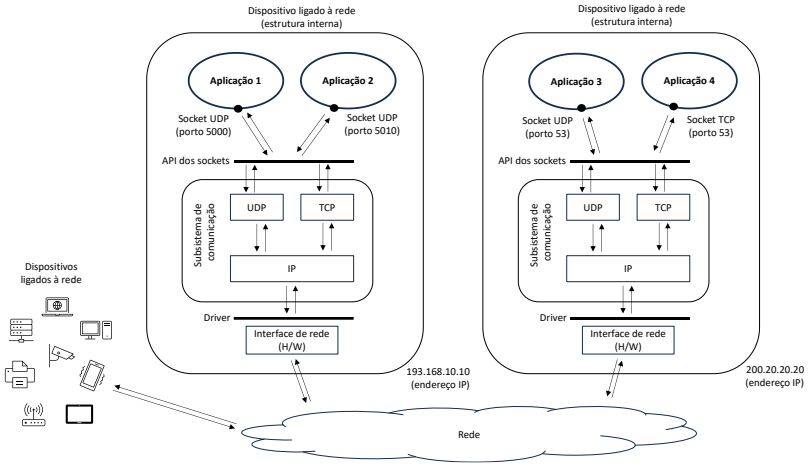
Sockets Java

Introdução

- Para desenvolver aplicações distribuídas que recorrem aos protocolos UDP e TCP da pilha protocolar TCP/IP, são geralmente usadas API (*Application Programming Interfaces*) baseadas no conceito de *Socket*
- Esta abstração representa um ponto de comunicação através do qual é possível enviar e receber mensagens (cabeçalho + dados) através de uma rede
- Linguagens de programação e sistemas distintos possuem API próprias
- Linguagem Java: *package* "java.net"

Introdução



Endereços IP

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

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

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");
}
```

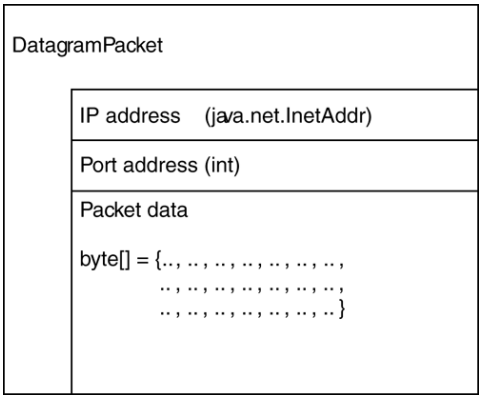
```
try
{
         // Resolve host and get InetAddress
         InetAddress addr = InetAddress.getByName( hostName );
         System.out.println ("IP address : " + addr.getHostAddress() );
         System.out.println ("Hostname : " + addr.getHostName() );

}catch (UnknownHostException e){
         System.out.println ("Error - unable to resolve hostname" );
}
```


Protocolo UDP

- Um dos dois protocolos da camada de transporte da pilha protocolar TCP/IP
- Não orientado a ligação
- Envio e receção de pacotes independentes designados *datagramas*
- Não garante a entrega ordenada, sem duplicação e sem perdas dos dados
- Sobrecarga protocolar reduzida
- Suporta o envio por difusão
- ***java.net.DatagramPacket***: representa um *datagrama*
- ***java.net.DatagramSocket***: envio e receção de *datagramas*

Protocolo UDP

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



https://www.oreilly.com/library/view/javatm-network-programming/0201710374/0201710374_ch05lev1sec2.html

Protocolo UDP

- 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 **receção**

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

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

Protocolo UDP

- ***java.net.DatagramSocket***
  - Operações sobre sockets UDP podem gerar exceções do tipo ***java.net.SocketException***
  - Operações de envio e receção podem gerar exceções 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***

Protocolo UDP

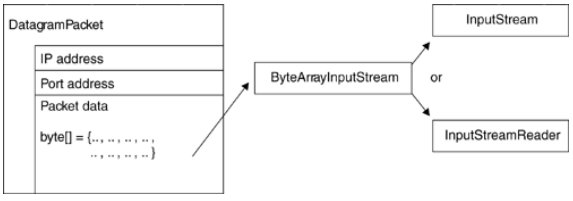
Método	Objetivo
void **send**(DatagramPacket packet)	Envia um datagrama UDP
void **receive**(DatagramPacket packet)	Recebe um datagrama UDP e armazena-o no DatagramPacket
void **close**()	Fecha o socket e liberta o porto loca
InetAddress **getLocalAddress**()	Devolve o endereço local associado ao socket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

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

Protocolo UDP




```
ByteArrayInputStream bin = new ByteArrayInputStream(packet.getData());
DataInputStream din = new DataInputStream(bin);

// Read the contents of the UDP packet
// ...
```

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 and set data length
    // ...
    socket.send(packet);

    // Do something else, like read other packets, or check to
    // see if no more packets to send
    // ...
}

socket.close();
```


Protocolo UDP

- Exemplo 1
  - 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];
```

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 (ARRAY OF BYTES)
            byte[] barray = "Greetings!".getBytes();

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

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


Protocolo UDP

```
            // 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!");

            socket.close();

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

Protocolo UDP

- Receção 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);
```


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());

         // CREATE AND DISPLAY A STRING BASED ON PACKET CONTENTS
         String msg = new String(packet.getData(), 0, packet.getLength());
         System.out.println(msg);

         socket.close();

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

Protocolo UDP

- Exemplo 2
  - Servidor de eco (reenvia o conteúdo recebido à origem)

```
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 = null;
```


Protocolo UDP

```
    public EchoServer() //constructor
    {
        try
        {
            // 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];
```

Protocolo UDP

```
        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);
            }

        } // while
    } // serviceClientes() method
```


Protocolo UDP

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

- Cliente de eco (envia *datagrama* e aguarda receção de volta)

```
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;
```

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;
    }
```


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 BYTE ARRAY
        String message = "Packet number " + i ;
        byte[] sendbuf = message.getBytes();

        // CREATE A PACKET TO SEND TO THE UDP SERVER
        DatagramPacket sendPacket = new DatagramPacket(sendbuf,
                                                sendbuf.length, addr,
                                                SERVICE_PORT);
```

Protocolo UDP

```
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;
```


Protocolo UDP

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

if (!timeout){

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

   // CREATE AND DISPLAY A STRING BASED ON PACKET CONTENT
   String msg = new String(receivePacket.getData(), 0,
                           receivePacket.getLength());
   System.out.println(msg);

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

Protocolo UDP

```
            // SLEEP FOR A SECOND, TO ALLOW USER TO SEE PACKET
            try{
               Thread.sleep(1000);
            }catch (InterruptedException e) {}

         } // for

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

   } // main

}
```


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* (≠ datagramas UDP)
- Entrega de dados ordenada, sem duplicações e livre de erros
- Na perspetiva do programador, é mais simples do que o UDP quando existem requisitos de fiabilidade
- ***java.net.Socket***: envio e receção de *bytes*
- ***java.net.ServerSocket***: aceitação de pedidos de ligação

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);
}
```


Protocolo TCP

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

Protocolo TCP

Método	Objetivo
OutputStream **getOutputStream**()	Devolve uma *stream* de saída que permite enviar dados para uma ligação TCP
InputStream **getInputStream**()	Devolve uma *stream* de entrada que permite receber dados de uma ligação TCP
void **close**()	Fecha a ligação
InetAddress **getLocalAddress**()	Devolve o endereço IP local associado à ligação
int **getLocalPort**()	Devolve o porto local associado à ligação
InetAddress **getInetAddress**()	Devolve o endereço IP remoto associado à ligação
int **getPort**()	Devolve o porto remoto associado à ligação
void **setSoTimeout**(int timeout)	Define o valor do timeout para operações de leitura
int **getSoTimeout**()	Devolve o valor do timeout para operações de leitura
void **setTcpNoDelay**(boolean on)	Ativa/desativa o algoritmo de Nagle
boolean **getTcpNoDelay**()	Verifica se o algoritmo de Nagle está ativo
void **shutdownInput**()	Encerra a *stream* de entrada associada à ligação TCP
void **shutdownOutput**()	Encerra a stream de saída associada à ligação TCP

Classe abstrata `java.io.InputStream`	Classe abstrata `java.io.OutputStream`
int read()	void write(byte[] b)
int read(byte[] b)	void write(byte[] b, int off, int len)
int read(byte[] b, int off, int len)	void write(int b)
.	.
.	.
.	.

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);
}
```

Protocolo TCP

- Os *timeouts* de receção 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 (InterruptedIOException e){

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

}catch (IOException e){

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

}
```


Protocolo TCP

- ***java.net.ServerSocket***
  - Construtores

    ServerSocket(int port)

    ServerSocket(int port, int backlog)

    ServerSocket(int port, int backlog, InetAddress bindAddress)

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

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 {
```


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 InterruptedIOException
         System.err.println ("Error " + e);
      }
   }
}
```

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
            while(true){
                // GET THE NEXT TCP CLIENT
                Socket nextClient = server.accept();
```


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);
        }
    }
}
```

Protocolo TCP

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

Classe	Significado
BindException	Impossibilidade de associação ao porto local. Possivelmente, o porto já estará associado a outro socket.
ConnectException	Não é possível estabelecer a ligação com o destino pretendido (porto não associado no destino, etc.).
NoRouteToHostException	Não é possível encontrar um caminho até ao destino, devido a um erro de rede
PortUnreachableException	Foi recebida uma mensagem ICMP de porto não alcançável

Try-with-resources

- Preferível à abordagem *try-finally* quando se usam recursos que devem ser encerrados depois de utilizados (implementam a interface *java.io.Closeable*)
- Encerramento automático quando ocorre a saída do bloco
- Código mais conciso e claro, facilitando a sua gestão

```
try(DatagramSocket socket = new DatagramSocket(2000)){
   //. . .
}catch (Exception e){/*. . .*/}

try(Socket socket = new Socket("www.isec.pt", 80)){/*. . .*/}

try(Socket socket = new Socket(serverAddr, serverPort);
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true)){
        /*. . .*/
}
```

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 respetivo grupo/endereço
- Classe: ***MulticastSocket*** (subclasse de *DatagramSocket*)

Aplicações *Multicast*

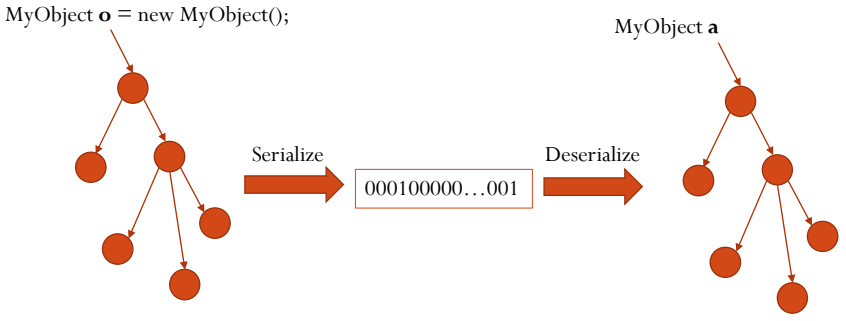
```
InetAddress group = InetAddress.getByName("224.1.2.3");
MulticastSocket socket = new MulticastSocket(port);
socket.joinGroup(group); //Deprecated
Socket.setTimeToLive(1); //TTL

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); //Deprecated
```

Aplicações *Multicast*

- É aconselhável usar os métodos ***joinGroup*** e ***leaveGroup*** que especificam a interface de rede (NIC) associada ao grupo

```
String NicId = ... //e.g., "127.0.0.1", "lo", "en0", "eth0", "10.1.1.1", ...
NetworkInterface nif;

try{
    nif = NetworkInterface.getByInetAddress(InetAddress.getByName(NicId));
    //e.g., 127.0.0.1, 192.168.10.1, ...
}catch (Exception ex){
    nif = NetworkInterface.getByName(NicId); //e.g., lo, eth0, wlan0, en0, ...
}

socket = new MulticastSocket(port);
socket.joinGroup(new InetSocketAddress(group, port), nif);

//socket.setNetworkInterface(nif);
//socket.joinGroup(group);
```


Serialização de objetos



Serialização de objetos

- 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();
```


Serialização de objetos

- Datagramas UDP

```
s = new DatagramSocket();

//TRANSMIT OBJECT

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

out.writeObject(objectToTransmit);
//out.writeUnshared(objectToTransmit) //for avoiding caching issues

out.flush();

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

s.send(packet);
```

Serialização de objetos

- 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();
```


Bibliografia

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