

# Sockets Java

Programação Distribuída / José Marinho

## Endereços IP

- ***java.net.InetAddress***
  - Encapsula endereços IP
  - Permite a resolução de nomes

Método	Objectivo
byte[] <b>getAddress()</b>	Devolve o endereço IP associado ao InetAddress, na ordem <i>most significant byte first</i>
static InetAddress[] <b>getAllByName</b> ( String hostname )	Resolve o nome
static InetAddress <b>getByName</b> ( String hostname )	Resolve o nome
String <b>getHostAddress()</b>	Devolve o endereço IP associado ao InetAddress, no formato <i>dotted decimal</i>
static InetAddress <b>getLocalHost()</b>	Devolve o endereço IP da máquina local
String <b>getHostName()</b>	Devolve o nome associado ao InetAddress
boolean <b>isMulticastAddress()</b>	Determina se o InetAddress é um endereço da classe D

2

Programação Distribuída / José Marinho

## Endereços IP

```
try
{
    // Get the local host
    InetAddress localAddress = InetAddress.getLocalHost();

    System.out.println ("IP address : " + localAddress.getHostAddress() );
}catch (UnknownHostException e){
    System.out.println ("Error - unable to resolve localhost");
}
```

```
try
{
    // Resolve host and get InetAddress
    InetAddress addr = InetAddress.getByName( hostName );
    System.out.println ("IP address : " + addr.getHostAddress() );
    System.out.println ("Hostname : " + addr.getHostName() );
}catch (UnknownHostException e){
    System.out.println ("Error - unable to resolve hostname" );
}
```

3

Programação Distribuída / José Marinho

## Protocolo UDP

- ***java.net.DatagramPacket***

- Construção de um datagrama UDP para envio

***DatagramPacket**(byte[] **buffer**, int **length**, **InetAddress** **dest\_addr**, int **dest\_port**)*

```
InetAddress addr = InetAddress.getByName("192.168.0.1");
byte[] data = new byte[128];

//Fill the array with the data to be sent
//...

DatagramPacket packet = new DatagramPacket ( data, data.length, addr,
2000);
```

- Construção de um datagrama UDP para recepção

***DatagramPacket**(byte[] **buffer**, int **length**)*

```
DatagramPacket packet = new DatagramPacket(new byte[256], 256);
```

4

Programação Distribuída / José Marinho

## Protocolo UDP

- ***java.net.DatagramSocket***
  - Operações sobre sockets UDP podem gerar excepções do tipo ***java.net.SocketException***
  - Operações de envio e recepção podem gerar excepções do tipo ***java.io.IOException***
  - Criação de um socket UDP cliente (porto local automático)  
***DatagramSocket()***
  - Criação de um socket UDP servidor  
***DatagramSocket(int port)*** throws ***java.net.SocketException***

5

Programação Distribuída / José Marinho

## Protocolo UDP

Método	Objectivo
void <b>send</b> (DatagramPacket packet)	Envia um datagrama UDP
void <b>receive</b> (DatagramPacket packet)	Recebe um datagrama UDP e armazena-o no DatagramPacket
void <b>close</b> ()	Fecha o socket e liberta o porto local
InetAddress <b>getLocalAddress</b> ()	Devolve o endereço local associado ao socket
int <b>getLocalPort</b> ()	Devolve o porto local associado ao socket
void <b>setReceiveBufferSize</b> (int length)	Especifica o tamanho máximo do buffer de recepção
int <b>getReceiveBufferSize</b> ()	Devolve o tamanho máximo para o buffer de recepção
void <b>setSendBufferSize</b> (int length)	Especifica o tamanho máximo do buffer de envio
int <b>getSendBufferSize</b> ()	Devolve o tamanho máximo para o buffer de envio
void <b>setSoTimeout</b> (int duration)	Especifica o valor do timeout de recepção em milissegundos (poderá dar origem a excepções do tipo <b><i>java.io.InterruptedIOException</i></b> )
int <b>getSoTimeout</b> ()	Devolve o valor do timeout de recepção (zero significa sem timeout)

6

Programação Distribuída / José Marinho

## Protocolo UDP

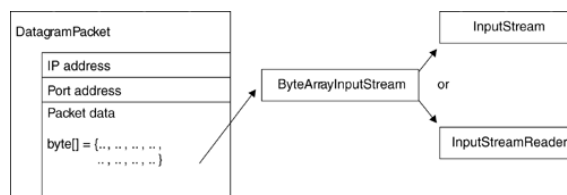
```
DatagramPacket packet = new DatagramPacket (new byte[256], 256);  
DatagramSocket socket = new DatagramSocket(2000);  
boolean finished = false;  
  
while (!finished )  
{  
    socket.receive (packet);  
    // process the packet  
    //...  
}  
socket.close ();
```

- Processar directamente um *array* de *bytes* pode não ser a forma mais adequada/prática
- A solução passa por usar fluxos de entrada baseados na classe *java.io.InputStream* ou *java.io.InputStreamReader*, assumindo o *array* como dispositivo de entrada

7

Programação Distribuída / José Marinho

## Protocolo UDP



```
ByteArrayInputStream bin = new ByteArrayInputStream(packet.getData());  
DataInputStream din = new DataInputStream (bin);  
  
// Read the contents of the UDP packet  
// ...
```

8

Programação Distribuída / José Marinho

# Protocolo UDP

```
DatagramSocket socket = new DatagramSocket();

DatagramPacket packet = new DatagramPacket(new byte[256], 256);
packet.setAddress ( InetAddress.getByName ( somehost ) );
packet.setPort( 2000 );

boolean finished = false;

while !finished )
{
    // Write data to packet buffer
    // ...
    socket.send(packet);

    // Do something else, like read other packets, or check to
    // see if no more packets to send
    // ...
}
socket.close();
```

9

Programação Distribuída / José Marinho

# Protocolo UDP

- Um exemplo completo
  - Cliente (envio de um datagrama)

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

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

        // CHECK FOR VALID NUMBER OF PARAMETERS
        int argc = args.length;
        if (argc != 1){
            System.out.println ("Syntax :");
            System.out.println ("java PacketSendDemo hostname");
            return;
        }

        String hostname = args[0];
```

10

Programação Distribuída / José Marinho

## Protocolo UDP

```
try{
    System.out.println ("Binding to a local port");
    // CREATE A DATAGRAM SOCKET, BOUND TO ANY AVAILABLE LOCAL PORT
    DatagramSocket socket = new DatagramSocket();
    System.out.println ("Bound to local port " + socket.getLocalPort());

    // CREATE A MESSAGE TO SEND USING A UDP PACKET
    ByteArrayOutputStream bout = new ByteArrayOutputStream();
    PrintStream pout = new PrintStream (bout);
    pout.print ("Greetings!");

    // GET THE CONTENTS OF OUR MESSAGE AS AN ARRAY OF BYTES
    byte[] barray = bout.toByteArray();

    // CREATE A DATAGRAM PACKET, CONTAINING OUR BYTE ARRAY
    DatagramPacket packet = new DatagramPacket( barray, barray.length );
    System.out.println ("Looking up hostname " + hostname );
```

Abordagem mais directa:

```
byte[] barray = "Greetings!".getBytes();
```

11

Programação Distribuída / José Marinho

## Protocolo UDP

```
// LOOKUP THE SPECIFIED HOSTNAME, AND GET AN InetAddress
InetAddress addr = InetAddress.getByName(hostname);
System.out.println ("Hostname resolved as "+addr.getHostAddress());
// ADDRESS PACKET TO SENDER
packet.setAddress(addr);

// SET PORT NUMBER TO 2000
packet.setPort(2000);

// SEND THE PACKET - REMEMBER NO GUARANTEE OF DELIVERY
socket.send(packet);
System.out.println ("Packet sent!");

}catch (UnknownHostException e){
    System.err.println ("Can't find host " + hostname);
}catch (IOException e){
    System.err.println ("Error - " + e);
}
}
```

12

Programação Distribuída / José Marinho

# Protocolo UDP

- Servidor (recepção de um datagrama)

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

public class PacketReceiveDemo{

    public static void main (String args[]){

        try{
            System.out.println ("Binding to local port 2000");

            // CREATE A DATAGRAM SOCKET, BOUND TO THE SPECIFIC PORT 2000
            DatagramSocket socket = new DatagramSocket(2000);

            // CREATE A DATAGRAM PACKET WITH A MAXIMUM BUFFER OF 256 BYTES
            DatagramPacket packet = new DatagramPacket(new byte[256], 256);

            // RECEIVE A PACKET (BY DEFAULT, THIS IS A BLOCKING OPERATION)
            socket.receive(packet);
```

13

Programação Distribuída / José Marinho

# Protocolo UDP

```
// DISPLAY PACKET INFORMATION
InetAddress remote_addr = packet.getAddress();
System.out.println("Sent by: " + remote_addr.getHostAddress());
System.out.println ("Sent from port: " + packet.getPort());

// DISPLAY PACKET CONTENTS, BY READING FROM BYTE ARRAY
ByteArrayInputStream bin = new ByteArrayInputStream(packet.getData());
int data;
while ((data = bin.read()) != -1)
    System.out.print((char) data);

socket.close();

}catch (IOException e){
    System.err.println ("Error - " + e);
}
}
```

Abordagem mais directa:

```
String msg = new String(packet.getData(), 0, packet.getLength());
System.out.println(msg);
```

14

Programação Distribuída / José Marinho

# Protocolo UDP

- Outro exemplo
  - Servidor de eco

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

public class EchoServer
{
    // UDP PORT TO WHICH SERVICE IS BOUND
    public static final int SERVICE_PORT = 7000;

    // MAX SIZE OF PACKET, LARGE ENOUGH FOR ALMOST ANY CLIENT
    public static final int BUFSIZE = 4096;

    // SOCKET USED FOR READING AND WRITING UDP PACKETS
    private DatagramSocket socket;
```

15

Programação Distribuída / José Marinho

# Protocolo UDP

```
public EchoServer() //constructor
{
    try
    {
        socket = null;
        // BIND TO THE SPECIFIED UDP PORT
        socket = new DatagramSocket( SERVICE_PORT );
        System.out.println("Server active on port "+socket.getLocalPort());

    }catch (Exception e){
        System.err.println ("Unable to bind port");
    }
}

public void serviceClients()
{
    if(socket == null) return;

    // CREATE A BUFFER LARGE ENOUGH FOR INCOMING PACKETS
    byte[] buffer = new byte[BUFSIZE];
```

16

Programação Distribuída / José Marinho



## Protocolo UDP

```
for (;;) { //while(true)
    try {
        // CREATE A DATAGRAMPACKET FOR READING UDP PACKETS
        DatagramPacket packet = new DatagramPacket ( buffer, BUFSIZE );

        // RECEIVE INCOMING PACKETS
        socket.receive(packet);

        System.out.println("Packet received from " + packet.getAddress()
+ ":" + packet.getPort() + " of length " + packet.getLength());

        // ECHO THE PACKET BACK - ADDRESS AND PORT ARE ALREADY SET!
        socket.send(packet);

    } catch (IOException e) {
        System.err.println ("Error : " + e);
    }

} // for(;;)
} // serviceClientes() method
```

17

Programação Distribuída / José Marinho

## Protocolo UDP

```
public static void main(String args[])
{
    EchoServer server = new EchoServer();
    server.serviceClients();
}
```

- Cliente de eco

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

public class EchoClient
{
    // UDP PORT TO WHICH SERVICE IS BOUND
    public static final int SERVICE_PORT = 7000;

    // MAX SIZE OF PACKET
    public static final int BUFSIZE = 256;
```

18

Programação Distribuída / José Marinho

# Protocolo UDP

```
public static void main(String args[])
{
    if (args.length != 1){
        System.err.println ("Syntax - java EchoClient hostname");
        return;
    }
    String hostname = args[0];

    // GET AN INETADDRESS FOR THE SPECIFIED HOSTNAME
    InetAddress addr = null;
    try{

        // RESOLVE THE HOSTNAME TO AN INETADDR
        addr = InetAddress.getByName(hostname);

    }catch (UnknownHostException e){
        System.err.println ("Unable to resolve host");
        return;
    }
}
```

19

Programação Distribuída / José Marinho

# Protocolo UDP

```
try {
    // BIND TO ANY FREE PORT
    DatagramSocket socket = new DatagramSocket();

    // SET A TIMEOUT VALUE OF TWO SECONDS
    socket.setSoTimeout (2 * 1000);

    for (int i = 1 ; i <= 10; i++){
        // COPY SOME DATA TO OUR PACKET
        String message = "Packet number " + i ;
        char[] cArray = message.toCharArray();

        byte[] sendbuf = new byte[cArray.length];
        for (int offset = 0; offset < cArray.length ; offset++) {
            sendbuf[offset] = (byte) cArray[offset];
        }
    }
}
```

Abordagem mais directa: `sendbuf = message.getBytes();`

20

Programação Distribuída / José Marinho

## Protocolo UDP

```
// CREATE A PACKET TO SEND TO THE UDP SERVER
DatagramPacket sendPacket = new DatagramPacket(sendbuf,
                                                sendbuf.length, addr, SERVICE_PORT);
System.out.println ("Sending packet to " + hostname);

// SEND THE PACKET
socket.send (sendPacket);

System.out.print ("Waiting for packet.... ");

// CREATE A SMALL PACKET FOR RECEIVING UDP PACKETS
byte[] recbuf = new byte[BUFSIZE];
DatagramPacket receivePacket=new DatagramPacket(recbuf,BUFSIZE);

// DECLARE A TIMEOUT FLAG
boolean timeout = false;
```

21

Programação Distribuída / José Marinho

## Protocolo UDP

```
// CATCH ANY INTERRUPTEDIOEXCEPTION THAT IS THROWN WHILE WAITING A UDP PKT
try{
    socket.receive (receivePacket);
}catch (InterruptedException e){
    timeout = true;
}

if (!timeout){

    System.out.println ("packet received!");
    System.out.println ("Details : "+receivePacket.getAddress());

    // OBTAIN A BYTE INPUT STREAM TO READ THE UDP PACKET
    ByteArrayInputStream bin = new ByteArrayInputStream (
        receivePacket.getData(), 0, receivePacket.getLength() );

    // CONNECT A READER FOR EASIER ACCESS
    BufferedReader reader = new BufferedReader (
        new InputStreamReader ( bin ) );
```

22

Programação Distribuída / José Marinho

# Protocolo UDP

```
// LOOP INDEFINITELY UNTIL EVERY LINE OF TEXT WAS DISPLAYED
for (;;)
{
    String line = reader.readLine();

    // CHECK FOR END OF DATA
    if (line == null)
        break;
    else
        System.out.println (line);
}

}else{ // TIMEOUT HAS OCCURED
    System.out.println ("packet lost!");
}
```

Abordagem mais directa:

```
String msg = new String(receivePacket.getData(), 0, receivePacket.getLength());
System.out.println(msg);
```

23

Programação Distribuída / José Marinho

# Protocolo UDP

```
// SLEEP FOR A SECOND, TO ALLOW USER TO SEE PACKET
try{

    Thread.sleep(1000);

}catch (InterruptedException e) {}

} // for (int i = 1 ; i <= 10; i++)

}catch (IOException e){
    System.err.println ("Socket error " + e);
}

} // main(String args[])
}
```

24

Programação Distribuída / José Marinho

## Protocolo TCP

- Orientado a ligação (ligações virtuais)
- Apenas permite comunicações ponto-a-ponto
- Dados tratados como fluxos contínuos de *bytes*, à semelhança de *input* e *output streams* ( $\neq$  datagramas)
- Entrega de dados ordenada, sem duplicações e livre de erros
- Na perspectiva do programador, é mais simples do que o UDP quando existem requisitos de fiabilidade
- ***java.net.Socket***: para transferência de *bytes*
- ***java.net.ServerSocket***: para aceitação de pedidos de ligação

25

Programação Distribuída / José Marinho

## Protocolo TCP

- ***java.net.Socket***

- Construtores

*Socket (String host, int port)*

*Socket (InetAddress address, int port)*

*Socket (InetAddress address, int port, InetAddress bindAddress, int localPort)*

*Socket (String host, int port, InetAddress bindAddress, int localPort)*

```
try{
    // CONNECT TO THE SPECIFIED HOST AND PORT
    Socket mySocket = new Socket ( "www.awl.com", 80);
    // ...
}catch (Exception e){
    System.err.println ("Err - " + e);
}
```

26

Programação Distribuída / José Marinho

# Protocolo TCP

Método	Objectivo
OutputStream <b>getOutputStream()</b>	Devolve uma <i>stream</i> de saída que permite enviar dados para uma ligação TCP
InputStream <b>getInputStream()</b>	Devolve uma <i>stream</i> de entrada que permite receber dados de uma ligação TCP
void <b>close()</b>	Fecha uma ligação
InetAddress <b>getLocalAddress()</b>	Devolve o endereço associado ao socket local
int <b>getLocalPort()</b>	Devolve o porto ao qual se encontra associado o socket local
InetAddress <b>getInetAddress()</b>	Devolve o endereço da máquina remota
int <b>getPort()</b>	Devolve o porto remoto associado ao socket
void <b>setSoTimeout</b> (int duration)	Especifica o valor do <i>timeout</i> de recepção em milissegundos
int <b>getSoTimeout()</b>	Devolve o valor do <i>timeout</i> de recepção (zero significa sem <i>timeout</i> )
void <b>setTcpNoDelay</b> (boolean onFlag)	Activa ou desactiva a opção TCP_NODELAY
boolean <b>getTcpNoDelay()</b>	Devolve o estado da opção TCP_NODELAY (ver o Algoritmo de Nagle)
void <b>shutdownInput()</b>	Encerra a <i>stream</i> de entrada associada à ligação TCP
void <b>shutdownOutput()</b>	Encerra a <i>stream</i> de saída associada à ligação TCP

27

Programação Distribuída / José Marinho

# Protocolo TCP

- Enviar e receber dados através de uma ligação TCP

```
try{

    // CONNECT A SOCKET TO SOME HOST MACHINE AND PORT
    Socket socket = new Socket( somehost, someport );

    // CONNECT A BUFFERED READER
    BufferedReader reader = new BufferedReader(new InputStreamReader(
                                                socket.getInputStream()));

    // CONNECT A PRINT STREAM
    PrintStream pstream =new PrintStream( socket.getOutputStream() );

}catch (Exception e){
    System.err.println("Error - " + e);
}
```

28

Programação Distribuída / José Marinho

## Protocolo TCP

- Os *timeouts* de recepção geram exceções do tipo *java.io.InterruptedIOException* (subclasse de *java.io.IOException* )

```
try{
    Socket s = new Socket (...);
    s.setSoTimeout ( 2000 );

    // DO SOME READ OPERATION ....

}catch (InterruptedException e){

    timeoutFlag = true; // DO SOMETHING SPECIAL LIKE SET A FLAG

}catch (IOException e){

    System.err.println ("IO error " + e);
    System.exit(0);

}
```

29

Programação Distribuída / José Marinho

## Protocolo TCP

- *java.net.ServerSocket*
  - Construtores
    - ServerSocket(int port)*
    - ServerSocket(int port, int backlog)*
    - ServerSocket(int port, int backlog, InetAddress bindAddress)*

Método	Objectivo
Socket <b>accept()</b>	Aguarda por um pedido de ligação e aceita-o. Por omissão, é uma operação bloqueante.
void <b>close()</b>	Fecha o socket servidor (i.e., de escuta)
int <b>getLocalPort()</b>	Devolve o porto ao qual se encontra associado o socket servidor
InetAddress <b>getInetAddress()</b>	Devolve o endereço associado ao socket servidor
void <b>setSoTimeout(int duration)</b>	Especifica o valor de <i>timeout</i> do socket servidor em milissegundos
int <b>getSoTimeout()</b>	Devolve o valor de <i>timeout</i> do socket servidor (zero significa sem <i>timeout</i> )

30

Programação Distribuída / José Marinho

# Protocolo TCP

- Um exemplo concreto (serviço *Day Time*)
- Cliente

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

public class DaytimeClient
{
    public static final int SERVICE_PORT = 5001;

    public static void main(String args[])
    {
        if (args.length != 1){
            System.out.println ("Syntax - java DaytimeClient host");
            return;
        }
        // GET THE HOSTNAME OF SERVER
        String hostname = args[0];

        try {
```

31

Programação Distribuída / José Marinho

# Protocolo TCP

```
Socket daytime = new Socket (hostname, SERVICE_PORT);
System.out.println ("Connection established");

// SET THE SOCKET OPTION JUST IN CASE SERVER STALLS
daytime.setSoTimeout( 2000 ); //ms

// READ FROM THE SERVER
BufferedReader reader = new BufferedReader(
    new InputStreamReader(daytime.getInputStream()));
System.out.println ("Results : " + reader.readLine());

// CLOSE THE CONNECTION
daytime.close();

} catch (IOException e){ //catches also InterruptedException
    System.err.println ("Error " + e);
}
}
```

32

Programação Distribuída / José Marinho



# Protocolo TCP

## • Servidor

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

public class DaytimeServer
{
    public static final int SERVICE_PORT = 5001;

    public static void main(String args[])
    {
        try {
            // BIND TO THE SERVICE PORT
            ServerSocket server = new ServerSocket(SERVICE_PORT);

            System.out.println ("Daytime service started");

            // LOOP INDEFINITELY, ACCEPTING CLIENTS
            for (;;) {
                // GET THE NEXT TCP CLIENT
                Socket nextClient = server.accept();
```

33

Programação Distribuída / José Marinho

# Protocolo TCP

```
        System.out.println ("Received request from " +
            nextClient.getInetAddress() + ":" + nextClient.getPort() );

        OutputStream out = nextClient.getOutputStream();
        PrintStream pout = new PrintStream (out);
        // WRITE THE CURRENT DATE OUT TO THE USER
        pout.println(new java.util.Date());
        // FLUSH UNSENT BYTES
        pout.flush();
        // CLOSE THE CONNECTION
        nextClient.close();
    }

    } catch (BindException e) {
        System.err.println("Service already running on port " +
            SERVICE_PORT);
    } catch (IOException e) {
        System.err.println ("I/O error - " + e);
    }
}
```

34

Programação Distribuída / José Marinho

## Protocolo TCP

- Excepções
  - *java.net.SocketException* representa erros genéricos associados aos sockets (ver protocolo UDP)
  - Subclasses de *java.net.SocketException*

Classe	Significado
<b>BindException</b>	Impossibilidade de associação ao porto local. Possivelmente, o porto já estará associado a outro socket.
<b>ConnectException</b>	Não é possível estabelecer a ligação com o destino pretendido (porto não associado no destino, etc.).
<b>NoRouteToHostException</b>	Não é possível encontrar um caminho até ao destino, devido a um erro de rede
<b>InterruptedIOException</b>	Uma operação de leitura esteve bloqueada durante o tempo limite definido ( <i>setSoTimeout(int duration)</i> )

35

Programação Distribuída / José Marinho

## Aplicações *Multicast*

- O protocolo UDP permite o envio de datagramas para endereços de difusão e de *multicast*
- Com endereços IP do tipo difusão, todas as máquinas do domínio de difusão recebem os datagramas
- Com endereços IP do tipo *multicast* (classe D), apenas recebem os datagramas as máquinas que se tenham registado no respectivo grupo/endereço
- Classe: ***MulticastSocket*** (subclasse de *DatagramSocket*)

36

Programação Distribuída / José Marinho

## Aplicações *Multicast*

```
InetAddress group = InetAddress.getByName("224.1.2.3");
MulticastSocket socket = new MulticastSocket(port);
socket.joinGroup(group);
Socket.setTimeToLive(1); //TTL

byte[] buffer = new byte[1024];

//...

DatagramPacket packet = new DatagramPacket(buffer, buffer.length, group, port);
socket.send(packet);

//...

byte[] response = new byte[1024];
DatagramPacket packet = new DatagramPacket(response, response.length);
socket.receive(packet);

//...

socket.leaveGroup(group);
```

37

Programação Distribuída / José Marinho

## Serialização de objectos

- Ligações TCP

```
s = new Socket(serverAddr, serverPort);

//TRANSMIT OBJECT

in = new ObjectInputStream(s.getInputStream());
out = new ObjectOutputStream(s.getOutputStream());

out.writeObject(objectToTransmit);
//out.writeUnshared(objectToTransmit) in order to avoid caching issues

out.flush();

//RECEIVE OBJECT

returnedObject = (MyClass)in.readObject();
```

38

Programação Distribuída / José Marinho

# Serialização de objectos

- Datagramas UDP

```
s = new DatagramSocket();

//TRANSMIT OBJECT

bOut = new ByteArrayOutputStream();
out = new ObjectOutputStream(bOut);

out.writeObject(objectToTransmit);
//out.writeUnshared(objectToTransmit) in order to avoid caching issues

out.flush();

packet = new DatagramPacket(bOut.toByteArray(), bOut.size(), serverAddr,
                             serverPort);

s.send(packet);
```

39

Programação Distribuída / José Marinho

# Serialização de objectos

- Datagramas UDP

```
//RECEIVE OBJECT

packet = new DatagramPacket(new byte[MAX_SIZE], MAX_SIZE);

s.receive(packet);

in = new ObjectInputStream(new ByteArrayInputStream(packet.getData(), 0,
                                                       packet.getLength()));

returnedObject = (MyClass)in.readObject();
```

40

Programação Distribuída / José Marinho

## Bibliografia

- REILLY, David; REILLY, Michael - *Java Network Programming & Distributed Computing* - Addison-Wesley
- <http://download.oracle.com/javase/tutorial/essential/>