Paso de mensajes en Java

Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid

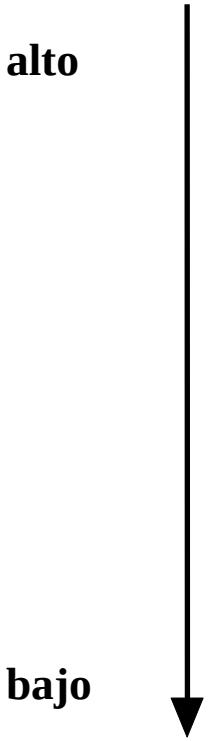
Contenidos

1. Introducción:
   1. Paradigma de paso de mensajes
   2. Entorno de programación Java
2. Paso de mensajes en Java: sockets
   1. Introducción a sockets
   2. Sockets en Java
      1. *Datagrama*
      2. *Orientado a conexión*
      3. *Difusión (comunicación en grupo)*

Contenidos

1. **Introducción:**
   1. **Paradigma de paso de mensajes**
   2. Entorno de programación Java
2. Paso de mensajes en Java: sockets
   1. Introducción a sockets
   2. Sockets en Java
      1. *Datagrama*
      2. *Orientado a conexión*
      3. *Difusión (comunicación en grupo)*

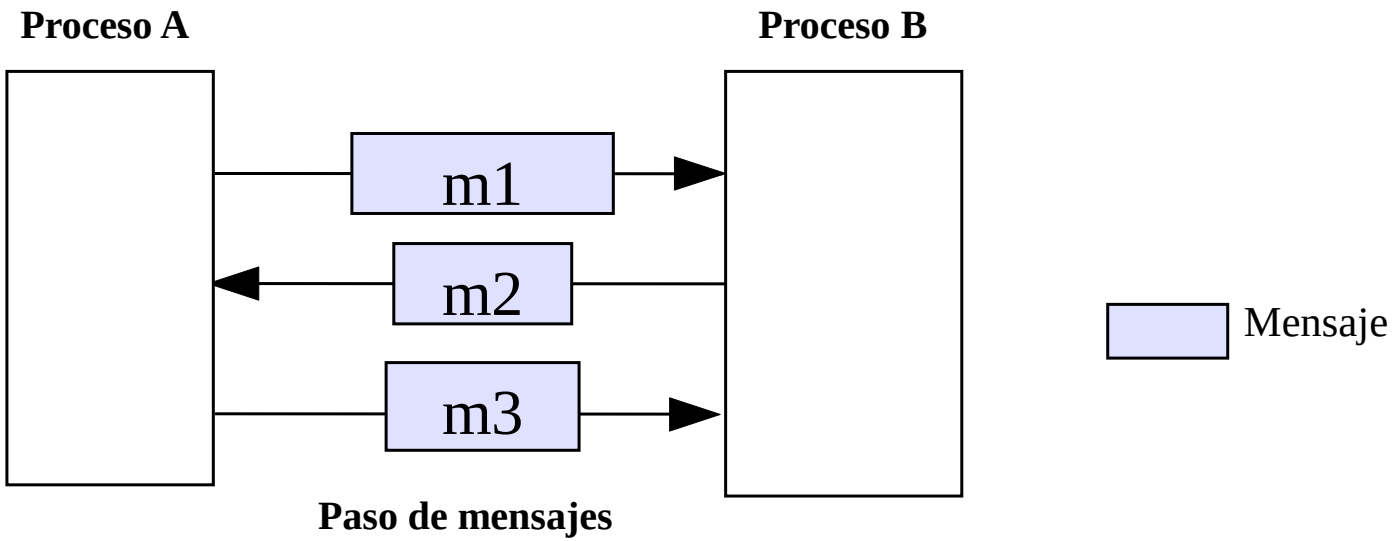
Paradigma de paso de mensajes



Espacio de objetos, aplicaciones colaborativas
Servicios de red, *object request broker*, agentes móviles
procedimientos remotos, métodos remotos
Cliente-servidor, *peer-to-peer*
Paso de mensajes

Paradigma de paso de mensajes

- Paradigma fundamental para aplicaciones distribuidas
  - Un proceso envía un mensaje de solicitud
  - El mensaje llega al receptor, el cual procesa la solicitud y devuelve un mensaje en respuesta
  - Esta respuesta puede originar posteriores solicitudes por parte del emisor



Paradigma de paso de mensajes

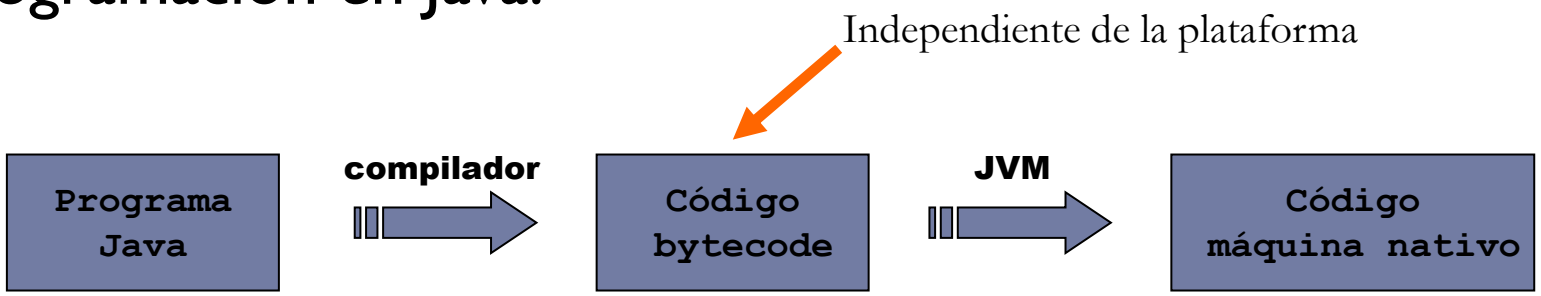
- Las operaciones básicas para soportar el paradigma de paso de mensajes son *enviar* y *recibir*
  - Protocolos más comunes: IP y UDP
- Para las comunicaciones orientadas a conexión también se necesitan las operaciones *conectar* y *desconectar*
  - Protocolo más común: TCP
- Operaciones de Entrada/Salida que encapsulan el detalle de la comunicación a nivel del sistema operativo
  - Ejemplo: el API de *sockets*

Contenidos

1. **Introducción:**
   1. Paradigma de paso de mensajes
   2. **Entorno de programación Java**
2. Paso de mensajes en Java: sockets
   1. Introducción a sockets
   2. Sockets en Java
      1. *Datagrama*
      2. *Orientado a conexión*
      3. *Difusión (comunicación en grupo)*

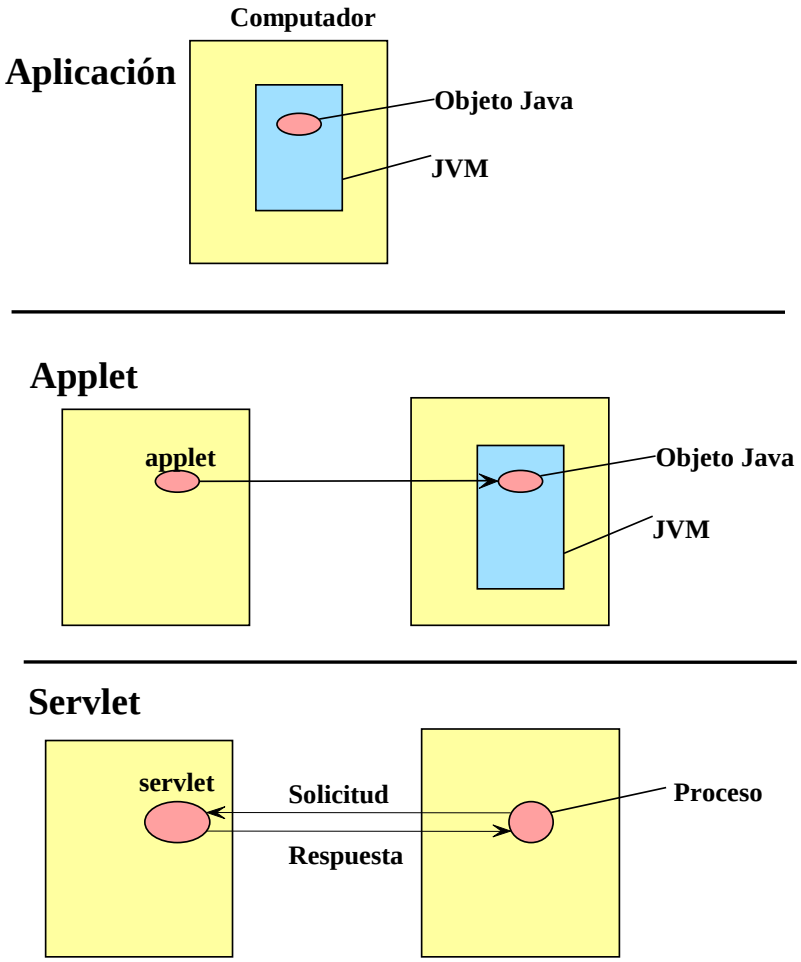
Java: programación

- Programación en Java.



- Tipos de programas en Java.
  - Aplicaciones.
  - *Applets*.
  - *Servlets*.

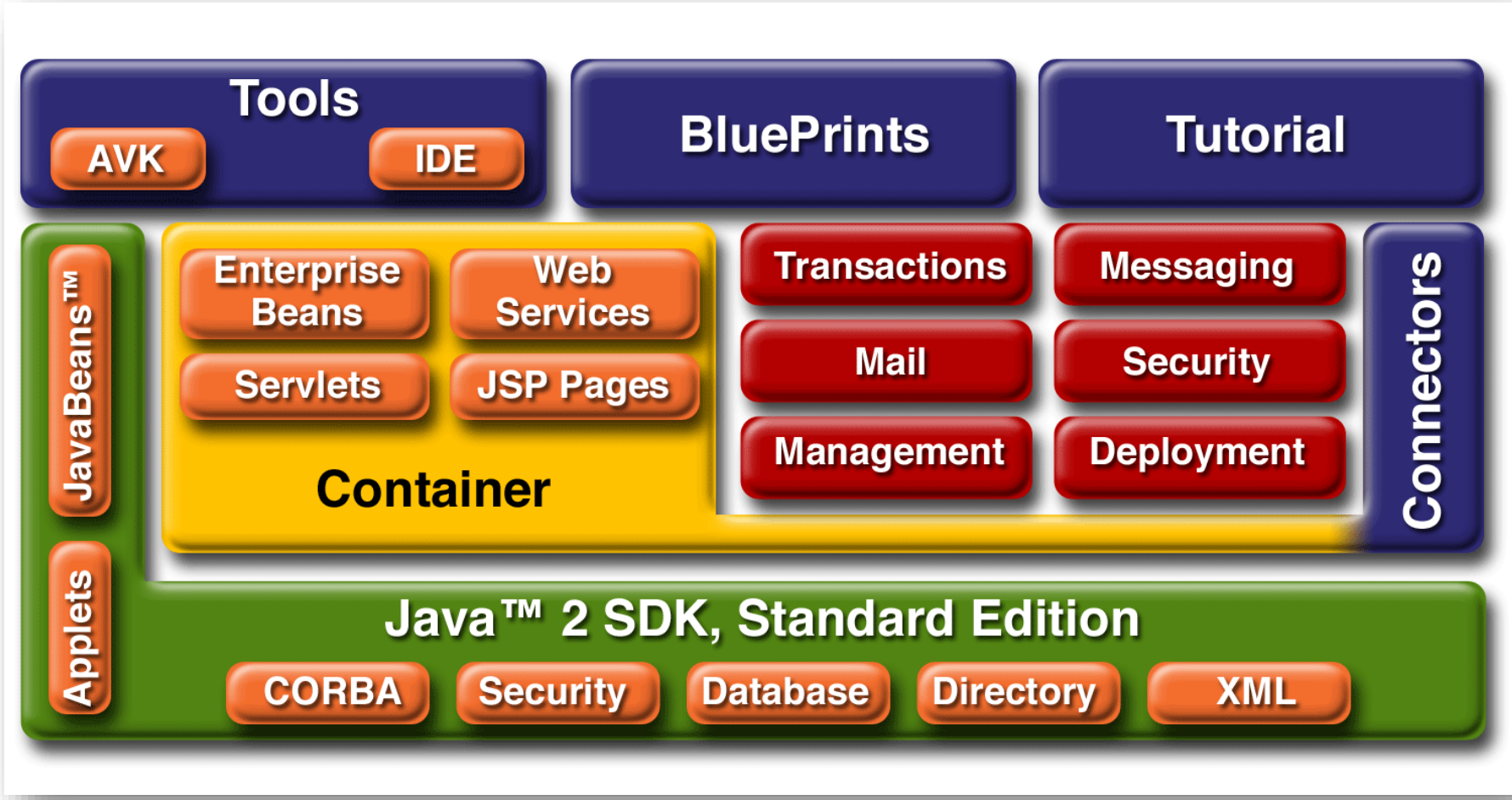
Java: tipos de programas



Java: características

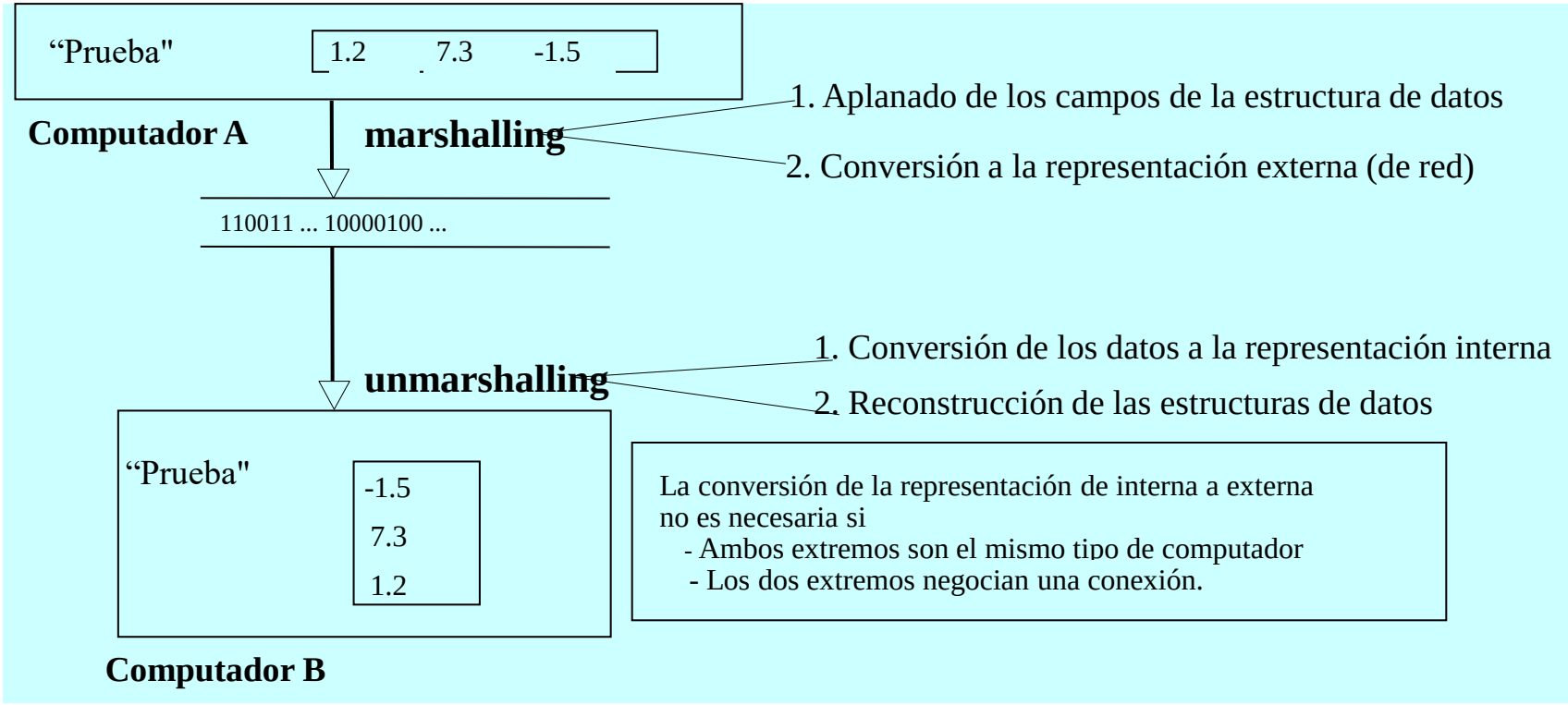
- Java no tiene punteros
  - Recolector de basura
- Java soporta programación *multi-threading*.
  - Riesgo de condiciones de carrera → regiones críticas.
- Java ofrece soporte para crear aplicaciones distribuidas
  - Diversas bibliotecas para uso de protocolos de comunicación

Java: características



Empaquetamiento de datos

- Transformaciones necesarias para poder transmitir datos o estructuras entre distintos ordenadores



Java: comunicación de datos y objetos

- Java soporta la serialización de objetos.
- Empaquetamiento y transmisión de objetos entre procesos.

```
theFile = new File(args[0]);
outStream = new FileOutputStream(theFile);
objStream = new ObjectOutputStream(outStream);
...
objStream.writeInt(3);
objStream.writeObject(new Crouton(7));
objStream.writeObject(new Tomato("Mars", 11, 5))
...
int primitive = objStream.readInt();
crunch = (Crouton) objStream.readObject();
Object tomato = objStream.readObject();
```

Contenidos

1. Introducción:
   1. Paradigma de paso de mensajes
   2. Entorno de programación Java
2. **Paso de mensajes en Java: sockets**
   1. **Introducción a sockets**
   2. Sockets en Java
      1. *Datagrama*
      2. *Orientado a conexión*
      3. *Difusión (comunicación en grupo)*

Sockets: introducción

- Aparecieron en 1981 en UNIX BSD 4.2
  - Intento de incluir TCP/IP en UNIX
  - Diseño independiente del protocolo de comunicación
- Abstracción que:
  - Se representa un extremo de una comunicación bidireccional con una dirección asociada
    - En el caso de sockets basados en TCP/UDP se tiene una dirección IP y un puerto
  - Ofrece interfaz de acceso a los servicios de red en el nivel de transporte
    - Protocolo TCP
    - Protocolo UDP
    - Otros…

Sockets: introducción

- Sujetos a proceso de estandarización dentro de POSIX
  - POSIX 1003.1g
- Actualmente:
  - Disponibles en casi todos los sistemas UNIX
  - Disponibles en otros sistemas operativos
    - WinSock: API de *sockets* de Windows
  - Accesible desde muchos lenguajes
    - En Java como clase nativa

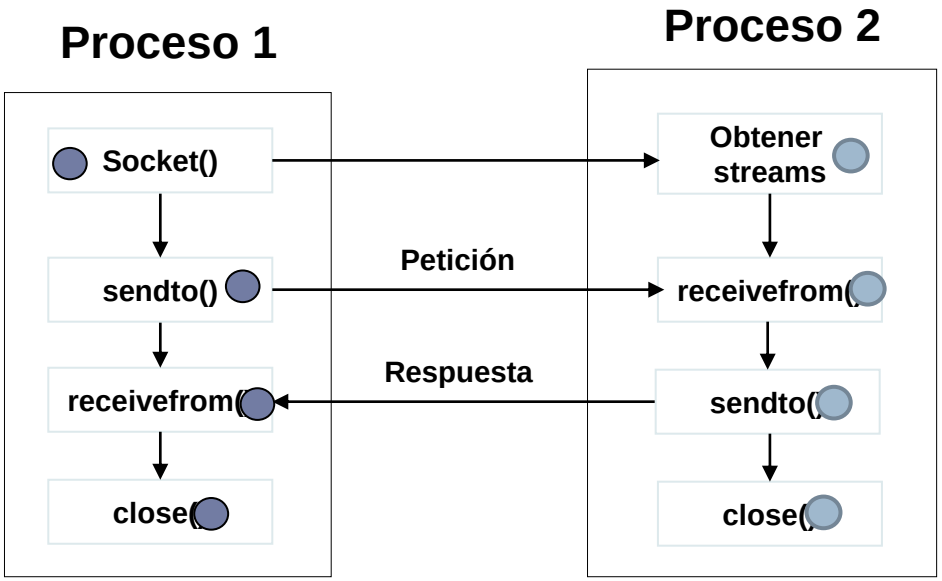
Tipos de sockets

- ***Datagrama sin conexión***
  - Sin conexión.
  - No fiable, no se asegura el orden en la entrega.
  - Mantiene la separación entre mensajes.
  - Asociado al protocolo UDP.

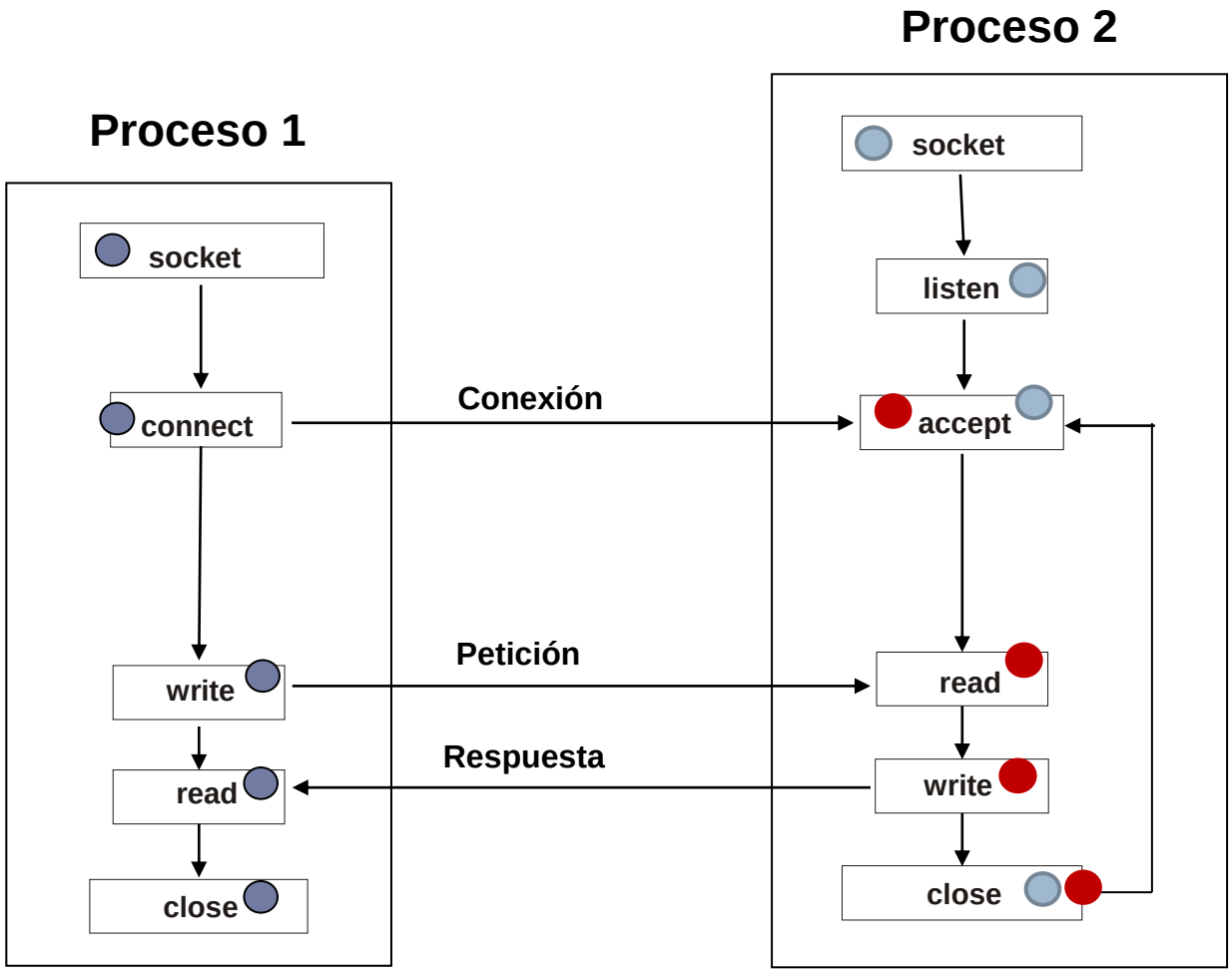
- ***Datagrama con conexión***
  - Orientado a conexión (no a nivel de red, sino nivel lógico).
  - Fiable, se asegura el orden de entrega de mensajes.
  - No mantiene separación entre mensajes.
  - Asociado al protocolo UDP.

- ***Sockets STREAM***
  - Concepto de flujo de datos.
  - Asociado al protocolo TCP.

Uso de sockets *datagram*



Uso de sockets *stream*



Contenidos

1. Introducción:
   1. Paradigma de paso de mensajes
   2. Entorno de programación Java
2. **Paso de mensajes en Java: sockets**
   1. Introducción a sockets
   2. **Sockets en Java**
      1. ***Datagrama***
      2. *Orientado a conexión*
      3. *Difusión (comunicación en grupo)*

Sockets de Java

- El paquete *java.net* de Java permite crear y gestionar *sockets* TCP/IP.

- Clases para *sockets* datagrama:
  - *DatagramSocket*
  - *DatagramPacket*

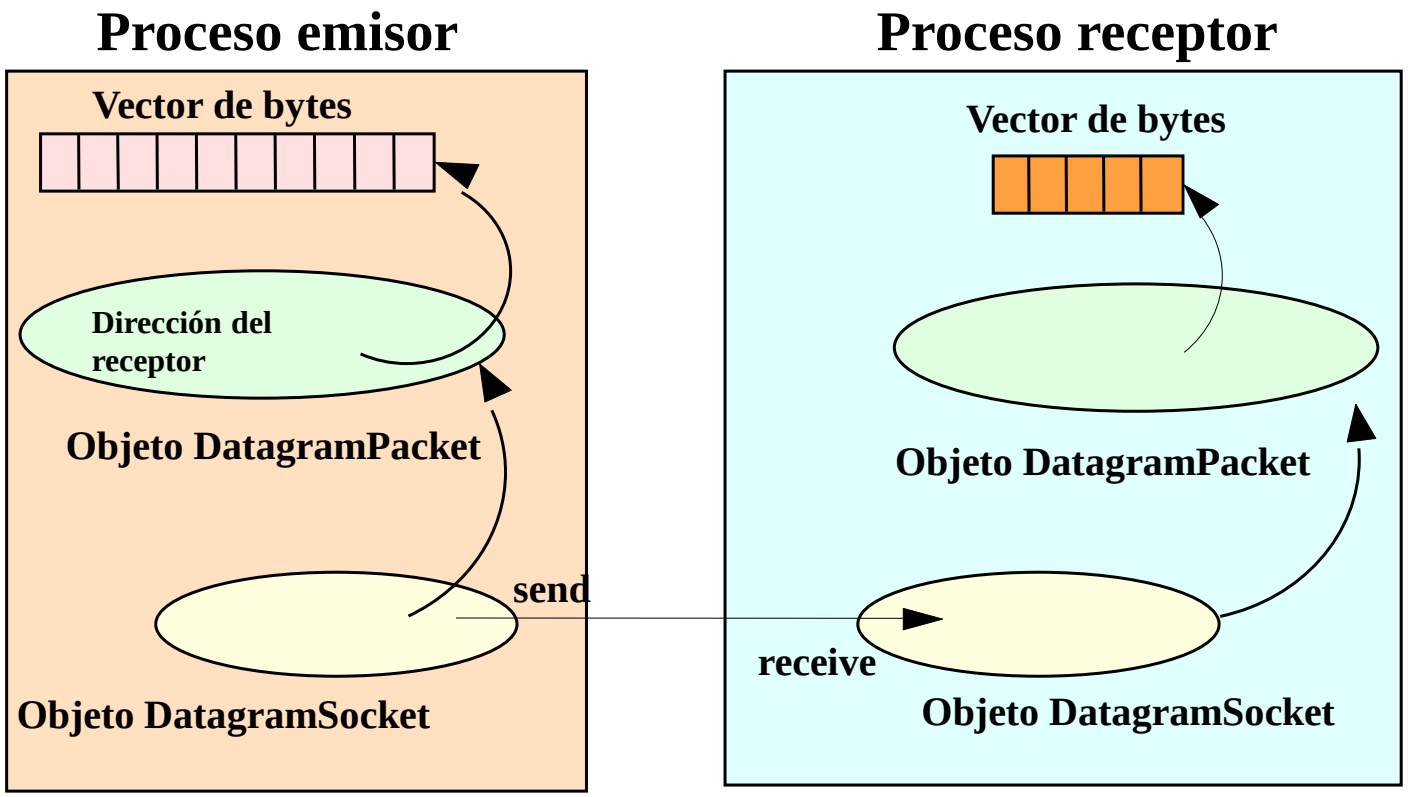
- Clases para *sockets stream:*
  - *ServerSocket*
  - *Socket*

Sockets datagrama

- ***DatagramPacket***:
  - implementa un objeto que permite enviar o recibir paquetes.
  - Constructor: *DatagramPacket*.
  - Métodos: *getAddres*, *getPort*, ...

- ***DatagramSocket***:
  - implementa un *socket* que se puede utilizar para enviar o recibir datagramas.
  - Constructor: *DatagramSocket*.
  - Métodos: *send*, *receive*, *close*, *setSoTimetout*, *getSoTimeout*,...

Sockets datagrama

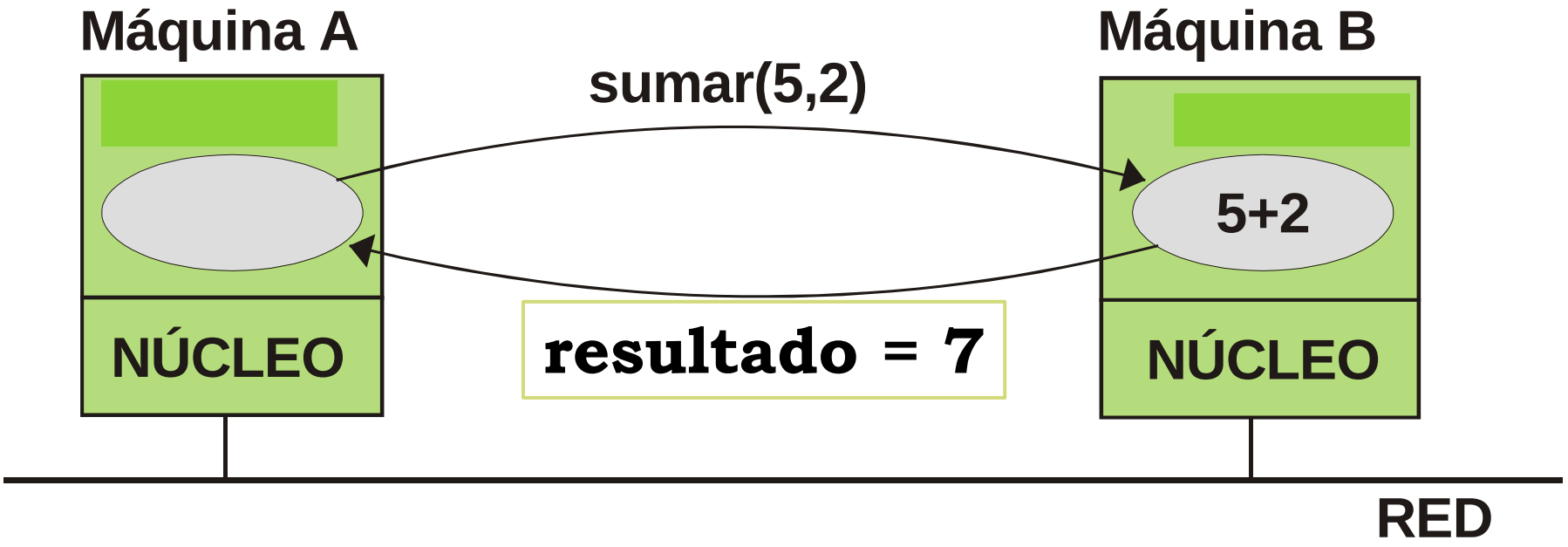


Sockets datagrama

```
// Ejemplo para un emisor
InetAddress receiverHost=
    InetAddress.getByName("localHost");
DatagramSocket theSocket = new DatagramSocket( );
String message = "Hello world!";
byte[ ] data = message.getBytes( );

DatagramPacket thePacket
    = new DatagramPacket(data, data.length,
                         receiverHost, 2345);
theSocket.send(thePacket );
```

```
//Ejemplo para un receptor
DatagramSocket ds = new DatagramSocket(2345);
DatagramPacket dp =
      new DatagramPacket(buffer, MAXLEN );
ds.receive(dp);
len = dp.getLength ( );
System.out.Println (len + " bytes received.\n");
String s = new String( dp.getData( ), 0, len);
System.out.println (dp.getAddress ( ) + " at port "
   + dp.getPort( ) + " says " + s);
```

Ejemplo (datagramas)



Emisor (datagramas)

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

public class client{
   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 num[] = new int[2];

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

Emisor (datagramas)

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

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

      num[0] = 2;
      num[1] = 5;

      // empaquetar los datos.
      ByteArrayOutputStream baos = new ByteArrayOutputStream() ;
      ObjectOutputStream     dos = new ObjectOutputStream(baos);
      dos.writeObject(num);

      bsend = baos.toByteArray() ; // se obtiene el buffer (datagrama)
      // un único envio
      out = new DatagramPacket(bsend, bsend.length, server_addr, 2500);
      s.send(out);
```

```
// Excerpt from the sending process
InetAddress   receiverHost =
    InetAddress.getByName   ("localHost ");
DatagramSocket   theSocket   = new  DatagramSocket  ( );
String message = "Hello world!";
byte[ ]  data  = message.getBytes  ( );

DatagramPacket   thePacket
    = new  DatagramPacket  (data , data.length  ,
                                receiverHost , 2345);
theSocket  .send (thePacket    );
```

Emisor (datagramas)

```
//Excerpt from a receiver program
DatagramSocket ds = new DatagramSocket(2345);
DatagramPacket dp =
      new DatagramPacket(buffer,  MAXLEN );
ds.receive(dp);
len = dp.getLength ( );
System.out.Println(len + " bytes received.\n");
String s = new String( dp.getData( ), 0, len);
System.out.println (dp.getAddress ( ) + " at port "
   + dp.getPort( ) + " says " + s);
```

```
        // 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() ;
     }
  }
}
```

Receptor (datagramas)

```
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 num[], res;
      try  {
         s = new DatagramSocket(2500);
         in = new DatagramPacket(brecv, 100);  // paquete para recibir la
                                               // solicitud
```

Receptor (datagramas)

```
//Excerpt from a receiver program
DatagramSocket ds = new DatagramSocket(2345);
DatagramPacket dp =
      new DatagramPacket(buffer,  MAXLEN );
ds.receive(dp);
len = dp.getLength ( );
System.out.Println(len + " bytes received.\n");
String s = new String( dp.getData( ), 0, len);
System.out.println (dp.getAddress ( ) + " at port "
   + dp.getPort( ) + " says " + s);
```

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

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

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

      num = (int[])dis.readObject();     res = num[0] + num[1];
```

Receptor (datagramas)

```
// Excerpt from the sending process
InetAddress   receiverHost =
    InetAddress.getByName   ("localHost ");
DatagramSocket   theSocket   = new  DatagramSocket  ( );
String message = "Hello world!";
byte[ ]  data  = message.getBytes  ( );

DatagramPacket   thePacket
    = new  DatagramPacket  (data , data.length  ,
                                receiverHost , 2345);
theSocket  .send (thePacket    );
```

```
      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() ;
   }
  }
}
```

Contenidos

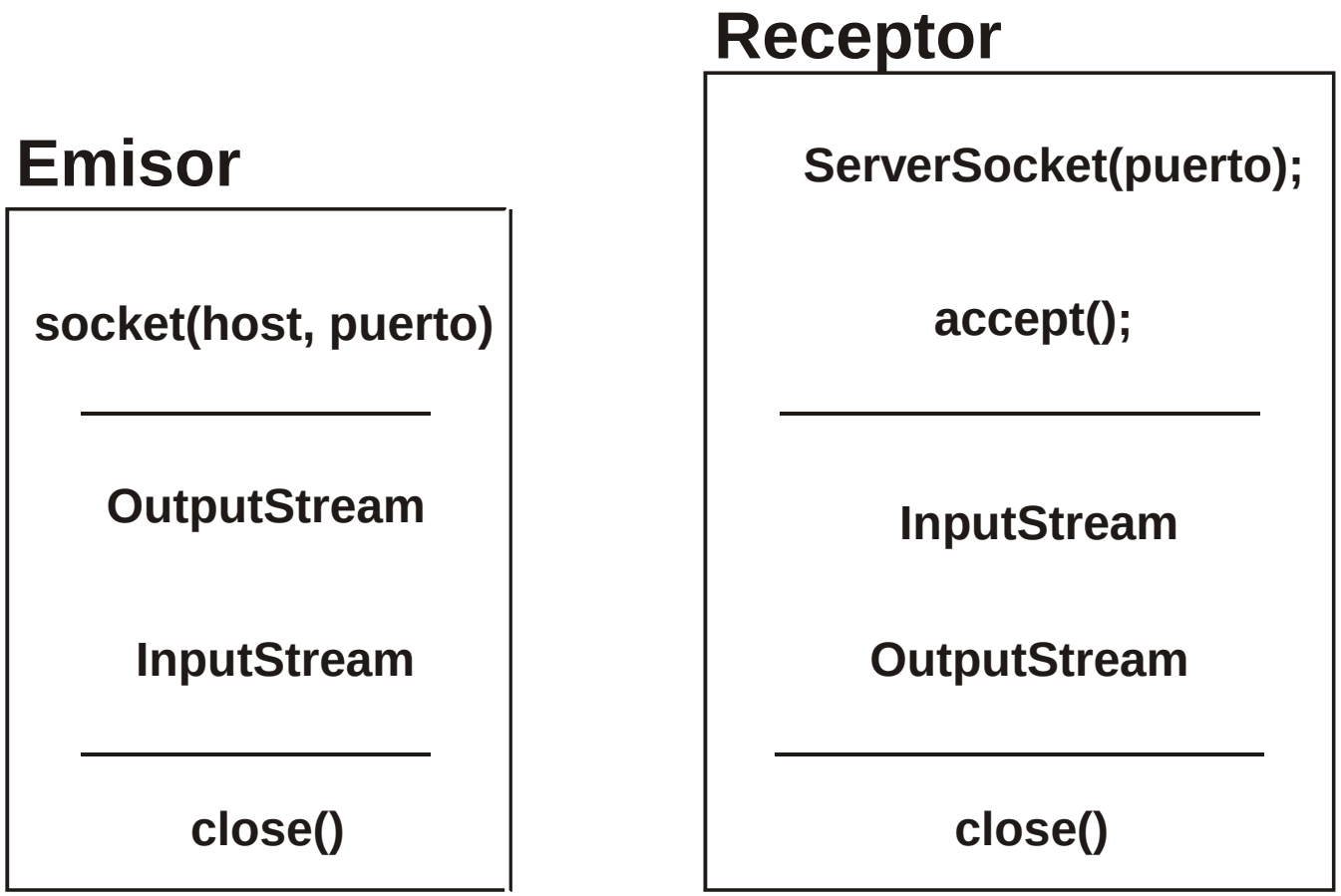
1. Introducción:
   1. Paradigma de paso de mensajes
   2. Entorno de programación Java
2. **Paso de mensajes en Java: sockets**
   1. Introducción a sockets
   2. **Sockets en Java**
      1. *Datagrama*
      2. ***Orientado a conexión***
      3. *Difusión (comunicación en grupo)*

Sockets stream

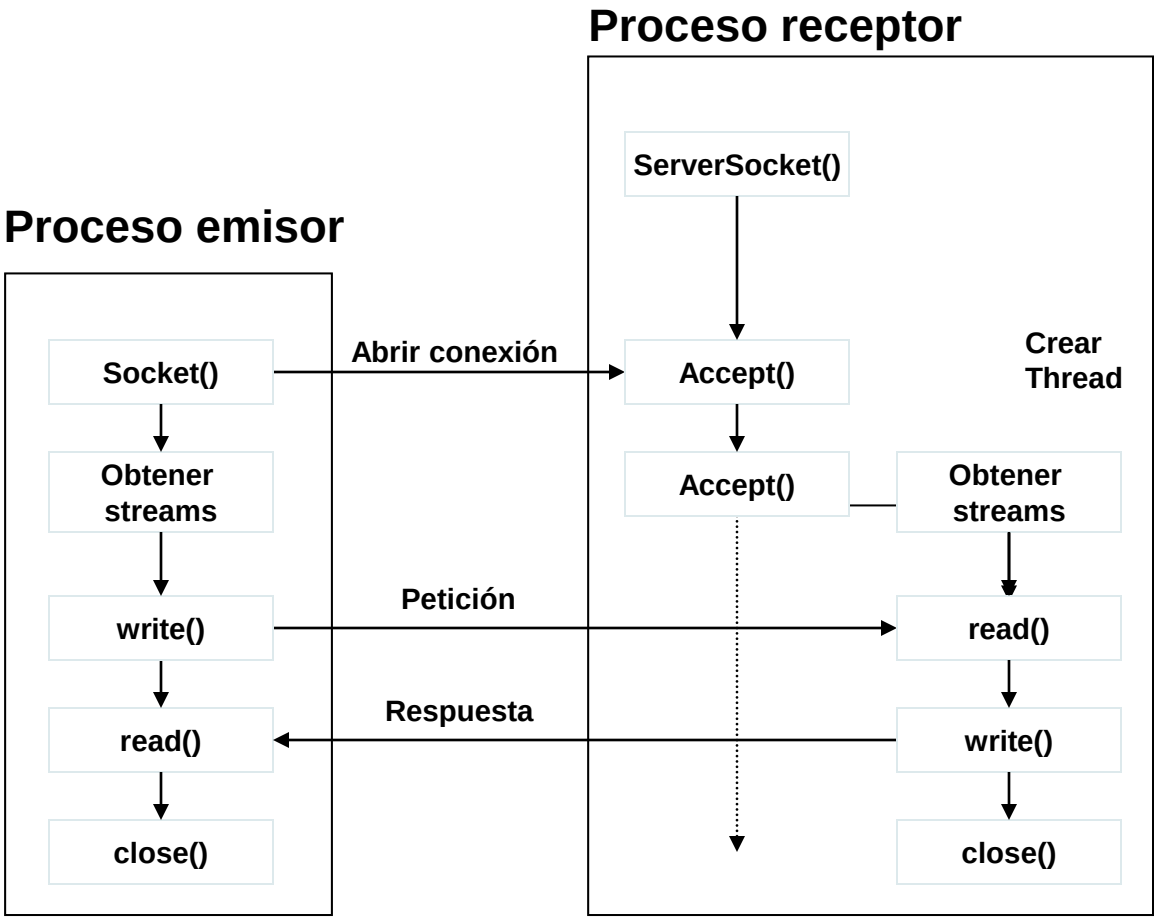
- La clase *socket* implementa un *socket stream*
  - Socket(InetAddress dirección, int puerto)
  - OutputStream getOutputStream()
    - flush,
  - InputStream getInputStream()
  - Void setSoTimeout(int tiempo_de_espera)

- La clase *ServerSocket* implementa un *socket* a utilizar en los servidores para esperar la conexiones de los clientes
  - Socket accept()
  - Void close()
  - Void setSoTimeout(int tiempo_de_espera)

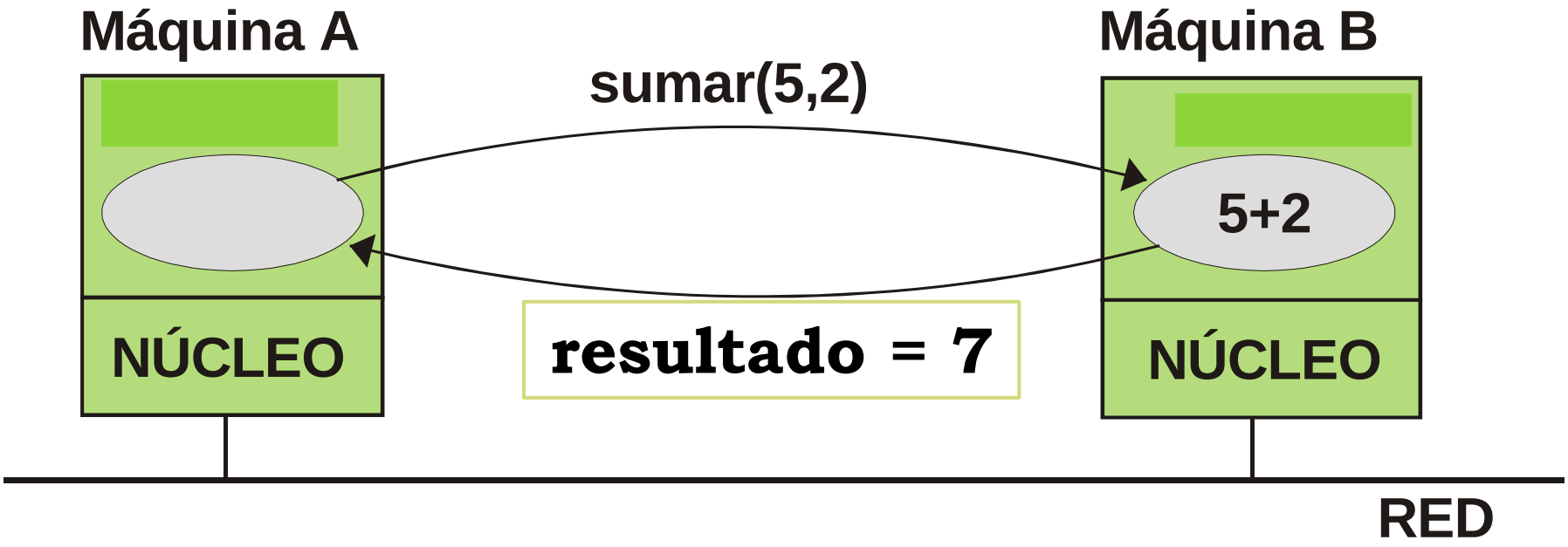
Esqueleto con sockets *streams* de Java



Esqueleto con sockets *streams* de Java



Ejemplo (streams)



Emisor (*streams*)

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

public class client
{
   public static void main ( String [] args)
   {
      int  res;
      int num[] = new int[2];

      if (args.length != 1) {
         System.out.println("Uso: cliente <host>");
         System.exit(0);
      }
      try {         // se crea la conexión
         String host = args[0];
         Socket sc = new Socket(host, 2500); //  conexión
```

Emisor(*streams*)

```
            OutputStream ostream = sc.getOutputStream();
            ObjectOutput s = new ObjectOutputStream(ostream);

            num[0] = 5;   num[1] = 2;  //prepara la petición

            s.writeObject(num);
            s.flush();

            DataInputStream istream = new DataInputStream(sc.getInputStream());
            res = istream.readInt();

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

Receptor (*streams*)

```
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;
      int num[] ;  // petición
      int res;
      try {
         serverAddr = new ServerSocket(2500);
      }
      catch (Exception e){
         System.err.println("Error creando socket");
      }
```

Receptor (*streams*)

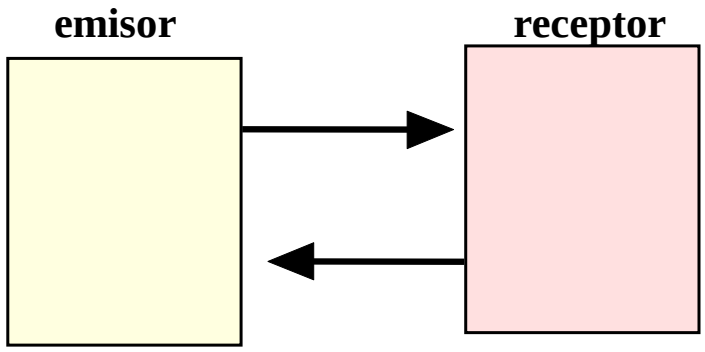
```
while (true) {
          try {
            sc = serverAddr.accept(); // esperando conexión
            InputStream istream = sc.getInputStream();
            ObjectInput in = new ObjectInputStream(istream);

            num = (int[]) in.readObject();
            res = num[0] + num[1];

            DataOutputStream ostream = new DataOutputStream(sc.getOutputStream());
            ostream.writeInt(res);
            ostream.flush();
            sc.close();
           } catch(Exception e) {
             System.err.println("excepcion " + e.toString() );
             e.printStackTrace() ;
           }
      } // while
   } // main
} // servidor
```

Contenidos

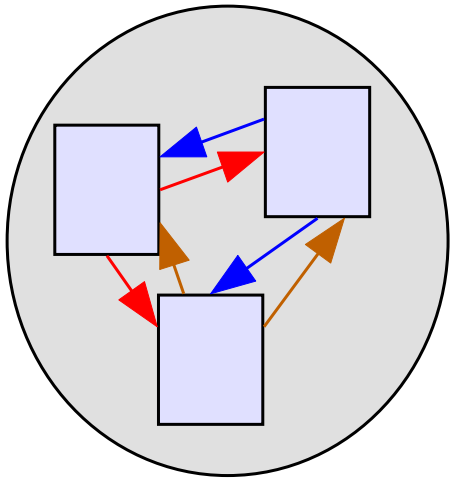
1. Introducción:
   1. Paradigma de paso de mensajes
   2. Entorno de programación Java
2. **Paso de mensajes en Java: sockets**
   1. Introducción a sockets
   2. **Sockets en Java**
      1. *Datagrama*
      2. *Orientado a conexión*
      3. ***Difusión (comunicación en grupo)***

Comunicación en grupo

- IPC en grupo o multidifusión.
- Demanda:
  - Aplicaciones.
  - Aumento de la robustez del sistema.



IPC uno-a-uno



IPC grupo o multidifusión

Comunicación en grupo

- Operaciones primitivas:
  - Incorporación
  - Abandono
  - Envío
  - Recepción
- Multidifusión:
  - Sin conexión: baja latencia.
    - Audio/vídeo en tiempo real.
  - Orientada a conexión: alta latencia.
    - Multidifusión de datos.

Tipos de sistemas de multidifusión

- No fiable
- Fiable
  - Sin orden
    - Los mensajes no tienen orden entre procesos, y los de un mismo proceso
  - FIFO
    - Se mantiene el orden de los mensajes de un mismo proceso (distintos entrelazados)
  - Orden casual
    - Relación *sucede-antes*/*casual* entre todos los mensajes
  - Orden atómico
    - Relación *sucede-antes*/*casual* entre *ciertos* mensajes

Ejemplos de los distintos tipos de sistemas de multidifusión

- Sin orden

  Se emiten m_1, m_2, m_3

  m_1-m_2-m_3, m_1-m_3-m_2, m_2-m_1-m_3, m_2-m_3-m_1, m_3-m_1-m_2, m_3-m_2-m_1

- FIFO

  A emite A_1-A_2 y *B emite* B_1-B_2

  A_1-A_2-B_1-B_2,
  A_1-B_1-A_2-B_2,
  A_1-B_1-B_2-A_2,
  B_1-A_1-A_2-B_2
  B_1-A_1-B_2-A_2
  B_1-B_2-A_1-A_2.

- Orden casual

  A emite A_1, *B emite* B_1, A emite $-A_2$ y *B emite* B_2

  A_1-A_2-B_1-B_2,

Ejemplos de los distintos tipos de sistemas de multidifusión

- Orden atómico:

Ejemplo 1:

P_1 envía m_1, P_2 ***envía*** m_2, y P_3 envía m_3.

m_1-m_2- m_3, m_1- m_3- m_2, m_2- m_1-m_3,

m_2-m_3-m_1, m_3-m_1- m_2, m_3-m_2-m_1.

Ejemplo 2:

P_1 envía m_1 y luego m_2.

P_2 responde a m_1 enviando m_3.

P_3 responde a m_3 enviando m_4

Orden a respetar: m_1 - m_3 - m_4

m_1- m_2- m_3- m_4, m_1- m_3- m_2- m_4, m_1- m_3- m_4- m_2.

Multidifusión en Java

- Extensión del protocolo UDP.
- Multidifusión no fiable.
- Ofrece clases próximas a las APIs de *sockets:*
  - *InetAddress*: identifica grupo de multidifusión.
  - *DatagramPacket*: paquete de datos enviado a todos los participantes y paquete de datos recibido por cada participante.
  - *MulticastSocket*: permite gestionar *sockets* multidifusión.
- Dirección IP de multidifusión:
  - Direcciones de multidifusión del protocolo de internet.
  - Dirección IP de clase D: 224.0.0.0 a 239.255.255.255.
  - Número de puerto UDP.

Direcciones de multidifusión asignadas

224.0.0.1	All Systems on this Subnet
224.0.0.11	Mobile-Agents 224.0.1.23 XINGTV
224.0.1.84	jini-announcement
224.0.1.85	jini-request
224.0.1.115	Simple Multicast
224.0.6.000-224.0.6.127	Cornell ISIS Project
224.0.7.000-224.0.7.255	Where-Are-You
224.0.8.000-224.0.8.255	INTV
224.0.9.000-224.0.9.255	Invisible Worlds
2240.12.000-224.0.12.063	Microsoft and MSNBC
224.0.18.000-224.0.18.255	Dow Jones
224.0.19.000-224.0.19.063	Walt Disney Company
224.0.22.000-224.0.22.255	WORLD MCAST
224.2.0.0-224.2.127.253	Multimedia Conference Calls

Multidifusión en Java

- Incorporación a un grupo de multidifusión

```
// Unirse a la dirección 239.1.2.3 puerto 3456
InetAddress group = InetAddress.getByName("239.1.2.3")
MulticastSocket s = new MulticastSocket(3456)
s.joinGroup(group);
```

- Envío a un grupo de multidifusión

```
String msg = "Mensaje de difusión.";
InetAddress group =  InetAddress.getByName("239.1.2.3");
MulticastSocket s = new   MulticastSocket(3456);
s.joinGroup(group);    // opcional
DatagramPacket hi = new  DatagramPacket(msg.getBytes(),
                                        msg.length(),group, 3456);
s.send(hi);
```

Multidifusión en Java

- Recepción de mensajes enviados

```
byte[] buf = new byte[1000];
InetAddress group =InetAddress.getByName("239.1.2.3");
MulticastSocket s = new MulticastSocket(3456);
s.joinGroup(group);
DatagramPacket recv = new DatagramPacket(buf, buf.length);
s.receive(recv);
```

- Abandono de un grupo de multidifusión

```
s.leaveGroup(group);
```

Multidifusión en Java

- Tiempo de vida
  - Objetivo: evitar que los mensajes circulen indefinidamente.
  - Número de enlaces a través de los que se retransmite el paquete.

```
String msg = "Hola!";
InetAddress group = InetAddress.getByName("224.0.0.1");
MulticastSocket s = new MulticastSocket(3456);
s.setTimeToLive(1);   // multidifusión en máquinas locales
DatagramPacket hi = new DatagramPacket(msg.getBytes(),
                                       msg.length(),group, 3456);
s.send(hi);
```

Multidifusión en Java

Valores del tiempo de vida:

```
0 <= tiempo_de_vida <= 255
```

- **0** misma máquina
- **1** misma red local
- **32** misma zona
- **64** misma región
- **128** mismo continente
- **255** no está restringida

Multidifusión en Java

- Interfaces de multidifusión fiable:
  - ***The Java Reliable Multicast Service*** (***JRM Service***).
  - ***Sistema Totem*** de la Universidad de California.
  - TASC's ***Reliable Multicast Framework*** (***RMF***)
    - Multidifusión FIFO.

Paso de mensajes en Java

Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid