Tema 4
Sockets en Java

Sistemas Distribuidos
Grado en Ingeniería Informática
Universidad Carlos III de Madrid

Contenido

- Conceptos básicos sobre sockets
- Modelo de comunicación
- Sockets
  - Datagrama
  - *Stream*
- API de programación
  - Sockets en C
  - Sockets en Java
- Guía de diseño de aplicaciones cliente-servidor con sockets

Sockets de Java

- El paquete *java.net* de Java permite crear *sockets* TCP/IP
- Clase para trabajar con direcciones: InetAddress
- Clases para *sockets* datagrama:
  - *DatagramSocket:* construye un datagrama
  - *DatagramPacket:* construye un socket dagtagrama

- Clases para *sockets stream:*
  - *ServerSocket:* socket de servidor en un puerto
  - *Socket:* socket para conectar a un puerto

Clase InetAddress

- Clase que representa una dirección IP
- Métodos más útiles:
  - String getHostName(): Devuelve la dirección del host local.
  - String toString(): convierte la dirección IP a un String
  - String getHostAddress(): devuelve la dirección IP en formato texto
  - InetAddress getByName(String host): obtiene la dirección IP de un host dado su nombre.
  - InetAddress getLocalHost(): devuelve la dirección IP del host local.

Ejemplo

address.java

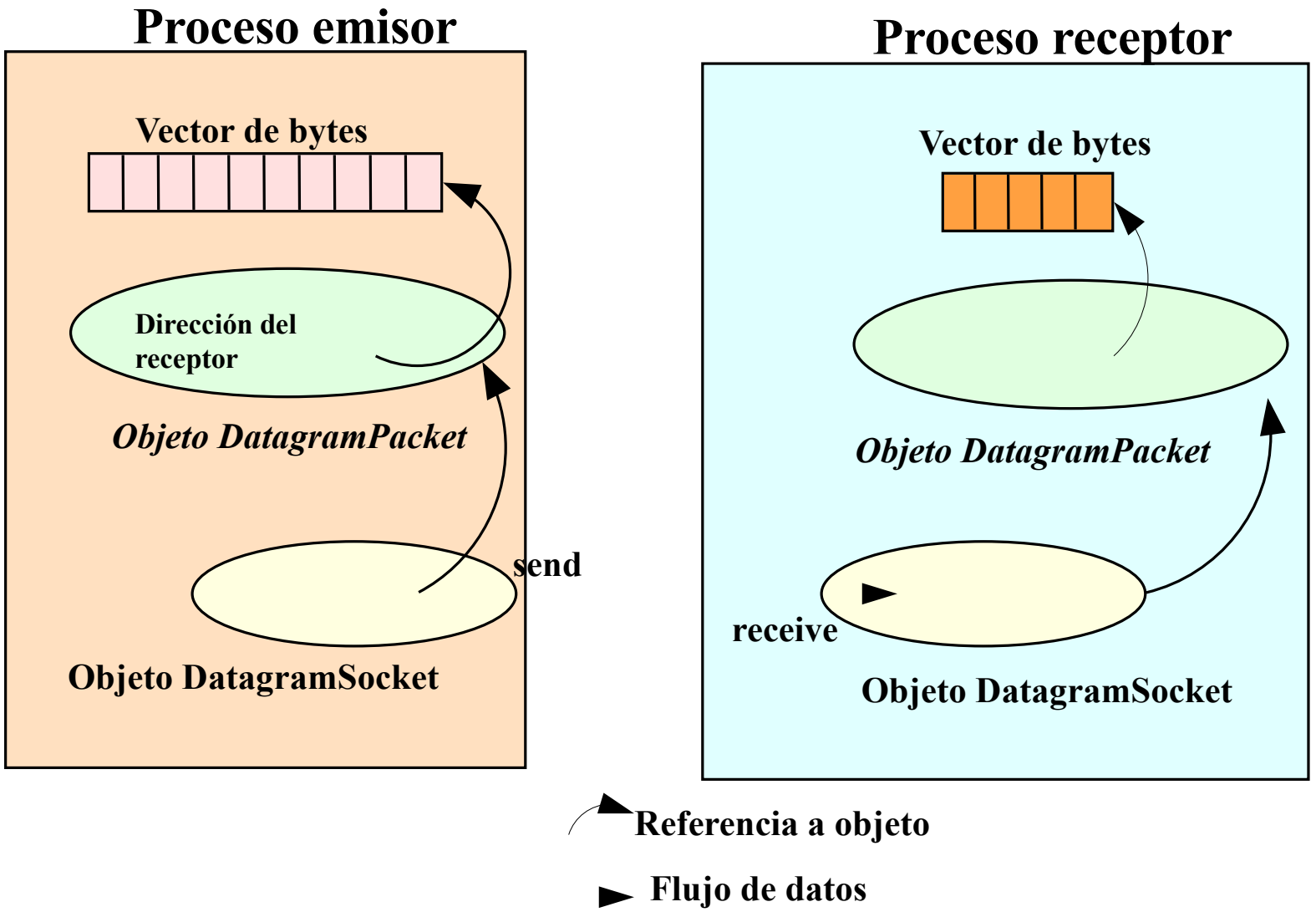
```
import java.net.*;
import java.io.*;

public class Address {
    public static void main(String[] args) {

        try {  // obtiene la dirección IP de la máquina local
                InetAddress host = InetAddress.getLocalHost();
                System.out.println("Host: " + host.getHostName());
        } catch (UnknownHostException e) {
                System.out.println("Error al obtener la direccion");
        }

        try { // obtiene la dirección IP a partir de dominio-punto
                InetAddress host = InetAddress.getByName("www.uc3m.es");
                System.out.println("Host: " + host.getHostAddress());
        } catch (UnknownHostException e) {
                System.out.println("Error al obtener la direccion");
        }

        try { // obtiene dominio-punto a partir de dirección decimal-punto
                InetAddress host = InetAddress.getByName("176.58.10.138");
                System.out.println("Host: " + host.getHostName());
        } catch (UnknownHostException e) {
                System.out.println("Error al obtener la direccion");
        }
    }
}
```

Sockets datagrama

- ***DatagramPacket***:
  - Implementa un objeto que permite enviar o recibir paquetes
  - Constructor: *DatagramPacket*
  - Métodos: *getAddress*, *getPort*, *getData*...

- ***DatagramSocket***:
  - Implementa un *socket* que se puede utilizar para enviar o recibir datagramas.
  - Constructor: *DatagramSocket*
  - Métodos: *bind, send, receive, close, …*

Sockets datagrama

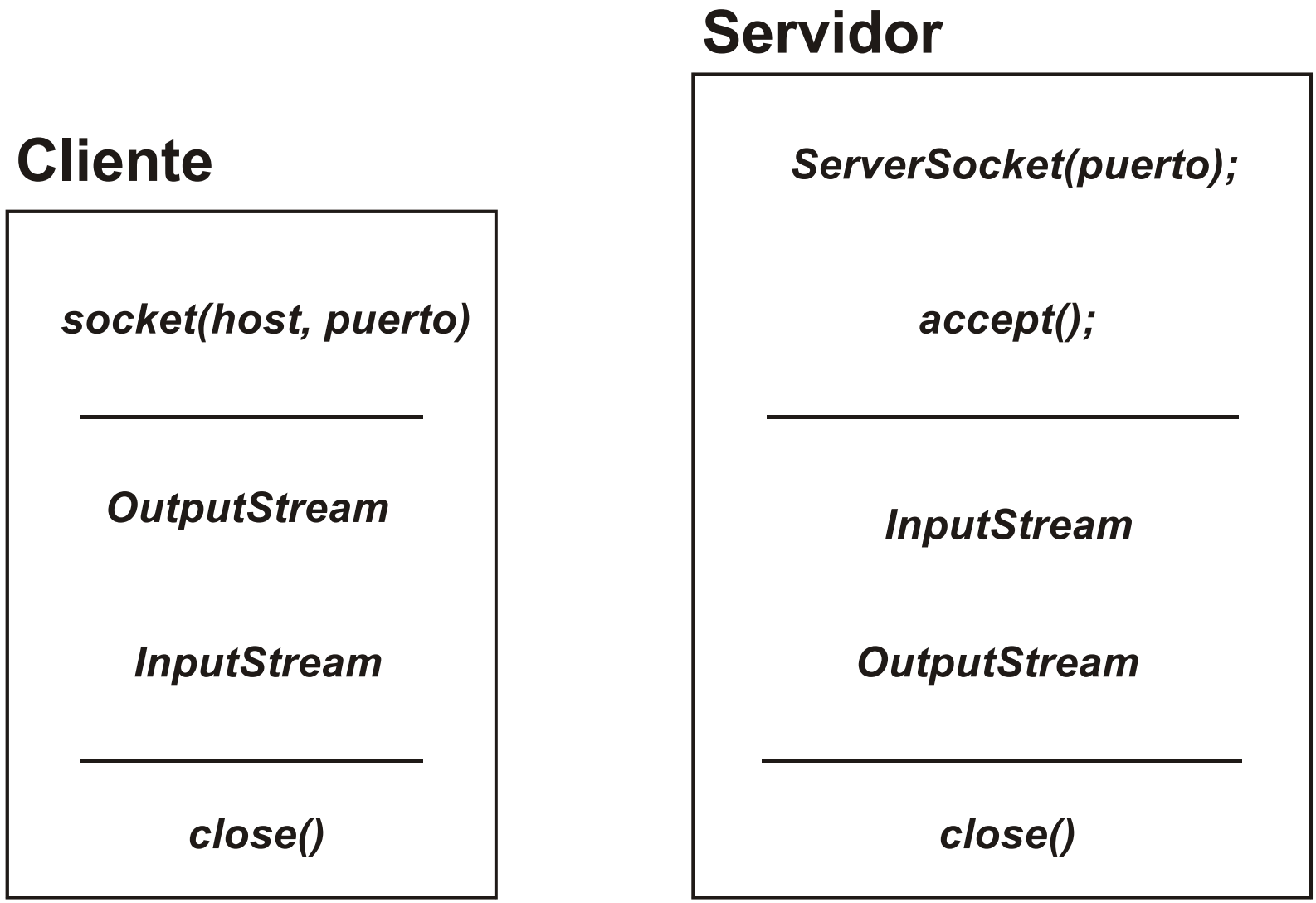


Sockets stream

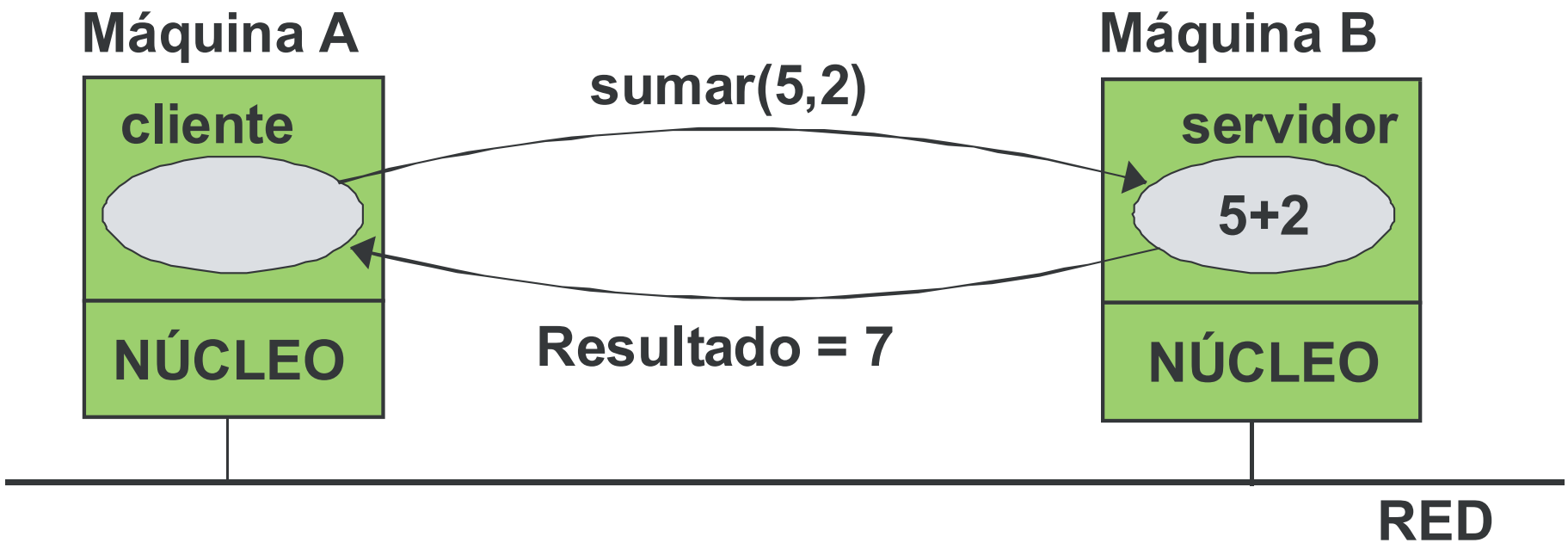
- La clase *socket* implementa un *socket stream del lado del cliente:*
  - Socket(InetAddress dirección, int puerto)
  - Socket(String host, int puerto)
  - Métodos:
  - getOutputStream(): flujo de salida asociado al socket
  - getInputStream(): flujo de entrada asociado al socket
  - getInetAddress(): diección del socket al que está conectado
  - *getPort(): puerto al que está conectado*
  - *getLocalPort(): puerto local*

- La clase *ServerSocket* implementa un *socket* a utilizar en los servidores para esperar la conexiones de los clientes
  - Socket accept()
  - void bind
  - void close()
  - Int getLocalPort()

Ejemplo: Modelo de comunicación con sockets stream



Ejemplo: servidor y cliente usando sockets *stream* de Java



Ejemplo: Servidor *stream*

servidor.java

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

public class servidor
{
   public static void main ( String [] args)
   {
      ServerSocket serverAddr = null;
      Socket sc = null;
      byte op;
      int a, b;
      int res;

      try {
         serverAddr = new ServerSocket(4200);
      }
      catch (Exception e){
         System.err.println("Error creando socket");
      }
```

Ejemplo: Servidor *stream*

servidor.java

```
while (true){
        try {
          sc = serverAddr.accept(); // esperando conexion

          DataInputStream istream = new DataInputStream(sc.getInputStream());
          DataOutputStream ostream = new DataOutputStream(sc.getOutputStream());

          op = istream.readByte();
          a = istream.readInt();
          b = istream.readInt();

          if (op == 0)
                  res = a + b;
          else
                  res = a - b;

          ostream.writeInt(res);

          ostream.flush();
          sc.close();
        }
        catch(Exception e) {
            System.err.println("excepcion " + e.toString() );
            e.printStackTrace() ;
        }
     }
   }
}
```

Ejemplo: Cliente *stream*

cliente.java

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

public class cliente
{
   public static void main ( String [] args)
   {
      byte op;
      int a, b;
      int  res;


      if (args.length != 1) {
         System.out.println("Uso: cliente <host>");
         System.exit(0);
      }
      try {
         // se crea la conexion
         String host = args[0];
         Socket sc = new Socket(host, 4200); //  conexion
```

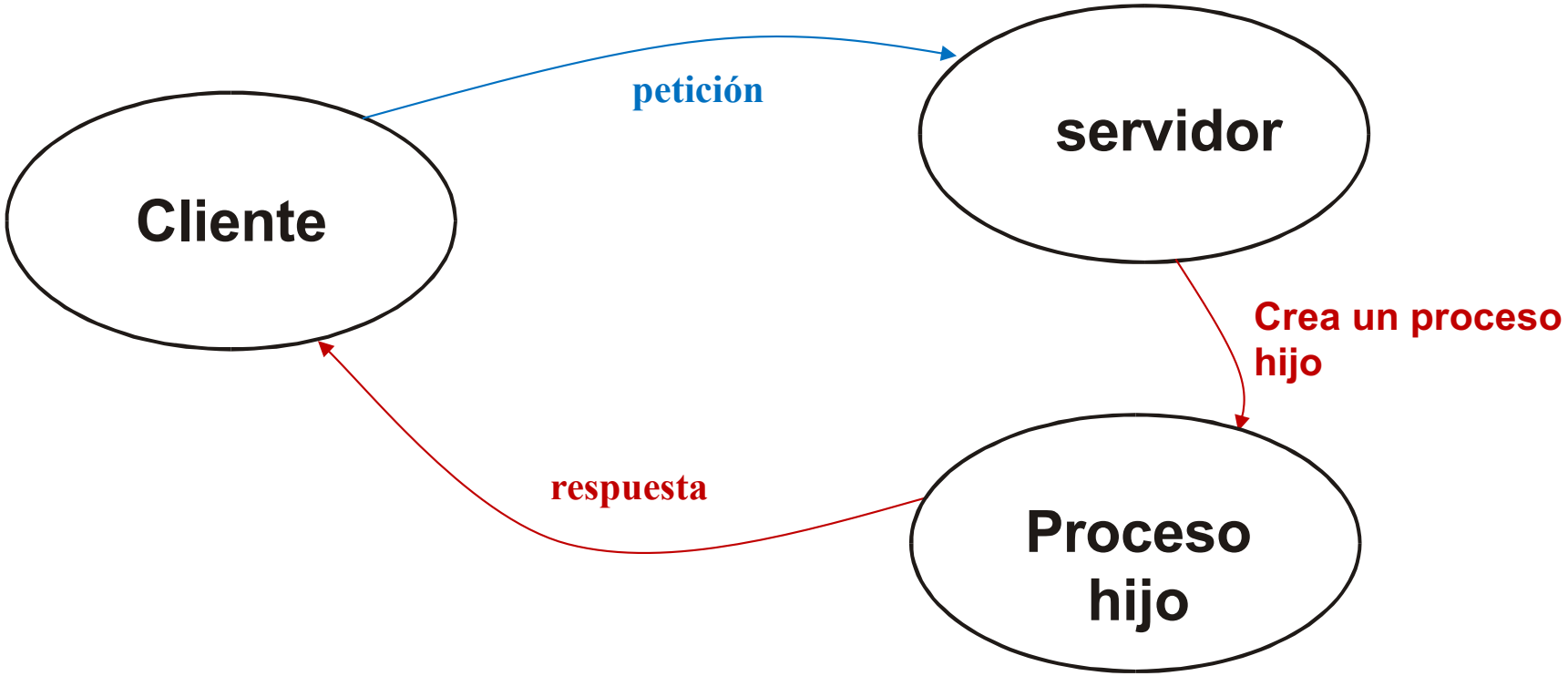
Ejemplo: Cliente *stream*

cliente.java

```
            DataInputStream istream = new DataInputStream(sc.getInputStream());
            DataOutputStream ostream = new DataOutputStream(sc.getOutputStream());

            op = 0;
            a = 5;
            b = 2;

            ostream.writeByte(op);
            ostream.writeInt(a);
            ostream.writeInt(b);
            ostream.flush();

            res = istream.readInt();
            System.out.println("La suma es " + res);

            sc.close();
         }
         catch (Exception e){
            System.err.println("excepcion " + e.toString() );
            e.printStackTrace() ;
         }
      }
}
```

Modelo de servidor concurrente

- El servidor crea un hijo que atiende la petición y envía la respuesta al cliente
- Se pueden atender múltiples peticiones de forma concurrente



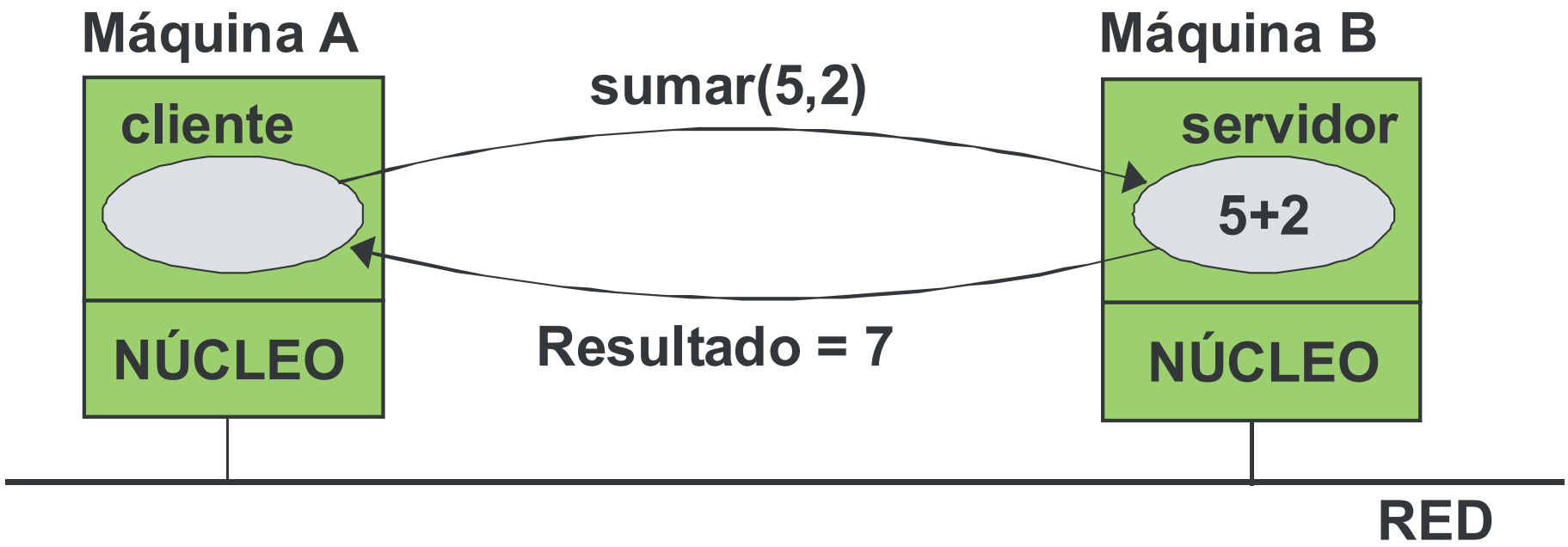
Servidor concurrente en Java (*stream*)

```
while (true){
          try {
            Socket cliente = serverAddr.accept();
             new TratarPeticion(cliente).start();
          }
          catch(Exception e) {
             System.err.println("excepcion " + e.toString() );
             e.printStackTrace() ;
          }
      }
   }
}
```

Servidor concurrente en Java (*streams*)

```
class TratarPeticion extend Thread {
  private Socket sc;

  TratarPeticion(Socket s) {
       sc = s;
  }

  public void run() {
       // TODO: código del cliente
  }
```

Ejemplo: servidor y cliente usando sockets datagramas de Java



Ejemplo: Servidor (datagramas)

servidor.java

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

public class servidor {
   public static void main ( String [] args) {
      DatagramSocket  s = null;
      DatagramPacket in, out;
      InetAddress client_addr = null;
      int client_port;
      byte brecv[] = new byte[100];
      byte bsend[] = new byte[100];
      int peticion[], res;

      try  {
         s  = new DatagramSocket(420);
         in = new DatagramPacket(brecv, 100); // paquete para recibir la solicitud
```

Ejemplo: Servidor (datagramas)

servidor.java

```
while (true) {
        s.receive(in); //esperamos a recibir

        // obtener datos
         brecv = in.getData();
         client_addr = in.getAddress();
         client_port = in.getPort();

         // desempaquetar los datos
         ByteArrayInputStream bais = new ByteArrayInputStream(brecv);
         ObjectInputStream ois      = new ObjectInputStream(bais);

         peticion = (int[])ois.readObject();

         if (peticion[0] == 0)
                res = peticion[1] + peticion[2];
         else
                res = peticion[1] - peticion[2];
```

Ejemplo: Servidor (datagramas)

servidor.java

```
        ByteArrayOutputStream baos =  new ByteArrayOutputStream();
        DataOutputStream dos       =  new DataOutputStream(baos);
        dos.writeInt(res);
        bsend = baos.toByteArray();

        out = new DatagramPacket ( bsend,
                                   bsend.length,client_addr,
                                   client_port);

        s.send(out);
     }
    }
   catch(Exception e) {
                System.err.println("excepcion " + e.toString() );
                e.printStackTrace() ;
   }
  }
}
```

Ejemplo: Cliente (datagramas)

cliente.java

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

public class cliente{
   public static void main ( String [] args)
   {
      byte bsend[] = new byte[100];
      byte brecv[] = new byte[100];

      InetAddress server_addr = null;
      DatagramSocket s = null;
      DatagramPacket in = null;
      DatagramPacket out = null;
      int res;
      int peticion[] = new int[3];

      if (args.length != 1) {
          System.out.println("Uso: cliente <host>");
          System.exit(0);
      }
```

Ejemplo: Cliente (datagramas)

cliente.java

```
try
   {
        // se crea el socket del cliente
        s = new DatagramSocket();

        // direcion del servidor
        server_addr = InetAddress.getByName(args[0]);

        peticion[0] = 0;  // sumar
        peticion[1] = 5;
        peticion[2] = 8;

        // empaquetar los datos.
        ByteArrayOutputStream baos = new ByteArrayOutputStream() ;
        ObjectOutputStream  oos = new ObjectOutputStream(baos);
        oos.writeObject(peticion);
        bsend = baos.toByteArray() ; // se obtiene el buffer (datagrama)

        // un unico  envio
        out = new DatagramPacket (bsend, bsend.length, server_addr, 4200);
        s.send(out);
```

Ejemplo: Cliente (datagramas)

cliente.java

```
            // se recibe el datagrama de respuesta
            in = new DatagramPacket (brecv, 100);
            s.receive(in);

            // se obtiene el buffer
            brecv = in.getData();

            // se desempaqueta
            ByteArrayInputStream bais = new ByteArrayInputStream(brecv) ;
            DataInputStream dis = new DataInputStream(bais);

            res = dis.readInt();
            System.out.println("Datos recibidos " + res);
      }
      catch (Exception e)      {
                  System.err.println("<<<<<excepcion " + e.toString() );
          e.printStackTrace() ;
       }
    }
}
```

Lectura de cadenas de caracteres con sockets stream

- Cuando una cadena de caracteres finaliza con el código ASCII '\0' no se sabe a priori su longitud y no se puede usar la función recvMessage para leerla
- En este caso hay que leer byte a byte hasta leer el el código ASCII '\0':
- Asociado a un objeto `sc` de tipo socket o serverSocket existen dos flujos de datos que se pueden utilizar para enviar y recibir datos del socket:

```
DataOutputStream out = new DataOutputStream(sc.getOutputStream());
DataInputStream  in  = new DataInputStream(sc.getInputStream());
```

- Para leer una cadena de caracteres:

```
byte[] ch = new byte[1];
String mensaje = new String();

do{
        ch[0] = in.readByte();
        if (ch[0] != '\0'){
              String d = new String(ch);
              mensaje = mensaje + d;
        }
}while(ch[0] != '\0');
```

Lectura de cadenas de caracteres con sockets stream

- Función para leer una cadena de caracteres:

```
String ReadString(DataInputStream  in)
{
      byte[] ch = new byte[1];
      String mensaje = new String();

      do{
            ch[0] = in.readByte();
            if (ch[0] != '\0'){
                  String d = new String(ch);
                  mensaje = mensaje + d;
            }
      }while(ch[0] != '\0’);

      return mensaje;
}
```

Envío de cadenas de caracteres con sockets stream

- Asociado a un objeto `sc` de tipo socket o serverSocket existen dos flujos de datos que se pueden utilizar para enviar y recibir datos del socket:

```
DataOutputStream out = new DataOutputStream(sc.getOutputStream());
DataInputStream  in  = new DataInputStream(sc.getInputStream());
```

- Para enviar una cadena de caracteres:

```
String mensaje = new String("Hola");
out.writeBytes(mensaje);
out.write('\0');// inserta el código ASCII 0 al final
```

Servidor stream y cliente de suma enviando cadenas de caracteres

- Para enviar un número como cadena:

```
int n = 1234;
String  cadena = Integer.toString(n);
out.writeBytes(mensaje);
out.write('\0');
```

Envía la cadena más el código ASCII '\0' que indica el fin de la cadena.

- De esta forma se independiza del formato concreto en el que se almacenen los enteros

Servidor stream y cliente de suma enviando cadenas de caracteres

- Para recibir un número como cadena de un `DataInputStream in`:

```
byte[] ch = new byte[1];
String mensaje = new String();

mensaje = ReadStream(in);

try {
      int numero = Integer.parseInt(mensaje);
} catch (final NumberFormatException e) {
      System.out.println("error de conversión");
}
```

Protocolo de cliente de suma en Java

```
char op = 0; // suma
int a = 5 ;
int b = 6;
int res;

// PETICIÓN
out.write(op) ;   // envío operación
String  cadena = Integer.toString(a); // envío a
out.writeBytes(mensaje);
out.write('\0’);
cadena = Integer.toString(b);      // envío b
out.writeBytes(mensaje);
out.write('\0’);

//RESPUESTA
String mensaje = ReadString(in);
res = Integer.parseInt(mensaje); // obtiene res
```

Protocolo de servidor de suma

```
char op = 0; // suma
int a;
int b;
int res;
String mensaje;

op = in.ReadByte(); // obtiene op
// recibe argumentos
mensaje = ReadString(in);
a = Integer.parseInt(mensaje);          // obtiene a
mensaje = ReadString(in);
b = Integer.parseInt(mensaje);          // obtiene b

if (op == 0) {
          res = a + b;
          String  cadena = Integer.toString(n);
          out.writeBytes(cadena);
          out.write('\0');
}
```