Sockets in Java

Oscar García Lorenzo

EISTI

oscar.garcialorenzo@eisti.fr

February 27, 2017

Overview

Java Sockets

Java Sockets

- Remember, there are no Sockets in C++
- To teach sockets in a different way, in OOP, Java is normally used.
- It is simpler than C, but same theory.

IP Addresses

public class java.net.InetAddress

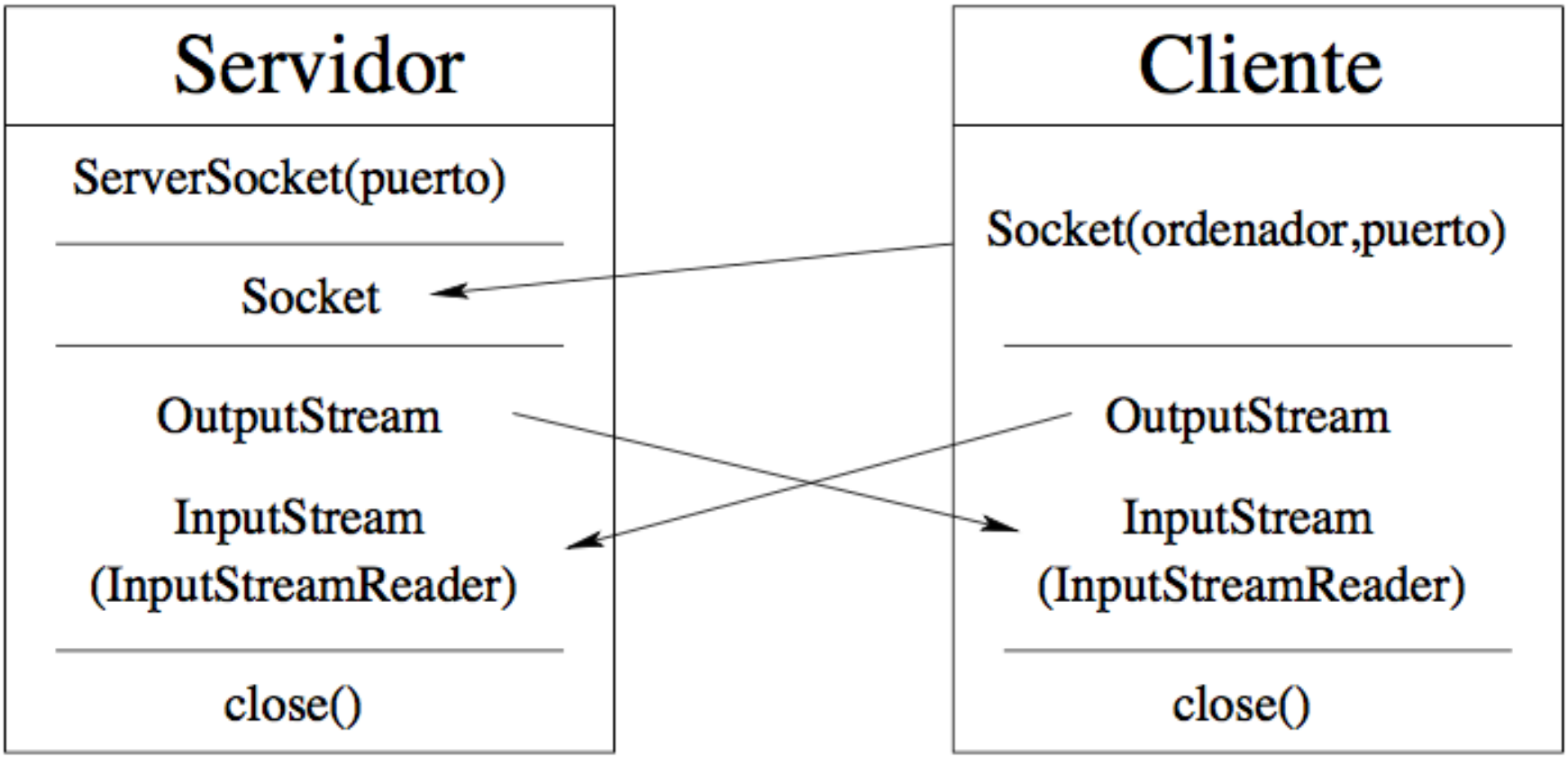
- `static InetAddress getLocalHost()` returns an object containing the name and IP address of the local computer
- `static InetAddress getByName (String host)` returns an `InetAddress` object from the name or IP address in textual format (numbers and points) of a computer.
- `static InetAddress getByAddress (byte [] addr)` returns an `InetAddress` object from the IP address in binary format (4 bytes) of a computer.
- `static InetAddress [] getAllByName (String host)` provides the set of `InetAddress` objects (alias names and multiple IP addresses) from the name or IP address of a computer.

IP Addresses

public class java.net.InetAddress

- `String getHostName()` returns the hostname from an `InetAddress` object.
- `String getHostAddress()` returns a string with the IP address in textual format (numbers and points) given an `InetAddress` object.
- String toString () provides a string of text in the hostname / IP address form from an `InetAddress` object.
- `Byte [] getAddress()` returns the IP address in binary format (four bytes) from an `InetAddress` object.

TCP


Servidor
ServerSocket(puerto)
Socket
OutputStream
InputStream
(InputStreamReader)
close()
Cliente
Socket(ordenador,puerto)
OutputStream
InputStream
(InputStreamReader)
close()

Java TCP Sockets

public class java.net.ServerSocket

- Only to wait for requests.
- `ServerSocket(port)` creates a server socket at that port, listening on any address.
- `accept()` waits for a connection to be accepted.

public class java.net.Socket

- The client socket, different!.
- `Socket(address,port)` asks for a connection.

Sockets communicate through `InputStream` (or `InputStreamReader`) and `OutputStream`

Server Sockets

We need a socket to accept connections

```
ServerSocket welcomeSocket;
welcomeServidor = new ServerSocket( port );
```

We need a socket to communicate

```
Socket connectionSocket;
connectionServidor = welcomeSocket.accept();
```

Client Socket

```
Socket cliSocket;
try { cliSocket = new Socket( "computer", port ); }
catch( IOException e ) { System.out.println( "message" ); }
```

Buffered Streams

public class DataInputStream

- Allows the reading of text lines and Java primitive data types in a highly portable way.
- `readChar()`, `readInt()`, `readFloat()` y `readDouble()`
- We must use the one function necessary depending on the type of data that we expect to receive from the server.

my_socket depends on the program, it already has a space for the input.

```
DataInputStream dinput;
dinput = new DataInputStream( my_socket.getInputStream() );
```

Buffered Streams

public class BufferedReader and InputStreamReader

- Better to read text lines.
- `readLine()`

```
BufferedReader dinput;
dinput = new BufferedReader(new InputStreamReader (
my_socket.getInputStream() ) );
```

Buffered Streams

public class DataOutputStream

- Allows you to write any of the Java primitive types in the output stream.
- `writeBytes(String s)`,`writeInt(int v)`, `writeFloat(float v)`, `writeDouble(double v)`

```
DataOutputStream dOutput;
dOutput = new DataOutputStream( my_socket.getOutputStream() );
```

Buffered Streams

public class PrintStream

- Better for text.
- `write()`,`println()`

```
PrintStream dOutput;
dOutput = new PrintStream( my_socket.getOutputStream() );
```

Remember to close

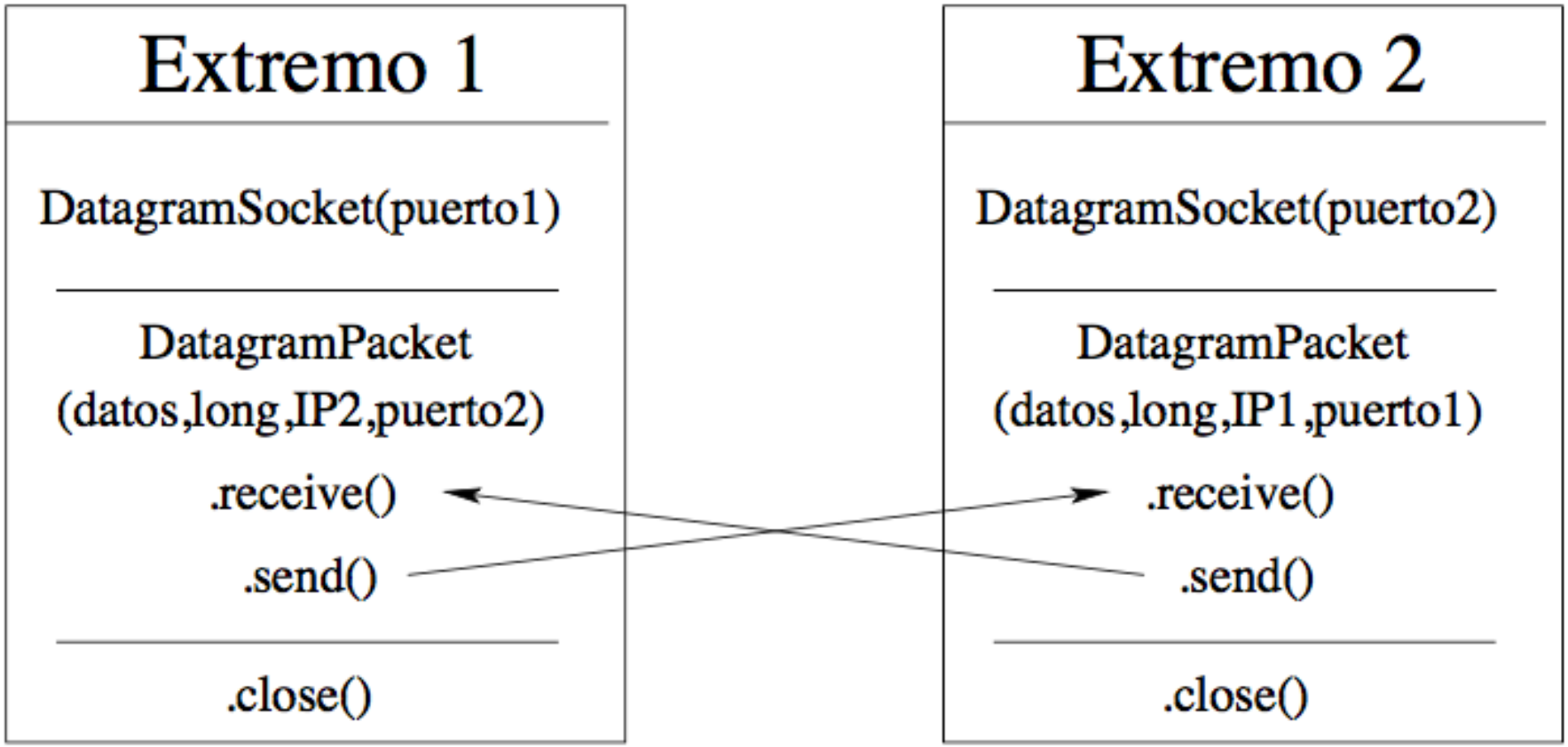
Client:

```
dOutput.close(); dInput.close(); cliSocket.close();
```

Server:

```
dOutput.close(); dInput.close(); welcomeSocket.close();connect
```

UDP


Extremo 1
DatagramSocket(puerto1)
DatagramPacket
(datos,long,IP2,puerto2)
.receive()
.send()
.close()
Extremo 2
DatagramSocket(puerto2)
DatagramPacket
(datos,long,IP1,puerto1)
.receive()
.send()
.close()

Java UDP Sockets

public class java.net.DatagramSocket

- Same for everyone.
- `receive(DatagramPacket p)`, receives a datagram packet from this socket.
- `send(DatagramPacket p)`, sends a datagram packet from this socket.

```
DatagramSocket socketDat;
socketDat = new DatagramSocket( port );
```

Better would be:

```
DatagramSocket socketDat;
try { socketDat = new DatagramSocket( port );
} catch( SocketException e ) { System.out.println(e); }
```

Java UDP

Now we need to know the IP of the other end:

```
InetAddress IP;
IP=InetAddress.getByName( "name or IP" );
```

Java UDP Sockets

public class java.net.DatagramPacket

- We need to send Datagrams.
- `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 on the specific host.

To send:

```
String msg = "write something";
byte[] data = new byte[1024];
data = msg.getBytes();
DatagramPacket sendPacket = new DatagramPacket
      (data, data.length,IP,port);
socketDat.send( sendPacket );
```

Java UDP Sockets

To receive:

```
byte[] dataRec = new byte[1024]; DatagramPacket packetRec =
  new DatagramPacket(dataRec,dataRec.length);
socketDat.receive( packetRec  );

String frase = new String( packetRec.getData() );
InetAddress IP = packetRec.getAddress();
int port = packetRec.getPort();
```

Java UDP Sockets

Remember to close.

references

References

- "TCP/IP Sockets in C: Practical Guide for Programmers" by Michael J. Donahoo and Kenneth L. Calvert
- Sockets and Client/Server Communication, Jeff Chase, Duke University