Network Programming in JAVA (Sockets)

Redes e Serviços

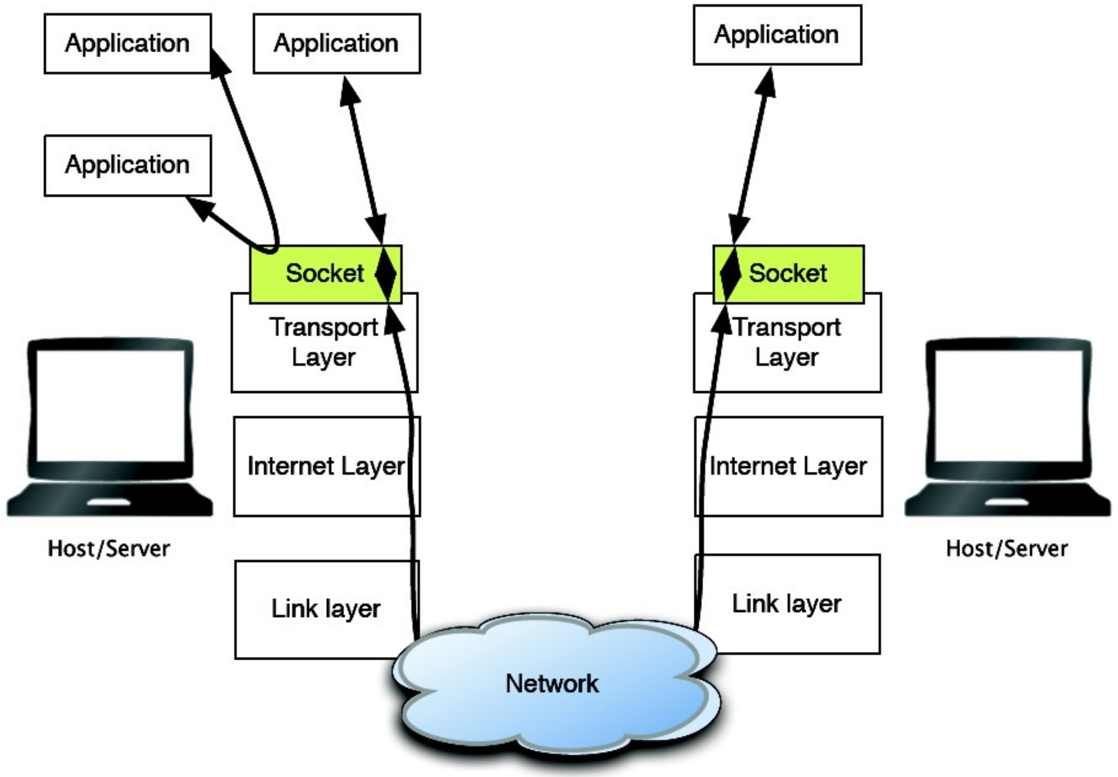
Licenciatura em Engenharia Informática
DETI-UA

universidade de aveiro

deti.ua.pt

Sockets (1)

- **Inter-process communication mechanism**
  - Either local or remote processes
- **Provide an abstraction for processes to exchanging information**
  - Follows a client/server paradigm.



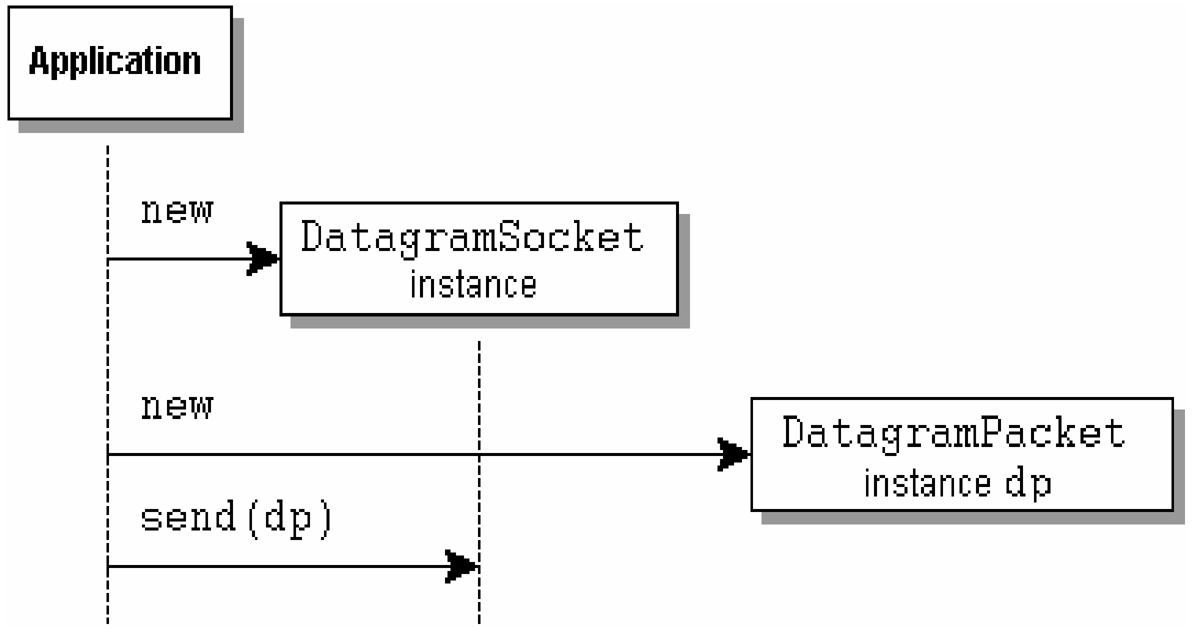
Sockets (2)

- **A Socket is identified by**
  - Family: AF_INET (IPv4), AF_INET6 (IPv6) and many other less common.
    - Defines the address structure.
    - Defines also the communications layer (e.g. IP version).
  - Type: Determines what transport protocol is used.
    - UDP – Connectionless.
    - TCP – Connection oriented.
    - RAW – Direct access to a layer of the stack.
      - Allows to send and receive crafted packets.
      - e.g. the ping command (ICMP packets).
  - Address: local address(IP or path)
    - Also remote address if connection oriented
  - Port: Local port 0-65535
    - Also remote port if connection oriented
- **Restriction**
  - 1 socket per Address per Port per Protocol per Family per Host

Sockets (3)

- **`AF_INET/AF_INET6 families`**
  - Allows communication between processes on any IP/IPv6 enabled machine.
  - Endpoints can be on local or remote machines
    - 127.0.0.1 or ::1 for the localhost
- **`A Socket must be “Bound” to a local IP/PORT`**
  - Sockets can be bound to a specific address or to any address
    - e.g. 192.168.0.1 (only listens in this address)
    - e.g. 0.0.0.0 (listens in all active addresses and broadcast)
  - bind() method can be used to associate a Socket to a local IP/Port.
    - In Java, socket binding is usually made within the Socket creation methods.

Connectionless Socket



- **DatagramSocket is a socket used for sending and receiving datagram packets over a network via UDP.**
- **A DatagramPacket is sent from a DatagramSocket by calling the send(...) method of DatagramSocket with DatagramPacket as the argument**
  - send(DatagramPacket dp).
- **receive(DatagramPacket dp) is used for receiving a DatagramPacket.**

Address Classes

- **`InetAddress`**
  - This class represents an Internet Protocol (IP) address.
  - **`static InetAddress getByName(String host)`**
    - Method to determine the IP address of a host, given the host's name (or IP address as string).
- **`SocketAddress`**
  - This class represents a Socket Address with no protocol attachment.
  - As an abstract class, it is meant to be subclassed with a specific, protocol dependent, implementation.
- **`InetSocketAddress`**
  - Extends SocketAddress
  - This class implements an IP Socket Address (IP address + port number).
  - It can also be a pair (hostname + port number).

DatagramPacket

- **`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, with size equal to length, to the specified port number on the specified host.
- **`DatagramPacket(byte[] buf, int length, SocketAddress address)`**
  - Constructs a datagram packet for sending packets of length, with size equal to length, to the specified port number on the specified host.

DatagramPacket - Methods

- **`byte[] getData()`**
  - Returns the data buffer.
- **`Void setData(byte[] buf)`**
  - Set the data buffer for this packet.
- **`InetAddress getAddress()`**
  - Returns the IP address of the machine to which this datagram is being sent or from which the datagram was 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.

DatagramSocket

- **`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.
  - Port equal to zero binds it to a random (available) port.
- **`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.

DatagramSocket - Methods

- **`void receive(DatagramPacket p)`**
  - Receives a datagram packet from this socket.
- **`void send(DatagramPacket p)`**
  - Sends a datagram packet from this socket.
- **`int getLocalPort()`**
  - Returns the port number on the local host to which this socket is bound.
- **`void close()`**
  - Closes this datagram socket.

DatagramSockets
Example without Handling Exceptions

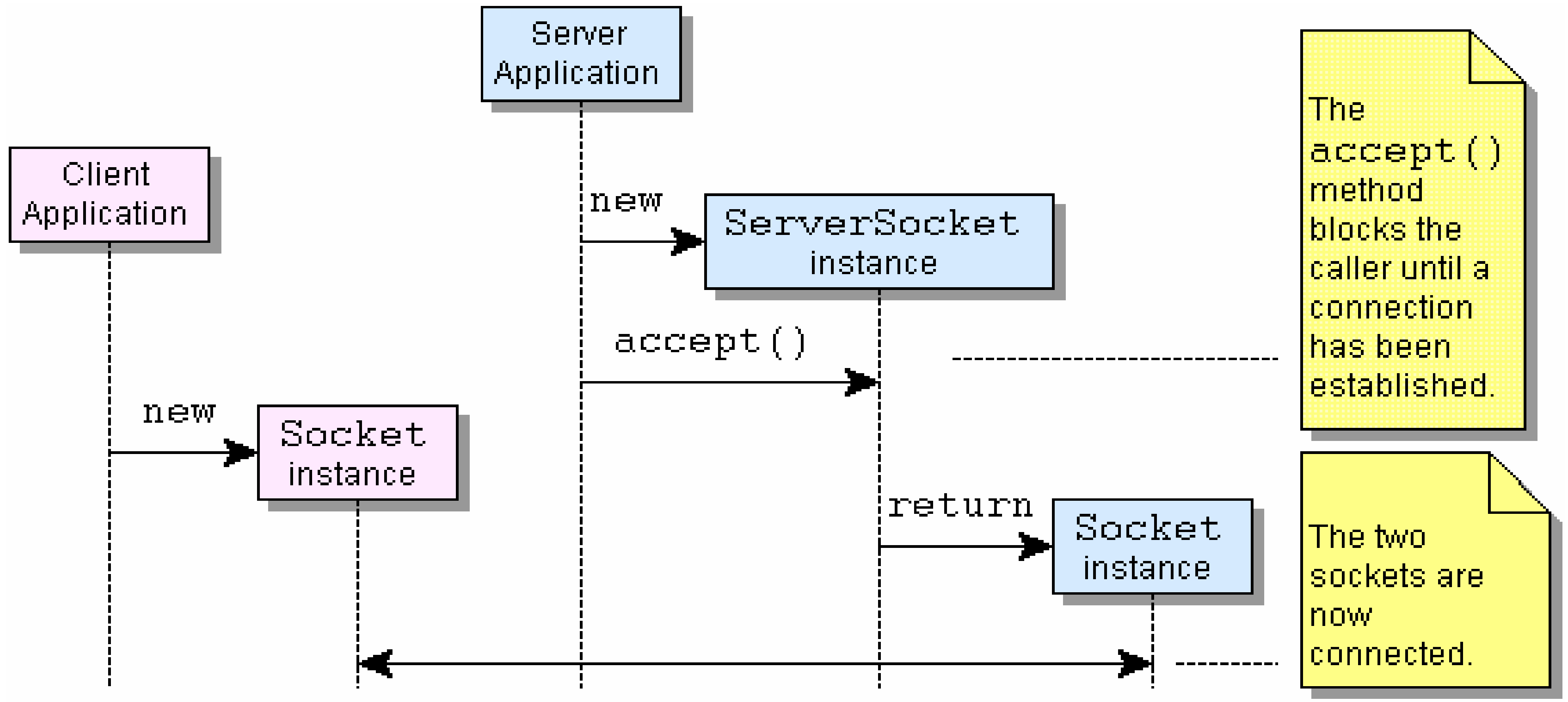
- **Server**

```
DatagramSocket datagramSocket = new DatagramSocket(1234);

byte[] buffer = new byte[1500];
DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

datagramSocket.receive(packet);

Integer nBytes=packet.getLength();
buffer = packet.getData();

String message = new String(buffer,0,nBytes);
InetAddress senderIP = packet.getAddress();
Integer senderPort = packet.getPort();
```

- **Client**

```
DatagramSocket datagramSocket = new DatagramSocket();

String message = new String("Hello\n\0");
byte[] buffer = message.getBytes();

InetAddress receiverIP = InetAddress.getByName("localhost");

DatagramPacket packet = new DatagramPacket(buffer, buffer.length, receiverIP, 1234);

datagramSocket.send(packet);

datagramSocket.close();
```

Connection Oriented Socket


Server Application
Client Application
new
ServerSocket instance
accept()
new
Socket instance
return
Socket instance
The accept() method blocks the caller until a connection has been established.
The two sockets are now connected.

- **`Once the connection is established, getInputStream() and getOutputSteam() may be used in communication between the sockets`**

ServerSocket

- **`ServerSocket()`**
  - Creates an unbound server socket.
- **`ServerSocket(int port)`**
  - Creates a server socket, bound to the specified port.

Methods

- **`void bind(SocketAddress endpoint)`**
  - Binds the ServerSocket to a specific address (IP address and port number).
- **`void close()`**
  - Closes this socket.
- **`Socket accept()`**
  - Listens for a connection to be made to this socket and accepts it.

Socket

- **`Socket()`**
  - Creates an unconnected socket.
- **`Socket(InetAddress address, int port)`**
  - Creates a stream socket and connects it to the specified port number at the specified IP address.
- **`Socket(InetAddress address, int port, InetAddress localAddr, int localPort)`**
  - Creates a socket and connects it to the specified remote address on the specified remote port.
- **`Socket(String host, int port)`**
  - Creates a stream socket and connects it to the specified port number on the named host.
- **`Socket(String host, int port, InetAddress localAddr, int localPort)`**
  - Creates a socket and connects it to the specified remote host on the specified remote port.

Socket - Methods

- **`void bind(SocketAddress bindpoint)`**
  - Binds the socket to a local address.
- **`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.
- **`InputStream getInputStream()`**
  - Returns an input stream for this socket.
- **`OutputStream getOutputStream()`**
  - Returns an output stream for this socket.
- **`void close()`**
  - Closes this socket.

OutputStream/InputStream

- **`OutputStream Methods`**
  - void write(byte[] data)
    - Writes entire array of bytes to the output stream.
  - void write(byte[ ] data, int offset, int length)
    - Writes length bytes from data starting from byte offset.
  - void close()
    - Terminates the stream.
- **`InputStream Methods`**
  - int read(byte[] data)
    - Reads up to data.length bytes (or until the end-of-stream) from the input stream into data and returns the number of bytes read.
    - If no data is available, read () blocks until at least 1 byte can be read or the end-of-stream is detected, indicated by a return of -1.
  - int read(byte[ ] data, int offset, int length)
    - Reads up to length bytes (or until the end-of-stream) from the input stream into data, starting at position offset, and returns the number of bytes read.
    - If no data is available, read() blocks until at least 1 byte can be read or the end-of-stream is detected, indicated by a return of -1.
  - int available()
    - Returns the number of bytes available for input.
  - void close()
    - Terminates the stream.

BufferedReader and InputStreamReader

- **`InputStreamReader`**
  - An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters.
- **`BufferedReader`**
  - Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.
  - BufferedReader(InputStreamReader in)
  - Methods
    - int read()
      - Reads a single character.
    - int read(char[] cbuf, int off, int len)
      - Reads characters into a portion of an array.
    - String readLine()
      - Reads a line of text.

DataOutputStream

- **`DataOutputStream`**
  - A data output stream lets an application write primitive Java data types to an output stream.
  - DataOutputStream(OutputStream out)
  - Methods
    - void write(byte[] b)
      - Writes b.length bytes to this output stream.
    - void writeBytes(String s)
      - Writes out the string to the underlying output stream as a sequence of bytes.

ServerSocket/Socket @ Server

Example without Handling Exceptions and Socket States

```
String messageIn=new String();
String messageOut=new String();

ServerSocket server = new ServerSocket(serverPort);
Socket incomingConnection = new Socket();
incomingConnection = server.accept();

BufferedReader inFromClient = new BufferedReader(new
                              InputStreamReader(incomingConnection.getInputStream()));
DataOutputStream outToClient = new
                              DataOutputStream(incomingConnection.getOutputStream());

while(!messageIn.equals("bye"))
{
    messageIn = inFromClient.readLine();
    //...

    outToClient.write(messageOut.getBytes());
    //...
}
incomingConnection.close();
```

universidade de aveiro

Socket @ Client

Example without Handling Exceptions and Socket States

```
Socket socket = new Socket();
socket.connect(new InetSocketAddress(serverAddr, serverPort), 3000); //timeout 3000ms

DataOutputStream outToServer = new DataOutputStream(socket.getOutputStream());
BufferedReader inFromServer = new BufferedReader(new
                              InputStreamReader(socket.getInputStream()));
String messageOut=new String();
String messageIn=new String();

while(!messageOut.equals("bye\n"))
{
   BufferedReader bufferRead = new BufferedReader(new InputStreamReader(System.in));
   messageOut = bufferRead.readLine()+'\n';
   outToServer.write(messageOut.getBytes());
   //...
   messageIn = inFromServer.readLine();
   //...
}
socket.close();
```

Socket IO / Blocking

- **Socket Operations are Blocking**
  - They block until packet is fully sent, client is accepted, packet is

```
int nRetries, maxRetries = 10;
DatagramPacket msgSend = new DatagramPacket();
DatagramPacket msgRecv1 = new DatagramPacket();
DatagramPacket msgRecv2 = new DatagramPacket();
DatagramSocket socket1 = new DatagramSocket();
DatagramSocket socket2 = new DatagramSocket();
...

socket1.send(msgSend);
socket2.send(msgSend);

socket1.receive(msgRecv1);
socket2.receive(msgRecv2);
```

Sends message
To both sockets

Receives message
from socket1,
then from socket 2

If socket1 doesn't
receive a message
socket2.receive will
never be called!

Socket Timeouts

- **`For DatagramSocket, ServerSocket and Socket`**
  - It is possible to define a period of time for which if not packet/connection arrives the *Socket generates an exception (SocketTimeoutException).
- **`void setSoTimeout(int timeout)`**
  - Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
- **`Example`**

```
try {
   datagramSocket.setSoTimeout(3000); //3 seconds
   datagramSocket.receive(packet);
}catch (SocketTimeoutException toe) {
   //process timeout!!
}
```

Non-Blocking IO

- **Solutions for Non-Blocking IO in Java**
  - Threads
    - Multiple parallel process can be used to process simultaneous connections.
    - Most solutions used (and still use) IO operations with multiple threads.
  - Channels
    - Channels contain Sockets and register in a Selector.
      - DatagramChannel, ServerSocketChannel, SocketChannel
    - The selector waits for input in a set of Channels.

Threads Example

Without Handling Exceptions

```
public void acceptConnections() {
    ServerSocket server = new ServerSocket(1234);
    Socket incomingConnection = null;
    while (true) {
        incomingConnection = server.accept();
        handleConnection(incomingConnection);
    }
}

public void handleConnection(Socket connectionToHandle) {
    new Thread(new ConnectionHandler(connectionToHandle)).start();
}

public class ConnectionHandler implements Runnable{
    Socket socketToHandle;
    public ConnectionHandler(Socket aSocketToHandle) {
        socketToHandle = aSocketToHandle;
    }
    public void run() {
        //process socket socketToHandle
    }
}
```

Channels Example (1)

```
Selector sel = Selector.open();
DatagramChannel dChan = DatagramChannel.open();
ServerSocketChannel sChan = ServerSocketChannel.open();
InetAddress addr = new InetAddress("0.0.0.0",1234);

dChan.configureBlocking(false);
sChan.configureBlocking(false);

InetSocketAddress iaddr = new InetSocketAddress(addr,1234);
dChan.socket().bind(iaddr);
sChan.socket().bind(iaddr);

dChan.register(sel,SelectionKey.OP_READ);
sChan.register(sel,SelectionKey.OP_ACCEPT);
```

Create the two channels

Set channels to Non Blocking IO

Bind sockets

Register channels In selector

Channels Example (2)

```
while(true) {
     sel.select(1000);
     Iterator it = sel.selectedKeys().iterator();
     while(it.hasNext())
     {
          SelectionKey key = (SelectionKey) it.next();
          it.remove();
          if(!key.isValid()) continue;

          if(key.isAcceptable())
          {
               ServerSocketChannel ssc = (ServerSocketChannel) key.channel();
               SocketChannel newClient = ssc.accept();
               newClient.configureBlocking(false);
               newClient.register(sel, SelectionKey.OP_READ);
          }
```

Block until data is available
In any socket

Accept the client waiting.
Add new SocketChannel to selector

Channels Example (3)

```
if(key.isReadable())
{
    if(key.channel().getClass().equals(DatagramChannel.class))
    {
        byte[] buf = new byte[2048];
        DatagramPacket pkt = new DatagramPacket(buf,buf.length);
        DatagramChannel dc = (DatagramChannel) key.channel();
        dc.socket().receive(pkt);
        //Do something with packet
    }else
    if(key.channel().getClass().equals(SocketChannel.class))
    {
        //Do something with SocketChannel
    }
}
```

Actual data is available.
Read it from the socket

References

- **JavaTM 2 Platform**
  - http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html
  - http://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html
  - http://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html
  - http://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html
  - http://docs.oracle.com/javase/7/docs/api/java/net/InetSocketAddress.html
  - http://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html
  - http://docs.oracle.com/javase/7/docs/api/java/nio/channels/SocketChannel.html
- **The JavaTM Tutorials**
  - http://docs.oracle.com/javase/tutorial/