

2° Esercitazione (svolta):

Socket Java senza connessione

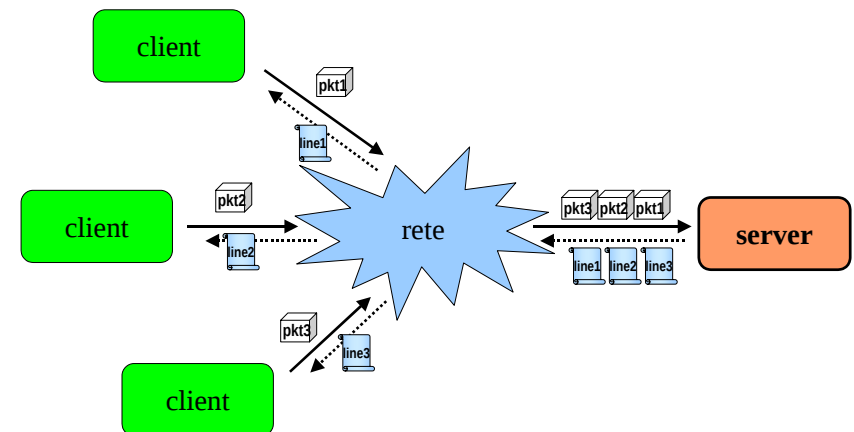
Sviluppare un'applicazione C/S in cui:

- il **client**, invocato con l'indirizzo IP e la porta sulla quale il server è in ascolto, invia al server pacchetti contenenti il nome del file e il numero della linea del file che vuole ricevere, che sono richiesti all'utente usando l'input da console. Successivamente il client stampa il contenuto del pacchetto ottenuto in risposta, che può essere la linea richiesta o una risposta negativa. Per evitare che, in caso di caduta del server o perdita di un pacchetto, il client si blocchi indefinitamente nell'attesa della risposta è previsto un timeout di 30 s.

Per il settaggio del timeout sull'operazione di lettura si faccia uso della primitiva `setTimeout`, invocata sulla socket dopo la costruzione della stessa. Attraverso tale primitiva è infatti possibile evitare la condizione di blocco indefinito qualora si invochino operazioni bloccanti sul di una socket. Le primitive bloccanti saranno infatti sbloccate dopo un tempo pari al timeout settato.

- il **server** verifica l'esistenza del file richiesto. Se il file non esiste notifica l'errore, altrimenti tenta di estrarre la linea richiesta. Se la linea non esiste invia una notifica di errore. Altrimenti invia al client (o ai client) un pacchetto contenente la linea richiesta.

Client e Server Datagram



NOTA

Si noti che il server è realizzato come **server sequenziale** (e quindi mono-processo), che esegue un ciclo infinito in cui compie i seguenti passi fondamentali:

1. attesa di richiesta;
2. generazione della risposta;
3. invio della risposta.

La Classe DatagramPacket

Classe con cui vengono rappresentati i pacchetti UDP da inviare e ricevere sulle socket di tipo Datagram.

Si costruisce un datagram packet specificando:

- il contenuto del messaggio (i primi `ilength` bytes dell'array `ibuf`)
- l'indirizzo IP del destinatario
- il numero di porta su cui il destinatario è in ascolto

```
public DatagramPacket(byte ibuf[], int ilength,  
                      InetAddress iaddr, int ippot)
```

Se il pacchetto deve essere ricevuto basta definire il contenuto:

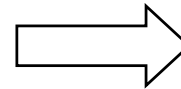
```
public DatagramPacket(byte ibuf[], int ilength)
```

La classe mette a disposizione una serie di metodi per estrarre o settare le informazioni:

```
getAddress(),   setAddress(InetAddress addr)  
getPort(),     setPort(int port)  
getData(),     setData(byte[] buf), etc.
```

La Classe InetAddress

Classe con cui vengono rappresentati gli indirizzi Internet, astruendo dal modo con cui vengono specificati (a numeri o a lettere)



Portabilità e trasparenza

No costruttori, utilizzo di tre metodi statici:

```
public static InetAddress getByName(String  
hostname);
```

restituisce un oggetto `InetAddress` rappresentante l'host specificato (come nome o indirizzo numerico); con il parametro `null` ci si riferisce all'indirizzo di default della macchina locale

```
public static InetAddress[]  
getAllByName(String hostname);
```

restituisce un array di oggetti `InetAddress`; utile in casi di più indirizzi IP registrati con lo stesso nome logico

```
public static InetAddress getLocalHost();
```

restituisce un oggetto `InetAddress` corrispondente alla macchina locale; se tale macchina non è registrata oppure è protetta da un firewall, l'indirizzo è quello di loopback: 127.0.0.1

Tutti possono sollevare l'eccezione `UnknownHostException` se l'indirizzo specificato non può essere risolto (tramite il DNS)

Schema di soluzione: il Client

1. Creazione socket, settaggio eventuali opzioni socket e creazione DatagramPacket:

```
DatagramSocket socket =  
    new DatagramSocket();  
socket.setTimeout(30000);  
byte[] buf = new byte[256];  
DatagramPacket packet = new DatagramPacket(  
    buf, buf.length, addr, port);
```

2. Interazione da console con l'utente:

```
BufferedReader stdIn =  
    new BufferedReader(  
        new InputStreamReader(System.in));  
System.out.print("Nome del file? ");  
String nomeFile = stdIn.readLine();  
System.out.print("Numero linea? ");  
int numLinea = Integer.parseInt(  
    stdIn.readLine());  
richiesta = nomeFile + " " + numLinea;  
String richiesta=nomeFile+" "+numLinea;
```

3. Creazione del pacchetto di richiesta con le informazioni inserite dall'utente:

```
ByteArrayOutputStream boStream =  
    new ByteArrayOutputStream();  
DataOutputStream doStream =  
    new DataOutputStream(boStream);  
doStream.writeUTF(richiesta);  
byte[] data = boStream.toByteArray();
```

4. Riempimento ed invio del pacchetto al server:

```
packet.setData(data);  
socket.send(packetOUT);
```

5. Inizializzazione e attesa del pacchetto di risposta:

```
packet.setData(buf);  
socket.receive(packet);
```

6. Estrazione delle informazioni dal pacchetto ricevuto:

```
ByteArrayInputStream biStream =  
    new ByteArrayInputStream(  
        packet.getData(), 0, packet.getLength());  
DataInputStream diStream =  
    new DataInputStream(biStream);  
  
String risposta = diStream.readUTF();
```

Schema di soluzione: il Server

1. Creazione socket e DatagramPacket:

```
DatagramSocket socket =  
    new DatagramSocket(PORT);  
byte[] buf = new byte[256];  
DatagramPacket packet =  
    new DatagramPacket(buf, buf.length);
```

2. Inizializzazione e attesa del pacchetto di richiesta:

```
packet.setData(buf);  
socket.receive(packet);
```

3. Estrazione delle informazioni dal pacchetto ricevuto:

```
ByteArrayInputStream biStream =  
    new ByteArrayInputStream(  
        packet.getData(), 0, packet.getLength());  
DataInputStream diStream =  
    new DataInputStream(biStream);  
  
richiesta = diStream.readUTF();  
  
StringTokenizer st =  
    new StringTokenizer(richiesta);  
nomeFile = st.nextToken();  
numLinea = Integer.parseInt(st.nextToken());
```

4. Preparazione della risposta con la linea richiesta:

```
String linea =  
    LineUtility.getLine(nomeFile, numLinea);  
  
ByteArrayOutputStream boStream =  
    new ByteArrayOutputStream();  
DataOutputStream doStream =  
    new DataOutputStream(boStream);  
doStream.writeUTF(linea);  
data = boStream.toByteArray();
```

5. Riempimento e invio del pacchetto al client:

```
packet.setData(data, 0, data.length);  
socket.send(packet);
```

LineUtility

```
public class LineUtility {
```

Metodo per recuperare una certa linea di un certo file:

```
    static String getLine
        (String nomeFile, int numLinea) {
        String linea = null;
        BufferedReader in = null;

        try {
            in = new BufferedReader(
                new FileReader(nomeFile));
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
            return linea = "File non trovato";
        }
        try {
            for (int i=1; i<=numLinea; i++){
                linea = in.readLine();
                if ( linea == null){
                    linea = "Linea non trovata";
                    in.close(); return linea;
                }
            }
        }
        catch (IOException e) {
            e.printStackTrace();
            return linea = "Linea non trovata";
        }
        return linea;
    }
```

Metodo per recuperare la linea successiva di un file già aperto in precedenza (servirà fra poco...):

```
    static String getNextLine(BufferedReader in) {

        String linea = null;

        try {
            if ((linea = in.readLine()) == null) {
                in.close();
                linea = "Nessuna linea disponibile";
            }
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
            return linea = "File non trovato";
        }
        catch (IOException e) {
            e.printStackTrace();
            linea = "Nessuna linea disponibile";
        }
        return linea;
    }
}
```

Line Client

```
public class LineClient {

    public static void main(String[] args) {
        InetAddress addr=null;
        int port = -1;
        try{
            if (args.length == 2){
                addr = InetAddress.getByName(args[0]);
                port = Integer.parseInt(args[1]);
            }
            else{ System.out.println("Usage: "+
                +"java LineClient addr port");
                System.exit(1);
            }
        }
        catch(UnknownHostException e){...}
        DatagramSocket socket=null;
        DatagramPacket packet = null;
        byte[] buf = new byte[256];

        try{
            socket = new DatagramSocket();
            socket.setSoTimeout(30000);
            packet = new DatagramPacket(
                buf, buf.length, addr, port);
        }
        catch (SocketException e) {
            e.printStackTrace();
            System.exit(2);
        }
    }
}
```

```
// interazione con l'utente
BufferedReader stdIn = new BufferedReader(
    new InputStreamReader(System.in));
System.out.print("\n^D(Unix)/^Z(Win)+ "+
    "invio per uscire, solo invio per cont.");
try{
    ByteArrayOutputStream boStream = null;
    DataOutputStream doStream = null;
    byte[] data = null;
    String nomeFile = null;
    int numLinea = -1;
    String richiesta=null;
    String risposta = null;
    ByteArrayInputStream biStream = null;
    DataInputStream diStream = null;

    while (stdIn.readLine()!=null){
        try{
            System.out.print("Nome del file? ");
            nomeFile = stdIn.readLine();
            System.out.print("Numero linea? ");
            numLinea = Integer.
                parseInt(stdIn.readLine());
            richiesta = nomeFile+" "+numLinea;
        }
        catch (Exception e) {
            System.out.println("Problemi "+
                "nell'interazione da console: ");
            e.printStackTrace();
            System.out.print("\n^D(Unix)/... ");
            continue;
        }
    }
}
```

```

// preparazione e spedizione richiesta
try{
    boStream = new ByteArrayOutputStream();
    doStream =
        new DataOutputStream(boStream);
    doStream.writeUTF(richiesta);
    data = boStream.toByteArray();
    packet.setData(data);
    socket.send(packet);
}
catch (IOException e){...}
// inizializzazione e ricezione pacchetto
try{
    packet.setData(buf);
    socket.receive(packet);
}
catch (IOException e){ /* come sopra */ }

try{
    biStream = new ByteArrayInputStream(
        packet.getData(), 0, packet.getLength());
    diStream = new DataInputStream(biStream);
    risposta = diStream.readUTF();
}
catch (IOException e){... }
    System.out.print("\n^D(Unix)/^Z(Win...)");
}
}
catch(Exception e){...}

System.out.println("LineClient:termino...");
socket.close();
}
}

```

Line Server

```

public class LineServer {
    private static final int PORT = 4445;

    public static void main(String[] args) {
        DatagramSocket socket = null;
        DatagramPacket packet = null;
        byte[] buf = new byte[256];

        try {
            // Nota: la porta si potrebbe anche
            // passare come argomento
            socket = new DatagramSocket(PORT);
            packet = new DatagramPacket(buf,
                                   buf.length);
        }
        catch (SocketException e) {...}

        try{

            String nomeFile = null;
            int numLinea = -1;
            String richiesta = null;
            ByteArrayInputStream biStream = null;
            DataInputStream diStream = null;
            StringTokenizer st = null;
            ByteArrayOutputStream boStream = null;
            DataOutputStream doStream = null;
            String linea = null;
            byte[] data = null;

```

```

while (true) {
    System.out.println("\nAttendo...");
    try{
        packet.setData(buf);
        socket.receive(packet);
    }
    catch(IOException e){ ... continue; }

    try{
        biStream = new ByteArrayInputStream(
            packet.getData(),0,packet.getLength());
        diStream=new DataInputStream(biStream);
        richiesta = diStream.readUTF();
        st = new StringTokenizer(richiesta);
        nomeFile = st.nextToken();
        numLinea =
            Integer.parseInt(st.nextToken());
    }
    catch(Exception e){/*... come sopra */}

    try{
        String linea =
            LineUtility.getLine(nomeFile, numLinea);

        boStream = new ByteArrayOutputStream();
        doStream =
            new DataOutputStream(boStream);
        doStream.writeUTF(linea);
        data = boStream.toByteArray();
        packet.setData(data);
        socket.send(packet);
    }
    catch(IOException e){/* come sopra */}

```

```

    }
}
catch(Exception e){...}

System.out.println("LineServer: termino..");
socket.close();
}

}

```

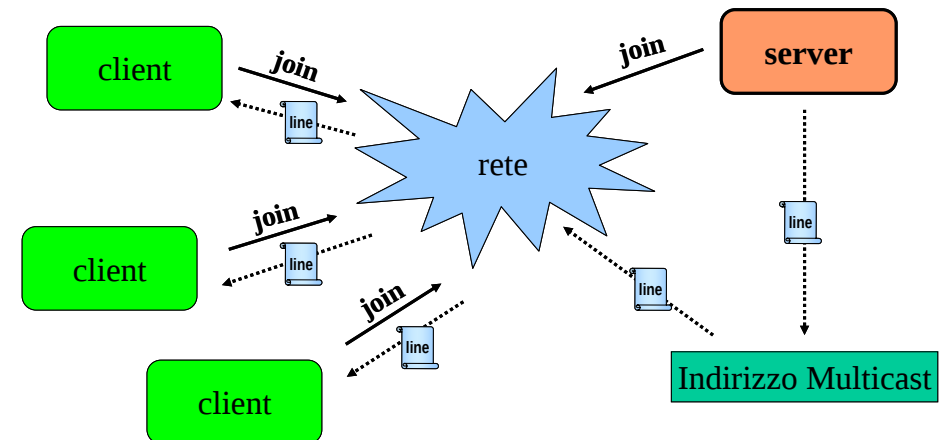
Socket multicast

Modificare il programma sviluppato nella prima parte dell'esercitazione in modo che:

- i client non inviano più nessun pacchetto di richiesta al server ma si associano al gruppo cui il server invia periodicamente le linee di un certo file, ne ricevono alcune (per es. 20), le stampano, poi si dissociano dal gruppo; il programma client è lanciato con due argomenti: l'indirizzo multicast e la porta sulle quali il server trasmette.
- il server invia periodicamente (per es. ogni 2 secondi), una linea di un certo file ad un prefissato indirizzo multicast (es. 230.0.0.1). Terminata la lettura, il server riprende dall'inizio la lettura del file e continua ad eseguire per tutta la propria esistenza questo ciclo infinito di letture e spedizioni.

Il server deve inoltre poter essere invocato da riga di comando con uno o due argomenti e cioè con la sola porta (in questo caso si utilizza l'indirizzo multicast di default, es. 230.0.0.1), oppure l'indirizzo multicast e la porta, presso i quali effettuare gli invii.

Client e Server Multicast



Schema di soluzione: il Client Multicast

1. Creazione socket, settaggio eventuali opzioni socket e creazione DatagramPacket:

```
// Aggancio alla porta passata come
// argomento
MulticastSocket socket =
    new MulticastSocket(port);
socket.setTimeout(20000);
byte[] buf = new byte[256];
DatagramPacket packet =
    new DatagramPacket(buf, buf.length);
```

2. Adesione al gruppo multicast passato come argomento:

```
address =
    InetAddress.getByName(args[0]);
socket.joinGroup(address);
```

3. Inizializzazione e ricezione del pacchetto di risposta:

```
packet.setData(buf);
socket.receive(packet);
```

4. Estrazione delle informazioni dal pacchetto ricevuto:

```
biStream = new ByteArrayInputStream(
    packet.getData(), 0, packet.getLength());
diStream = new DataInputStream(biStream);
risposta = diStream.readUTF();
```

5. Uscita dal gruppo multicast

```
socket.leaveGroup(address);
```

Schema di soluzione: il Server Multicast

1. Creazione socket e DatagramPacket:

```
// Aggancio alla porta passata come
// argomento
socket = new MulticastSocket(port);
byte[] data = new byte[256];
DatagramPacket packet =
    new DatagramPacket(data, data.length,
        group, port);
```

2. Adesione al gruppo multicast:

```
address =
    InetAddress.getByName(stringAddr);
socket.joinGroup(address);
```

3. Preparazione della risposta da inviare:

```
String linea = LineUtility.getNextLine(in);

boStream = new ByteArrayOutputStream();
doStream = new DataOutputStream(boStream);
doStream.writeUTF(linea);
data = boStream.toByteArray();
```

4. Riempimento e invio del pacchetto a tutto il gruppo:

```
packet.setData(data);
socket.send(packet);
```

Line Multicast Client

```
public class MulticastClient {

    public static void main(String[] args) {
        InetAddress group = null;
        int port = -1;
        try {
            if (args.length == 2) {
                group = InetAddress.getByName(args[0]);
                port = Integer.parseInt(args[1]);
            } else {
                System.out.println("Usage: java"+
                    "MulticastClient MCastAddr MCastPort");
                System.exit(1);
            }
        }
        catch (Exception e){... System.exit(2);}

        // creazione della socket multicast
        // e creazione datagram packet
        MulticastSocket socket = null;
        byte[] buf = new byte[256];
        DatagramPacket packet = null;

        try {
            socket = new MulticastSocket(port);
            socket.setSoTimeout(20000); // 20 secondi
            packet=new DatagramPacket(buf,buf.length);
            System.out.println("Creata la socket: " +
                socket);
        } catch (IOException e){... System.exit(3);}
    }
}
```

```
// adesione al gruppo associato
try {
    socket.joinGroup(group);
    System.out.println("Adesione al gruppo " +
        addr);
} catch (IOException e){... System.exit(4);}

ByteArrayInputStream biStream = null;
DataInputStream diStream = null;
String linea = null;

for (int i = 0; i < 20; i++) {
    try {
        packet.setData(buf);
        socket.receive(packet);
    } catch (SocketTimeoutException te) {
        System.out.println("Non ho ricevuto "+
            "niente per 20 secondi, chiudo!");
        System.exit(4);
    } catch (IOException e) {... continue;}
}
```

```

try {
    biStream = new ByteArrayInputStream(
        packet.getData(), 0, packet.getLength());
    diStream = new DataInputStream(biStream);
    linea = diStream.readUTF();
    System.out.println("Linea ricevuta: " +
        linea);
} catch (IOException e) {... continue;}
}
// uscita dal gruppo e chiusura della socket
System.out.println("\nUscita dal gruppo");
try {
    socket.leaveGroup(addr);
} catch (IOException e) { ... }

System.out.println("MulticastClient: "+
    "termino...");
socket.close();
}
} t

```

Line Multicast Server

```

public class MulticastServer {

    /*
     * File da dove leggo, si deve trovare nella
     * directory dove viene lanciato il server
     */
    private static final String FILE =
        "saggezza.txt";
    private static BufferedReader in = null;
    private static boolean moreLines = true;

    public static void main(String[] args) {
        long WAIT = 1000;
        // Indirizzo del gruppo multicast e porta
        InetAddress group = null;
        int port = -1;
        try {
            if (args.length == 1) {
                group = InetAddress.
                    getByName("230.0.0.1");
                port = Integer.parseInt(args[0]);
            } else if (args.length == 2) {
                group = InetAddress.getByName(args[0]);
                port = Integer.parseInt(args[1]);
            } else {
                System.out.println("Usage: ...");
                System.exit(1);
            }
        }
        catch (Exception e) {... System.exit(1);}
    }
}

```

```

// Creazione datagram packet e socket
MulticastSocket socket = null;
byte[] data = new byte[256];
DatagramPacket packet = new DatagramPacket(
    data, data.length, group, port);
try {
    socket = new MulticastSocket(port);
    socket.joinGroup(group);
    System.out.println("Socket: " + socket);
} catch (Exception e) { ... }
int count = -1; // contatore per debug
ByteArrayOutputStream boStream = null;
DataOutputStream doStream = null;

while (true) {
    count = 0; // azzero count
    moreLines = true;
    try {
        // costruisco un nuovo buffered reader
        in = new BufferedReader
            (new FileReader(FILE));
    } catch (Exception e) {... continue; }
    while (moreLines) {
        count++;
        byte[] buf = new byte[256];
        // estrazione della linea
        String linea =
            LineUtility.getNextLine(in);
        if (linea.equals("Nessuna linea "+
            disponibile")) {
            moreLines = false;
            break; //esco dal ciclo
        }
    }
}

```

```

try {
    boStream = new ByteArrayOutputStream();
    doStream =
        new DataOutputStream(boStream);
    doStream.writeUTF(linea);
    data = boStream.toByteArray();
    packet.setData(data);
    socket.send(packet);
} catch (Exception e) {... continue; }

// attesa tra un invio e l'altro...
try {
    Thread.sleep( 2 * WAIT );
} catch (Exception e) {... continue; }
}
}
}

```

Problemi utilizzo MulticastSocket con computer non connessi alla rete

Sono stati riscontrati alcuni problemi con l'utilizzo delle MulticastSocket con computer non connessi in rete.

Per superare tali problemi bisogna impostare l'interfaccia di rete **PRIMA** di fare la join al gruppo.

Bisogna quindi sostituire questo codice:

```
socket.joinGroup(group);
```

Con il seguente:

```
// mi procuro l'inetaddress locale
InetAddress netInterface =
    InetAddress.getByName("localhost");
// imposto l'interfaccia di rete
socket.setInterface(netInterface);
socket.joinGroup(group);
```

Per risolvere questo ed altri problemi realizzativi che si possono presentare durante il corso si tenga sempre monitorata **la sezione FAQ del sito!!!**