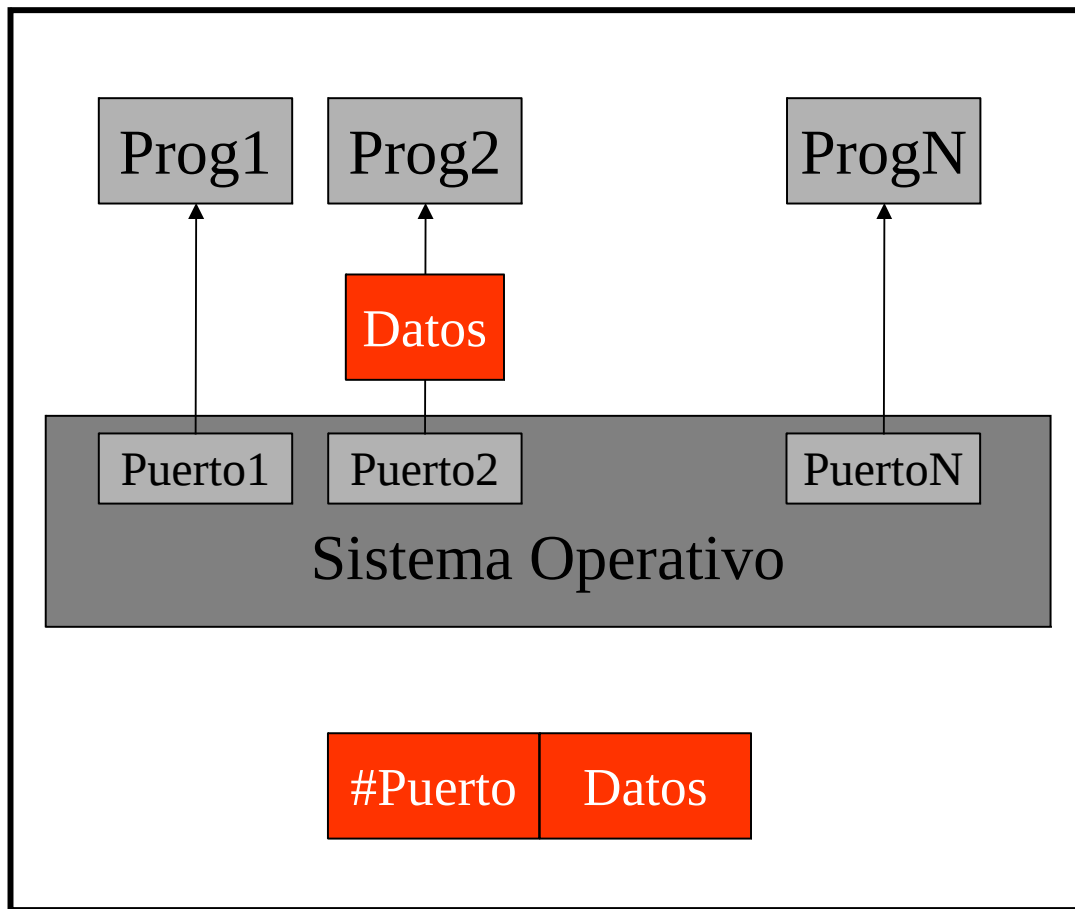
Paquete

Dir IP, puerto, datos



131.178.14.14

Direcciones IP y puertos

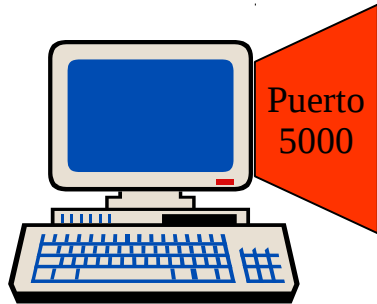


Sockets

- Un socket es un extremo de un enlace de comunicación bidireccional
- A un socket se le asocia un número de puerto, para que el sistema identifique a que aplicación mandar algún paquete
- También se le asocia una dirección IP para que un paquete mandado por el socket pueda viajar por la red hacia un destino
- Las aplicaciones escriben y leen de sockets

131.178.2.25

Socket



Sockets

Socket

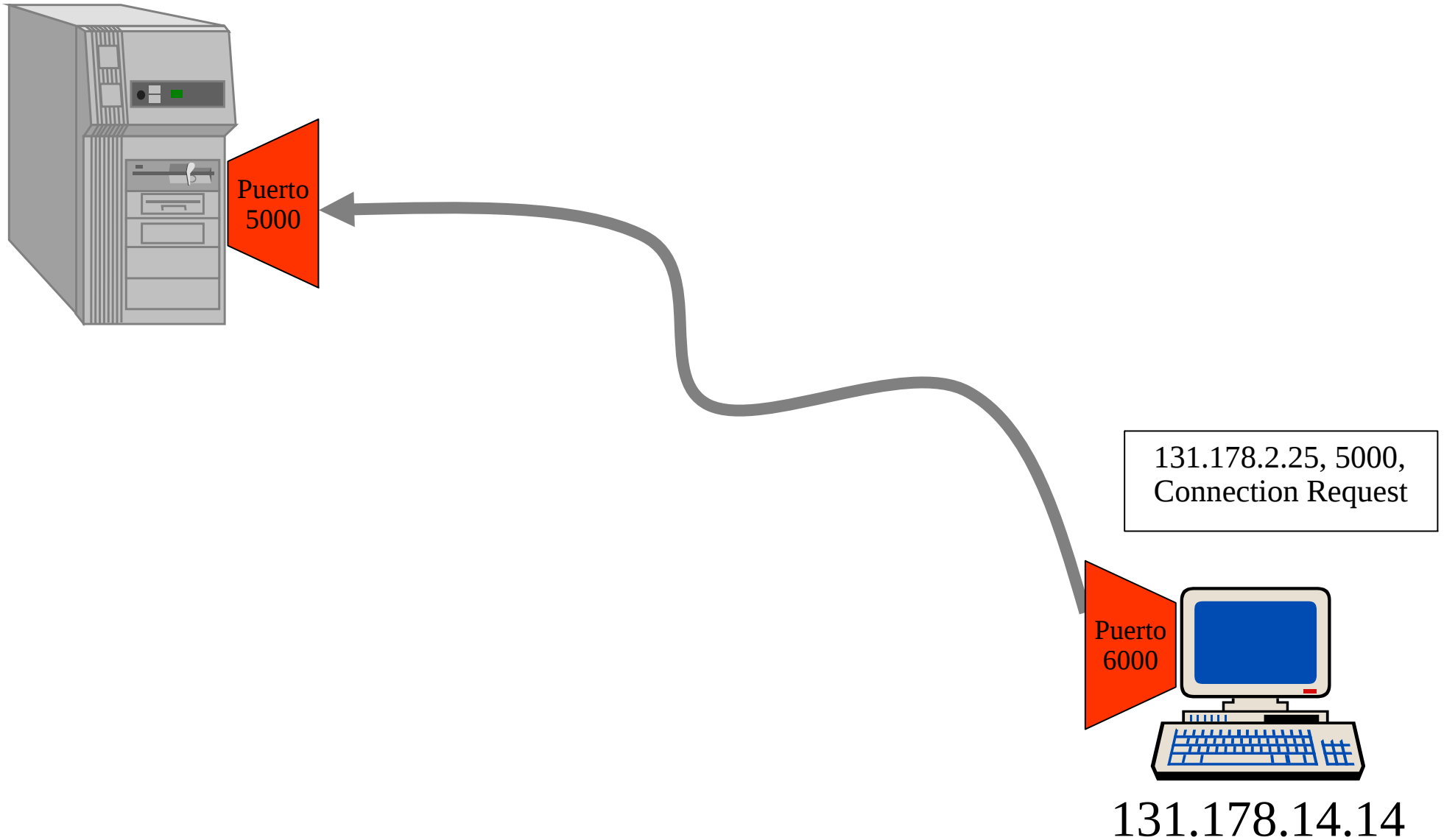


131.178.2.25, 5000, "Hola"

131.178.14.14

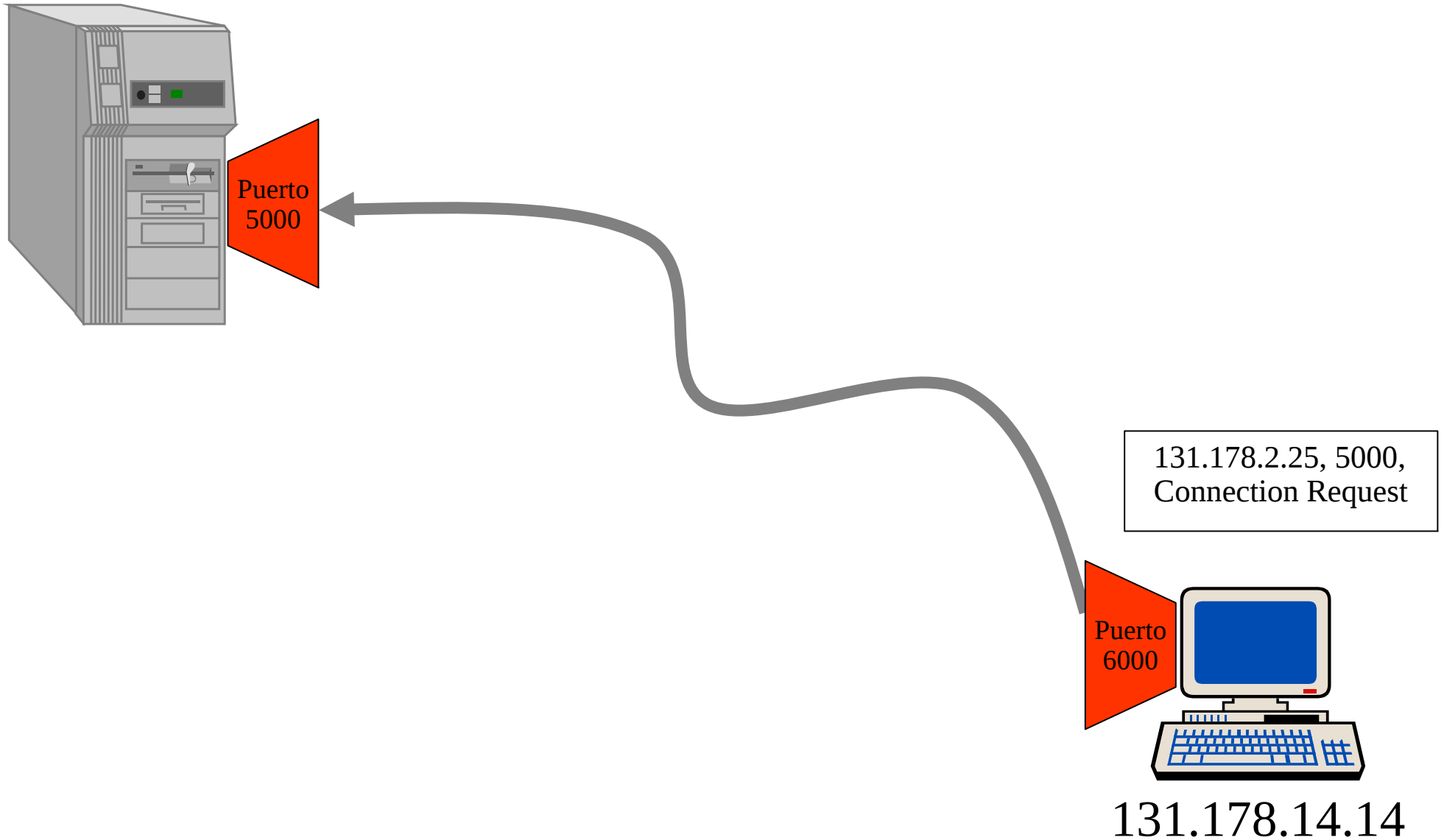
Petición de conexión

131.178.2.25



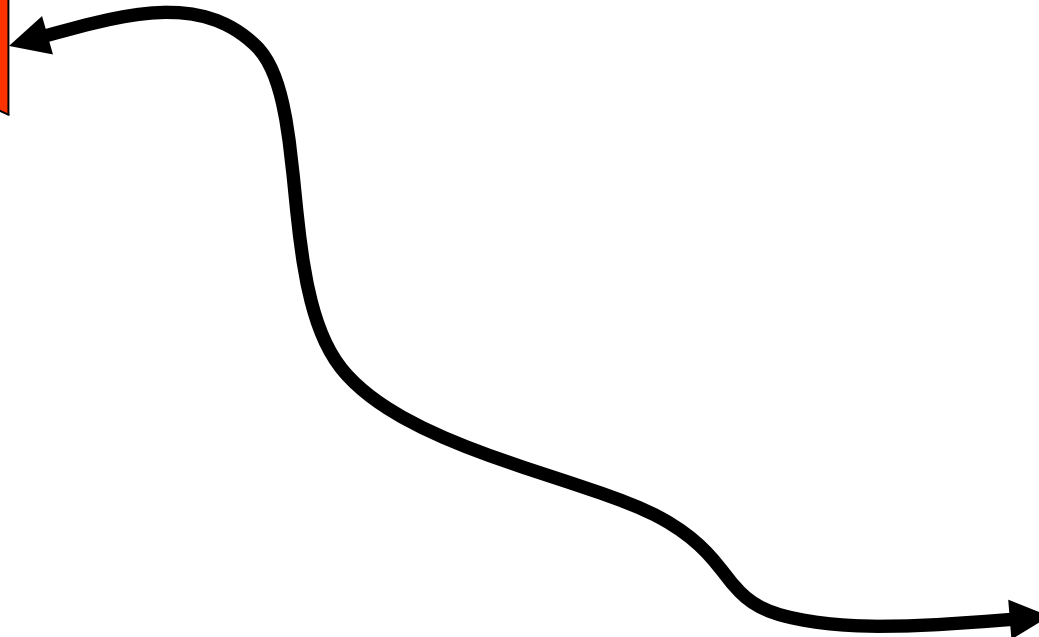
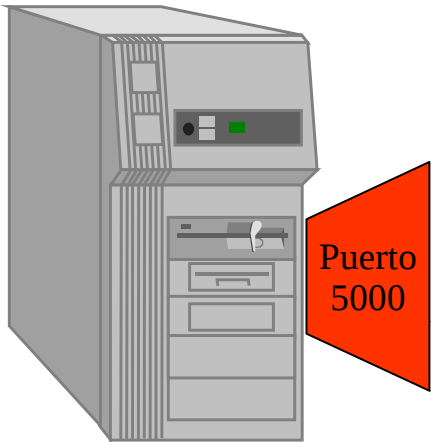
Petición de conexión

131.178.2.25



Establecimiento de la conexión

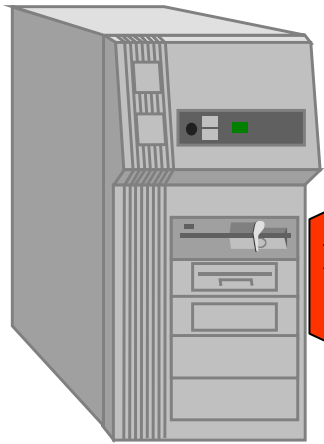
131.178.2.25



131.178.14.14

Establecimiento de la conexión

131.178.2.25



Puerto
5000



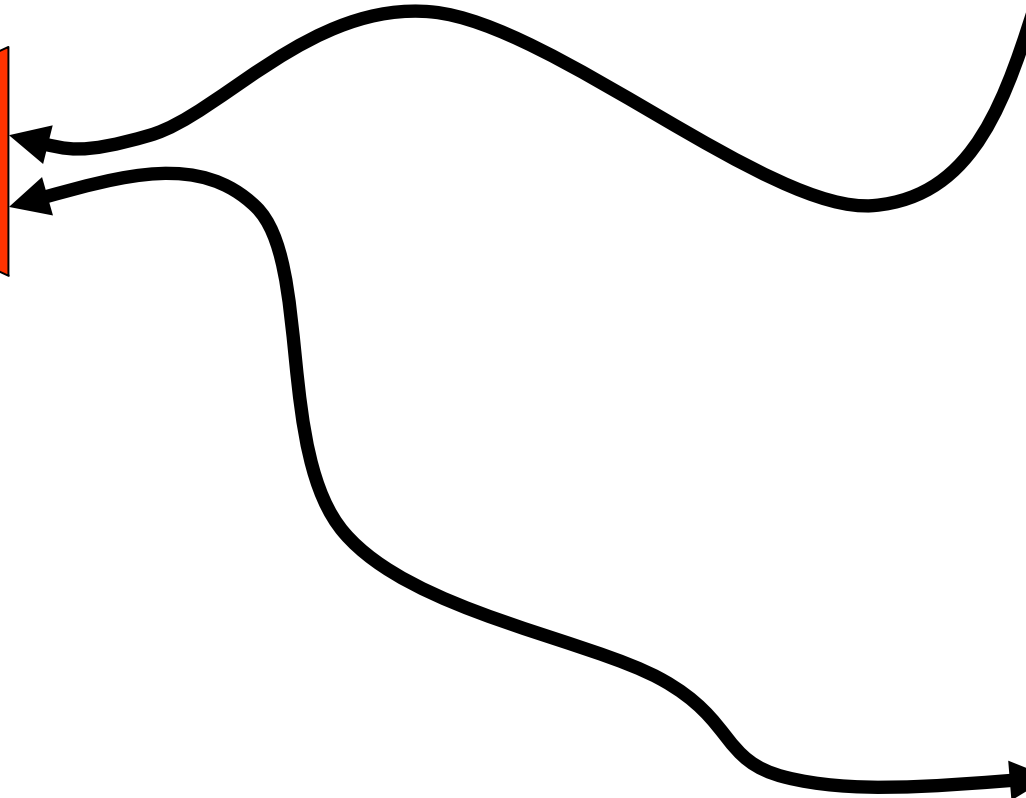
Puerto
7867

55.78.34.56



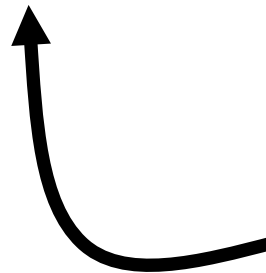
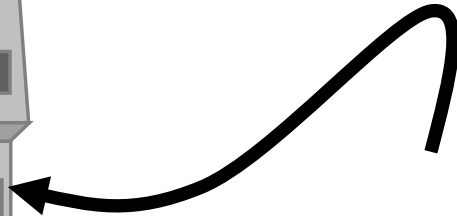
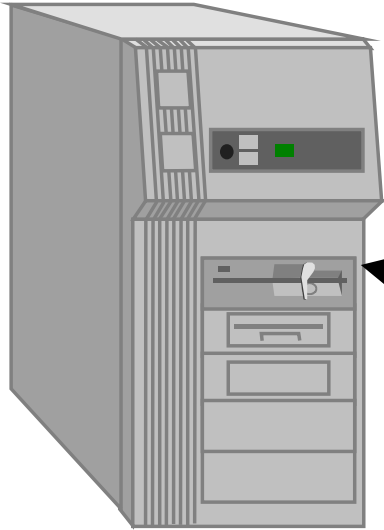
Puerto
6000

131.178.14.14

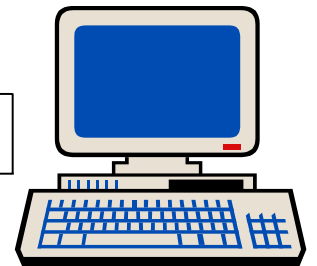


UDP

192.9.48.9



34.56.89.54, 5000, "Hola"



131.178.14.14

La clase InetAddress

- Representa un dirección de IP
- No tiene constructor
- Para crear un ejemplar de esta clase es necesario llamar a los métodos:
 - `getLocalHost()`
 - `getByName(String)`
 - `getAllByName(String)`

Métodos de InetAddress

- **public static InetAddress getLocalHost()**
 - Regresa la dirección IP de la máquina local
- **public static InetAddress
getByName(String host)**
 - Regresa la dirección IP de la máquina indicada en el parámetro, puedes ser el nombre (“javax.mty.itesm.mx”) o la dirección especificada como string (“131.178.14.228”)
- **public static InetAddress[]
getAllByName(String host)**
 - Regresa todas las direcciones IP asignadas a la máquina indicada como parámetro

Métodos de InetAddress

- **public String getHostName()**
 - Regresa el nombre de la máquina con esta dirección de IP
- **public String getHostAddress()**
 - Regresa la dirección de IP en formato tradicional (131.178.14.14)
- **public byte[] getAddress()**
 - Regresa la dirección de IP como un arreglo de bytes. El byte más significativo de la dirección está en el subíndice 0

Ejemplo

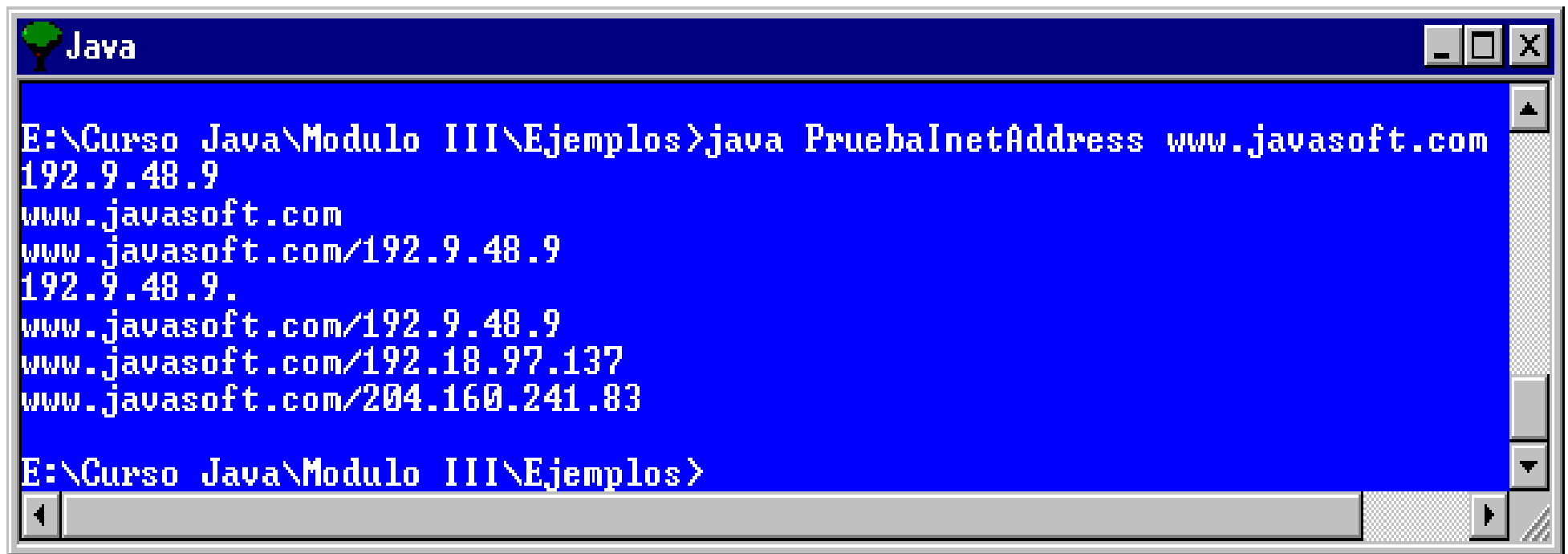
```
import java.net.*;

public class PruebaInetAddress{

    public static void main(String args[]){
        String argumento;
        InetAddress direccion=null;
        try{
            argumento = args[0];
        }catch (ArrayIndexOutOfBoundsException e){
            argumento = null;
        }
        try{
            if (argumento!=null)
                direccion = InetAddress.getByName(argumento);
            else
                direccion = InetAddress.getLocalHost();
        }catch (UnknownHostException e){
            System.out.println(e.getMessage());
            System.exit(0);
        }
    }
}
```

```
System.out.println(direccion.getHostAddress());
System.out.println(direccion.getHostName());
System.out.println(direccion.toString());
byte[] dir = direccion.getAddress();
for (int i=0;i<dir.length;i++){
    int b = dir[i] < 0 ? dir[i]+256 : dir[i];
    System.out.print(b + ".");
}
System.out.println();
try{
    InetAddress[] direcciones = InetAddress.getAllByName(direccion.getHostName());
    for (int i=0;i<direcciones.length;i++)
        System.out.println(direcciones[i].toString());
}catch (UnknownHostException e){}
```

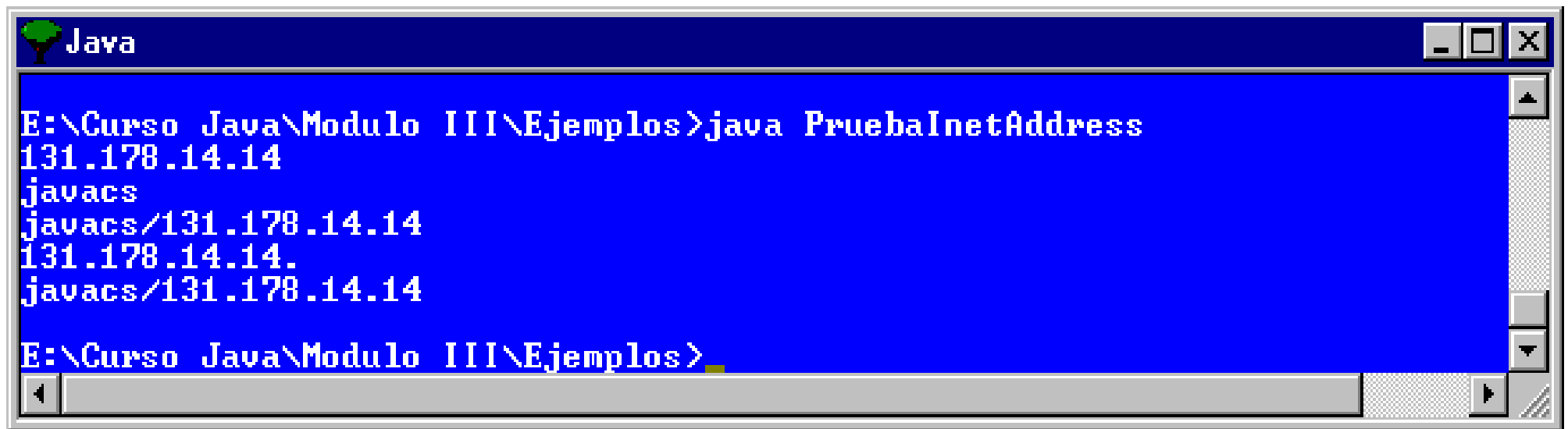
Ejemplo



```
E:\Curso Java\Modulo III\Ejemplos>java PruebaInetAddress www.javasoft.com
192.9.48.9
www.javasoft.com
www.javasoft.com/192.9.48.9
192.9.48.9.
www.javasoft.com/192.9.48.9
www.javasoft.com/192.18.97.137
www.javasoft.com/204.160.241.83

E:\Curso Java\Modulo III\Ejemplos>
```

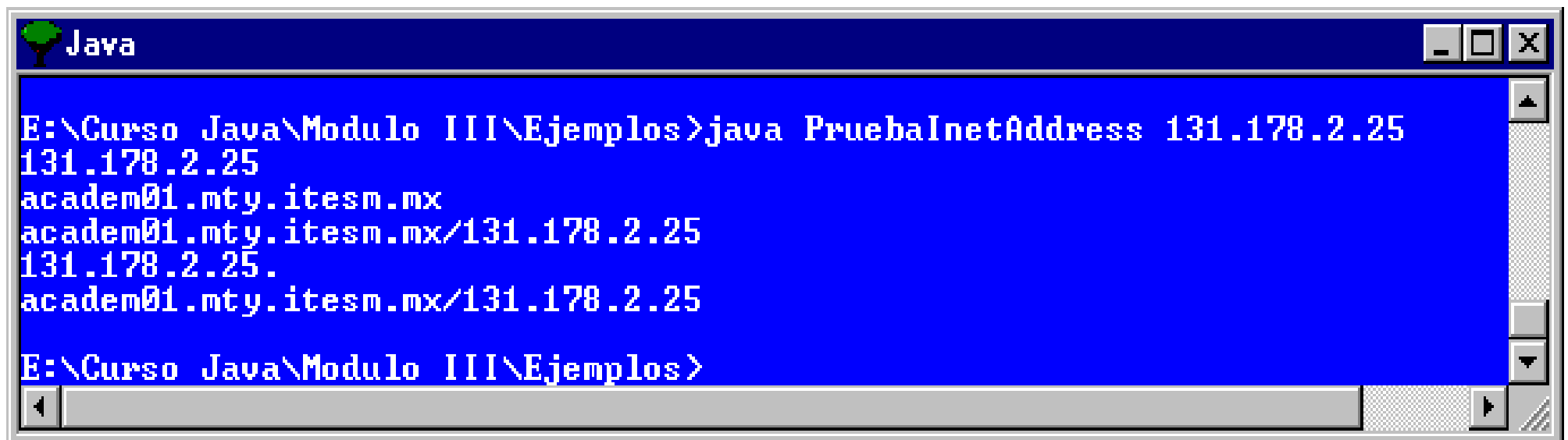
Ejemplo



```
Java

E:\Curso Java\Modulo III\Ejemplos>java PruebaInetAddress
131.178.14.14
javacs
javacs/131.178.14.14
131.178.14.14.
javacs/131.178.14.14

E:\Curso Java\Modulo III\Ejemplos>
```



```
Java

E:\Curso Java\Modulo III\Ejemplos>java PruebaInetAddress 131.178.2.25
131.178.2.25
academ01.mty.itesm.mx
academ01.mty.itesm.mx/131.178.2.25
131.178.2.25.
academ01.mty.itesm.mx/131.178.2.25

E:\Curso Java\Modulo III\Ejemplos>
```

La clases de *sockets*

- Existen tres clases de *sockets* en Java
 - `Socket`: *sockets* TCP (*streams*)
 - `ServerSocket`: para escuchar por conexiones
 - `DatagramSocket`: *sockets* UDP (*datagramas*)
- La decisión de utilizar `Socket` o `DatagramSocket` depende de la aplicación específica
- Cuando manejas *datagramas* no necesitas escuchar por peticiones de conexión

La clase Socket

- `Socket(String host,int port)`
 - Crea un *socket* TCP y lo conecta a la máquina especificada por `host` en el puerto `port`
 - Arroja `UnknownHostException`, `IOException`
- `Socket(InetAddress address,int port)`
 - Crea un *socket* TCP y lo conecta a la máquina con dirección IP `address` en el puerto `port`
 - Arroja `IOException`

La clase Socket

- `Socket(String host,int port, InetAddress localAddr,int localPort)`
 - Crea un *socket* TCP conectado a la máquina `host` en el puerto `port`, el *socket* queda enlazado a la máquina local en la dirección `localAddr` al puerto local `localPort`
- `Socket(InetAddress address,int port, InetAddress localAddr,int localPort)`
 - Crea un *socket* TCP conectado a la máquina con dirección IP `address` al puerto `port`, el *socket* queda enlazado a la máquina local en la dirección `localAddr` al puerto local `localPort`

Métodos de Socket

- `InetAddress getAddress()`
 - Regresa la dirección remota a la cual está conectada el *socket*
- `InetAddress getLocalAddress()`
 - Regresa la dirección local a la cual el *socket* está enlazado
- `int getPort()`
 - Regresa el puerto remoto al cual está conectado el *socket*
- `int getLocalPort()`
 - Regresa el puerto local al cual está enlazado el *socket*

Métodos de Socket

- `close()`
 - Cierra el *socket*
- `InputStream getInputStream()`
 - Regresa el flujo de entrada de este *socket*
- `OutputStream getOutputStream()`
 - Regresa el flujo de salida de este *socket*

La clase `ServerSocket`

- `ServerSocket(int port)`
 - Crea un *socket* servidor en el puerto especificado, `port = 0` crea un *socket* en cualquier puerto libre
 - La fila de espera para peticiones de conexiones se establece en 50
- `ServerSocket(int port, int backlog)`
 - Crea un *socket* servidor en el puerto especificado, `port = 0` crea un *socket* en cualquier puerto libre
 - La fila de espera para peticiones de conexiones se establece en `backlog`

La clase `ServerSocket`

- `ServerSocket(int port, int backlog, InetAddress bindAddr)`
 - Crea un socket servidor en el puerto `port`, con una fila para espera de conexiones de tamaño `backlog` y con la dirección IP `bindAddr`
 - Este constructor se utiliza en el caso de que se tengan servidores con múltiples direcciones IP asignadas

Métodos de la clase `ServerSocket`

- `Socket accept()`
 - Escucha por conexiones a este socket, el método bloquea la ejecución del programa hasta que se realiza una conexión
- `InetAddress getInetAddress()`
 - Regresa la dirección local a la cual está conectada el *socket*
- `int getLocalPort()`
 - Regresa el puerto local al cual está enlazado el *socket*
- `close()`
 - Cierra el *socket*

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

public class Servidor1{
    public static void main(String[] args){
        ServerSocket yo = null;
        Socket cliente = null;
        try{
            yo = new ServerSocket(5000);
        } catch (IOException e){
            System.out.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Socket escuchando en puerto 5000");
        try{
            cliente = yo.accept();
        } catch (IOException e){
            System.out.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Ya se conecto el cliente");
        try{
            cliente.close();
            yo.close();
        } catch (IOException e){
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Ejemplo

Ejemplo

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

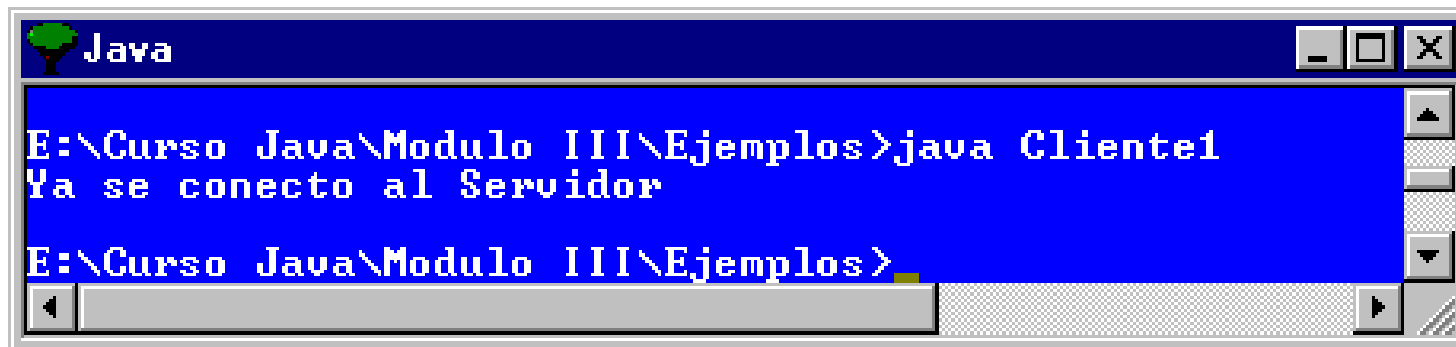
public class Cliente1{
    public static void main(String[] args){
        Socket yo = null;
        try {
            yo = new Socket("131.178.XXX.XXX", 5000);
        } catch (UnknownHostException e){
            System.out.println(e.getMessage());
            System.exit(1);
        }
        catch (IOException e){
            System.out.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Ya se conecto al Servidor");
        try {
            yo.close();
        } catch (IOException e){
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Ejemplo



```
Java - java Servidor1
E:\Curso Java\Modulo III\Ejemplos>java Servidor1
Socket escuchando en puerto 5000
```

Servidor



```
Java
E:\Curso Java\Modulo III\Ejemplos>java Cliente1
Ya se conecto al Servidor
E:\Curso Java\Modulo III\Ejemplos>
```

Cliente



```
Java
E:\Curso Java\Modulo III\Ejemplos>java Servidor1
Socket escuchando en puerto 5000
Ya se conecto el cliente
E:\Curso Java\Modulo III\Ejemplos>
```

Servidor

Cliente/Servidor

- Normalmente queremos algo más que conectarnos a un servidor
- El servidor nos va a dar un servicio
- la comunicación entre el cliente y el servidor, después de que se establece la conexión, tiene que respetar un protocolo. Tanto en el orden de la transmisión de los datos como en el tipo de datos que uno transmite y el otro espera recibir, o vamos a tener serios problemas.

Esquema Servidor

- Abre un socket para esperar por conexiones
- Al llegar una conexión crea otro socket para comunicarse con el cliente
- Asocia uno o más flujos al socket de comunicación
- Lee/escribe a los flujos de acuerdo al protocolo establecido
- Termina la comunicación
- Cierra los flujos y cierra los sockets

Esquema Cliente

- Abre un socket para conectarse y comunicarse con el servidor
- Establece conexión con el servidor
- Asocia uno o más flujos al socket de comunicación
- Lee/escribe al flujo de acuerdo al protocolo establecido
- Termina la comunicación
- Cierra los flujos y cierra los sockets

Ejemplo Cliente/Servidor

- En el siguiente ejemplo el cliente se conecta al servidor y le comienza a mandar una serie de strings, y el servidor le responde con la longitud de cada string (como un entero!)
- Ambos programas terminan cuando el cliente le envía al servidor un string nulo
- El cliente debe de teclear la dirección o nombre del servidor en la línea de comandos

Ejemplo Servidor2.java

```
ServerSocket yo = null;
Socket cliente = null;
BufferedReader entrada;
DataOutputStream salida;
String llego;

    yo = new ServerSocket(5000);

    cliente = yo.accept();

    entrada = new BufferedReader
                (new InputStreamReader(cliente.getInputStream()));
    salida = new DataOutputStream(cliente.getOutputStream());

    do{
        llego = entrada.readLine();
        System.out.println("Llego: " + llego);
        salida.writeInt(llego.length());
    }while(llego.length()!=0);

    entrada.close();
    cliente.close();
    yo.close();
```

Ejemplo Cliente2.java

```
Socket yo = null;
PrintWriter alServidor;
BufferedReader delTeclado;
DataInputStream delServidor;
String tecleado;

    yo = new Socket(comandos[0],5000);

    delTeclado  = new BufferedReader(new InputStreamReader(System.in));
    alServidor  = new PrintWriter(yo.getOutputStream(),true);
    delServidor = new DataInputStream(yo.getInputStream());

    do{
        System.out.print("Teclea un string: ");
        tecleado = delTeclado.readLine();
        alServidor.println(tecleado);
        System.out.println("Longitud = "+delServidor.readInt());
    }while(tecleado.length()!=0);

    delServidor.close();
    delTeclado.close();
    alServidor.close();
    yo.close();
```

Ejemplo

```
Java
E:\Curso Java\Modulo III\Ejemplos>java Servidor2
Socket escuchando en puerto 5000
Ya se conecto el cliente
Llego: Curso de Java
Llego: Módulo III
Llego: Pronto terminará
Llego:
Ya termine de recibir

E:\Curso Java\Modulo III\Ejemplos>
```

```
Java
E:\Curso Java\Modulo III\Ejemplos>java Cliente2 javacs
Ya se conecto al Servidor
Teclea un string: Curso de Java
Longitud = 13
Teclea un string: Módulo III
Longitud = 10
Teclea un string: Pronto terminará
Longitud = 16
Teclea un string:
Longitud = 0
Ya termine de enviar

E:\Curso Java\Modulo III\Ejemplos>
```

Aplicaciones Clientes/Servidor

- Aunque a veces es suficiente tener un servidor para que atienda a UN cliente, normalmente no es el caso
- Necesitamos que el servidor pueda atender a múltiples clientes en forma simultánea
- Para eso basta arrancar un hilo para atender a cada cliente
- En el hilo principal el servidor siempre estará escuchando por peticiones de conexión

Crear ServerSocket

Petición de
Conexión?

SI

Atender conexión
Con el socket creado

Atención de
la conexión
en un HILO

Atención de
la conexión
en un HILO

Atención de
la conexión
en un HILO

Atención de
la conexión
en un HILO

Atención de
la conexión
en un HILO

Ejemplo ServidorM2.java

```
ServerSocket yo = null;
Socket cliente = null;
boolean escuchando = true;

    yo = new ServerSocket(5000);

    while(escuchando){
        cliente = yo.accept();
        System.out.println("Ya se conecto un cliente desde: " +
                           cliente.getInetAddress().getHostName() +
                           "("+cliente.getPort()+")");
        new Atiende(cliente).start();
    }

yo.close();
```

Ejemplo ServidorM2.java

```
class Atiende extends Thread{

    public Atiende(Socket cliente){
        this.cliente = cliente;
        nombreyDirIP = this.cliente.getInetAddress().toString();
    }

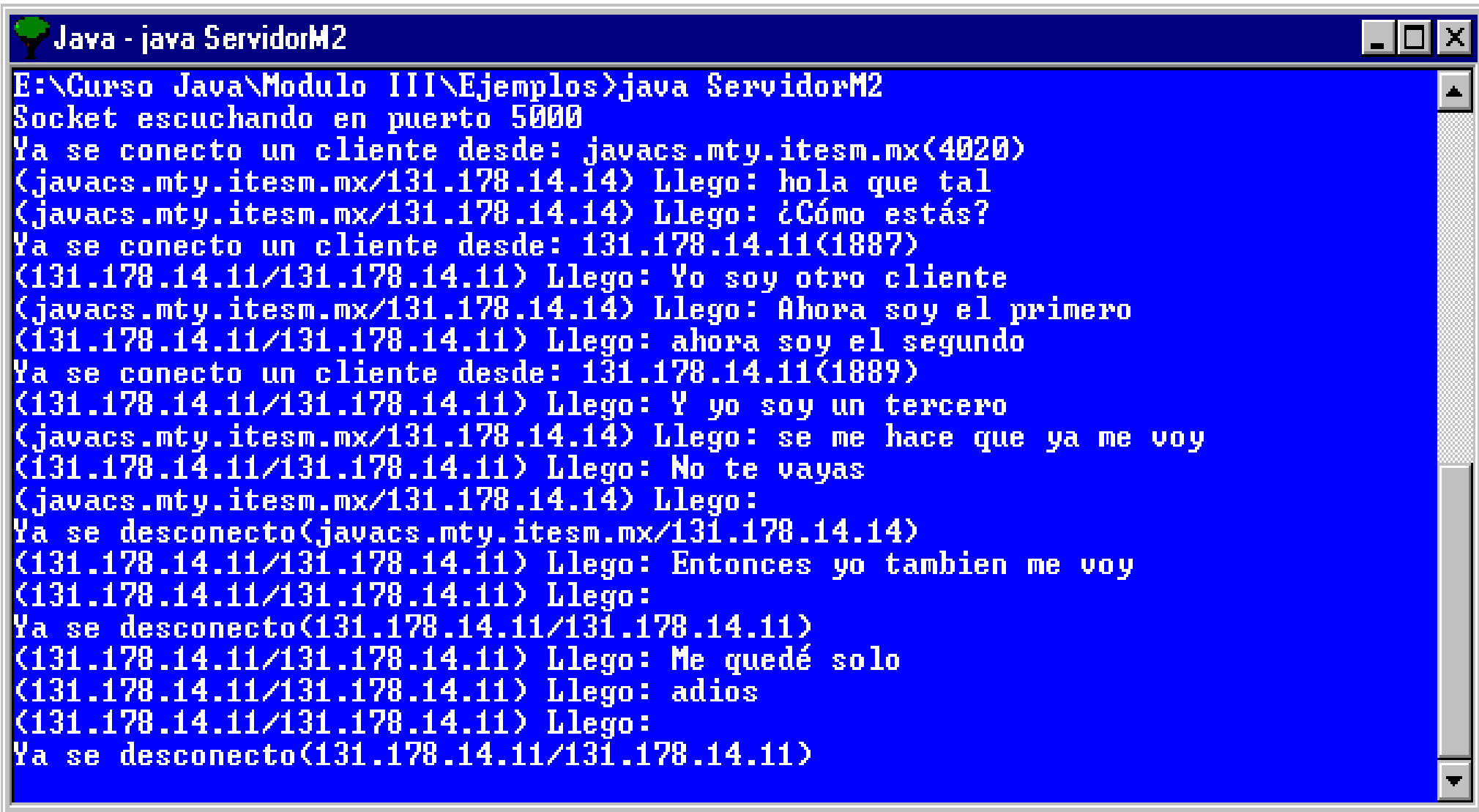
    public void run(){
        entrada = new BufferedReader
            (new InputStreamReader(cliente.getInputStream()));
        salida = new DataOutputStream(cliente.getOutputStream());

        do{
            llego = entrada.readLine();
            System.out.println("(" + nombreyDirIP + ") Llego: " + llego);
            salida.writeInt(llego.length());
        }while(llego.length() != 0);

        entrada.close();
        cliente.close();

        System.out.println("Ya se desconecto" + "(" + nombreyDirIP + ")");
    }
}
```

Ejemplo ServidorM2.java



```
Java - java ServidorM2
E:\Curso Java\Modulo III\Ejemplos>java ServidorM2
Socket escuchando en puerto 5000
Ya se conecto un cliente desde: javacs.mty.itesm.mx(4020)
(javacs.mty.itesm.mx/131.178.14.14) Llego: hola que tal
(javacs.mty.itesm.mx/131.178.14.14) Llego: ¿Cómo estás?
Ya se conecto un cliente desde: 131.178.14.11(1887)
(131.178.14.11/131.178.14.11) Llego: Yo soy otro cliente
(javacs.mty.itesm.mx/131.178.14.14) Llego: Ahora soy el primero
(131.178.14.11/131.178.14.11) Llego: ahora soy el segundo
Ya se conecto un cliente desde: 131.178.14.11(1889)
(131.178.14.11/131.178.14.11) Llego: Y yo soy un tercero
(javacs.mty.itesm.mx/131.178.14.14) Llego: se me hace que ya me voy
(131.178.14.11/131.178.14.11) Llego: No te vayas
(javacs.mty.itesm.mx/131.178.14.14) Llego:
Ya se desconecto(javacs.mty.itesm.mx/131.178.14.14)
(131.178.14.11/131.178.14.11) Llego: Entonces yo tambien me voy
(131.178.14.11/131.178.14.11) Llego:
Ya se desconecto(131.178.14.11/131.178.14.11)
(131.178.14.11/131.178.14.11) Llego: Me quedé solo
(131.178.14.11/131.178.14.11) Llego: adios
(131.178.14.11/131.178.14.11) Llego:
Ya se desconecto(131.178.14.11/131.178.14.11)
```

Write Once and Run Anywhere

Será verdad?

Datagramas

- Un socket de tipo datagrama es un punto de enlace orientado a no conexión
- El enlace no es confiable. No existe seguridad de que los paquetes llegarán ni en que orden lo harán
- Cada paquete enviado o recibido es direccionado y ruteado en forma individual
- Los paquetes enviados por una misma máquina puede seguir rutas diferentes

La classe DatagramSocket

Constructor Summary

[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 local host machine.

[DatagramSocket](#)(int port, [InetAddress](#) laddr)

Creates a datagram socket, bound to the specified local address.

Métodos de DatagramSocket

- Además de `close()`, `getLocalPort()` y `getLocalAddress()`, que funcionan igual que los métodos en la clase `Socket`, existen:
- **`void receive(DatagramPacket p)`**
 - Para recibir paquetes (datagramas)
 - El paquete recibido queda en `p`
- **`void send(DatagramPacket p)`**
 - Para enviar paquetes (datagramas)
 - El paquete enviado es el que estaba en `p`

DatagramPacket

Constructor Index

- [DatagramPacket](#)(byte[], int)
Constructs a DatagramPacket for receiving packets of length i length.
- [DatagramPacket](#)(byte[], int, InetAddress, int)
Constructs a datagram packet for sending packets of length i length to the specified port number on the specified host.

Method Index

- [getAddress](#)()
Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
- [getData](#)()
Returns the data received or the data to be sent.
- [getLength](#)()
Returns the length of the data to be sent or the length of the data received.
- [getPort](#)()
Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.
- [setAddress](#)(InetAddress)
- [setData](#)(byte[])
- [setLength](#)(int)
- [setPort](#)(int)

byte[]



Ejemplo ServidorD1

```
DatagramSocket yo = null;
DatagramPacket paquete;
InetAddress dirCliente = null;
int puertoCliente;
byte[] buffer = new byte[80];
String aMandar, recibido;

yo = new DatagramSocket(5000);
System.out.println("Socket escuchando en puerto 5000");

while(true){
    buffer = new byte[80];
    paquete = new DatagramPacket(buffer, buffer.length);
    yo.receive(paquete);
    recibido = new String(paquete.getData());
    dirCliente = paquete.getAddress();
    puertoCliente = paquete.getPort();
    System.out.println(dirCliente.toString()+" me mandó >"+recibido);
    aMandar = new String(recibido.toUpperCase());
    buffer = new byte[80];
    buffer = aMandar.getBytes();
    paquete = new DatagramPacket(buffer,buffer.length, dirCliente, puertoCliente);
    yo.send(paquete);
}
//yo.close();
```

Ejemplo ClienteD1

```
DatagramSocket yo = null;
InetAddress dir = null;
DatagramPacket paquete;
byte[] buffer = new byte[80];
String recibido, tecleado=" ";
BufferedReader delTeclado = new BufferedReader
                                (new InputStreamReader(System.in));

dir = InetAddress.getByName(args[0]);

yo = new DatagramSocket();

while(tecleado.length()!=0){
    System.out.print("Teclea un mensaje: ");
    tecleado = delTeclado.readLine();
    buffer = tecleado.getBytes();
    paquete = new DatagramPacket(buffer, buffer.length, dir, 5000);
    yo.send(paquete);
    buffer = new byte[80];
    paquete = new DatagramPacket(buffer, buffer.length);
    yo.receive(paquete);
    recibido = new String(paquete.getData());
    System.out.println(">> "+recibido);
}
yo.close();
```

Multicasting

- Es enviar el mismo paquete a varios destinatarios simultáneamente
- Sólo se puede hacer con datagramas, de hecho `MulticastSocket` extiende a `DatagramSocket`
- un paquete *multicast* se envía a un grupo de direcciones IP clase D :
 - 224.0.0.1 - 239.255.255.255
- `MulticastSocket` tiene métodos para unirse y abandonar grupos

Ejemplo ServidorMul

```
public static void main(String[] comandos) throws IOException {
    DatagramSocket yo = new DatagramSocket();
    while (true) {
        byte[] buf = new byte[256];
        String aMandar = new String(new Date().toString());
        buf = aMandar.getBytes();

        InetAddress group = InetAddress.getByName("230.0.0.1");
        DatagramPacket paquete = new DatagramPacket(buf, buf.length, group, 5000);
        yo.send(paquete);
        System.out.println("Ya mandé "+aMandar);
        long ahora = System.currentTimeMillis();
        while(ahora+1000>System.currentTimeMillis());
    }
    // yo.close();
}
```

Ejemplo ClienteMul

```
public static void main(String[] args) throws IOException {  
  
    MulticastSocket yo = new MulticastSocket(5000);  
    InetAddress dir = InetAddress.getByName("230.0.0.1");  
    yo.joinGroup(dir);  
  
    DatagramPacket paquete;  
    while (true) {  
        byte[] buf = new byte[256];  
        paquete = new DatagramPacket(buf, buf.length);  
        yo.receive(paquete);  
        String recibido = new String(paquete.getData());  
        System.out.println("Recibido " + recibido);  
    }  
  
    // yo.leaveGroup(address);  
    // yo.close();  
}
```

Transferencia de objetos

- Los flujos (streams y readers/writers) nos permiten mandar objetos por medio de la red sin problemas
- Capacidad de mandar estructuras de datos complejas
- Capacidad de mandar un objeto para procesamiento en una máquina más potente
- `ServidorM3.java` y `Cliente3.java`