Esercitazioni di base sulle Socket in Java:

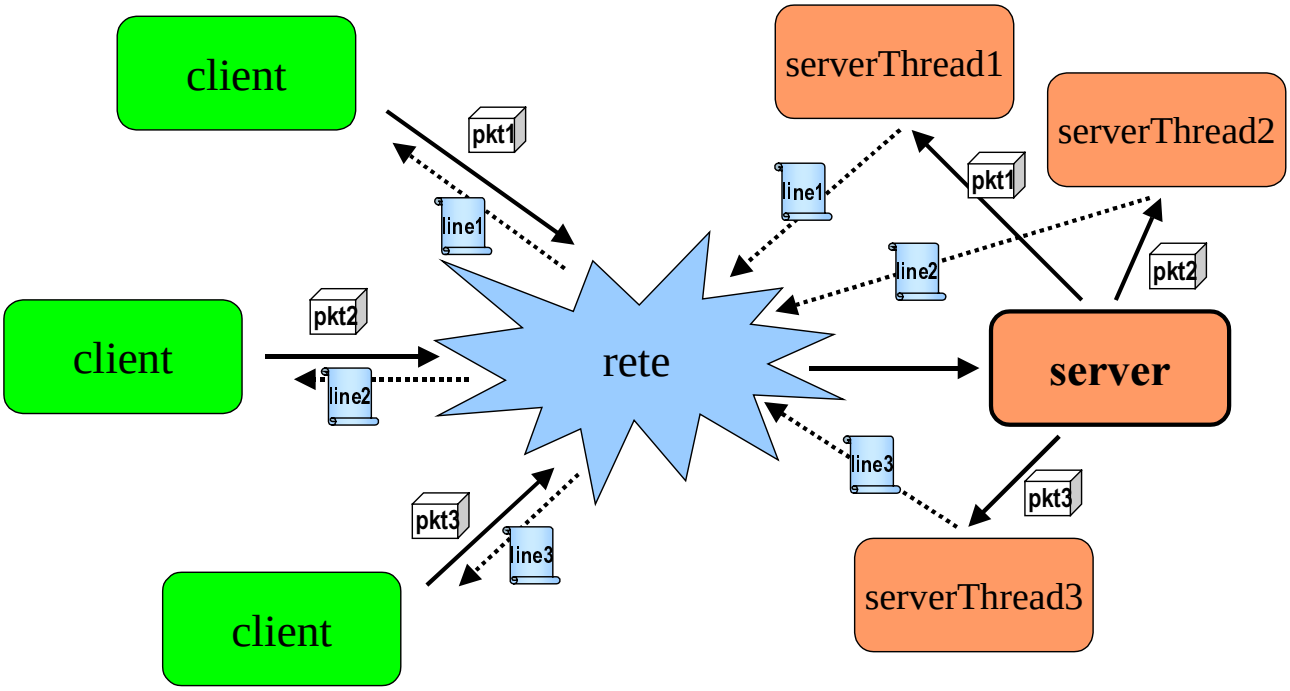
Socket di tipo datagram

a) Line Client e Server

Sviluppare un'applicazione C/S in cui:

- i client inviano al server pacchetti (vuoti) che vengono interpretati dal server come richiesta della linea corrente di un certo file di testo;

- il server invia a ciascun client un pacchetto contenente una linea (le linee vengono estratte in modo sequenziale) del file; se non riesce invia data e ora attuali



a) Line Client e Server

```
//  LineServer.java


import java.io.*;
import java.net.*;
import java.util.*;


public class LineServer {

public static final int PORT = 4445; // porta al di fuori del range 1-1024 !
public static final String FILE = "Dante.txt";
DatagramSocket socket;
static BufferedReader in = null;
static boolean moreLines = true;

public LineServer(){

    try {
    // creazione della socket datagram
    socket = new DatagramSocket(PORT);
    System.out.println("Socket: " + socket);
    } catch (SocketException e) {
         System.err.println("Unable to create the socket");
      }

    try {
       // associazione di uno stream di input al file da cui estrarre le linee
      in = new BufferedReader(new FileReader(FILE));
      System.out.println("File "+FILE+" opened");

    } catch (FileNotFoundException e) {
         System.err.println("Could not open line file, serving time instead.");
      }
}

public static void main(String[] args) throws IOException {

   System.out.println("LineServer: started");
   LineServer server=new LineServer();

    while (moreLines) {
       try {
          byte[] buf = new byte[256];

          // ricezione della richiesta
          DatagramPacket packet = new DatagramPacket(buf, buf.length);
          server.socket.receive(packet);

          // preparazione della linea da inviare
          String dString = null;
          if (in == null) dString = new Date().toString();
          else dString = getNextLine();
          if (dString!=null) {
           buf = dString.getBytes();

            // "impacchettamento" e invio della risposta
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            packet = new DatagramPacket(buf, buf.length, address, port);
            server.socket.send(packet);
            System.out.println("Sending packet to "+address+", "+port);
          }

        } catch (IOException e) {
             e.printStackTrace();
             moreLines = false;
          }

    }
```

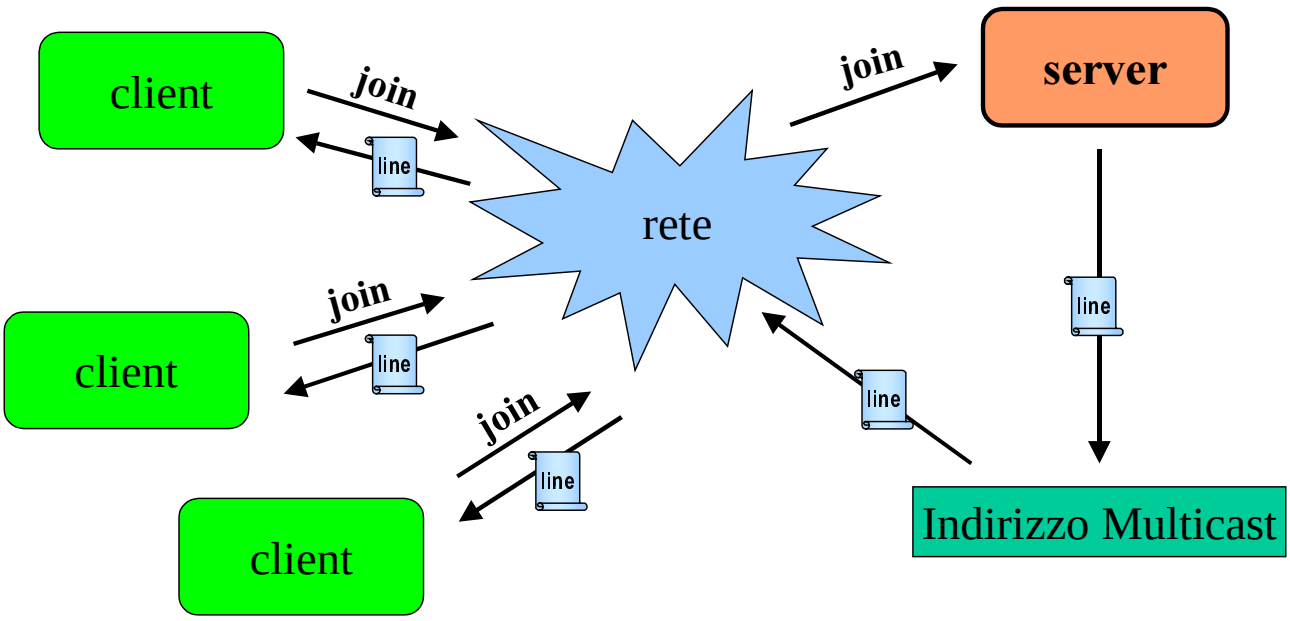
```
      System.out.println("No more lines. Goodbye.");
      System.out.println("LineServer: closing...");
      server.socket.close();
  }

    static String getNextLine() {
        String returnValue = null;
        try {
            if ((returnValue = in.readLine()) == null) {
                in.close();
                moreLines = false;
                returnValue = null;
                //returnValue = "No more lines. Goodbye.";
            }
        } catch (IOException e) {
            returnValue = "IOException occurred.";
        }
        return returnValue;
    }

}



//  LineClient.java


import java.io.*;
import java.net.*;
import java.util.*;


public class LineClient {

    public static void main(String[] args) throws IOException {

        // creazione della socket datagram con un timeout di 30s
        DatagramSocket socket = new DatagramSocket();
        socket.setSoTimeout(30000);
        System.out.println("LineClient: started");
         System.out.println("Socket: "+ socket);


        // creazione e invio del pacchetto di richiesta
        byte[] buf = new byte[256];
        InetAddress addr;
       if (args.length == 0) addr = InetAddress.getByName(null);
       else addr = InetAddress.getByName(args[0]);

         //for (int i=0; i<200; i++){

         for (int i=0; i<5; i++){
          DatagramPacket packet = new DatagramPacket(buf, buf.length, addr,
                                                                    LineServer.PORT);
          socket.send(packet);
          System.out.println("Request sent to "+ addr);

          // pulizia del buffer
          for (int j=0; j<256; j++) buf[j]=' ';

          // ricezione e stampa della risposta
          packet = new DatagramPacket(buf, buf.length);
          socket.receive(packet);
          // sospensiva solo per i millisecondi indicati, dopodichè solleva una
                                                                  SocketException
          String received = new String(packet.getData());
          System.out.println("Line got: " + received);
         }

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

b) Line Client e Server 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, ne ricevono alcune (per es. 5), le stampano, poi si dissociano dal gruppo;

- il server invia periodicamente (per es. ogni 5 secondi), una linea a tutti i client che si sono registrati all'indirizzo multicast



b) Line Client e Server multicast

```
// MulticastServer.java


import java.io.*;
import java.net.*;
import java.util.*;


public class MulticastServer extends LineServer{


    public MulticastServer(){
     super();
    }

    public static void main(String[] args) throws IOException {

      long WAIT = 1000;
      int count=0;

      System.out.println("MulticastServer: started");
      MulticastServer server=new MulticastServer();
      // creazione del gruppo associato all'indirizzo multicast
      InetAddress group = InetAddress.getByName("230.0.0.1");
      System.out.println("Creating group "+ group);

      while (moreLines) {
            try {
             count++;
                byte[] buf = new byte[256];

                // costruzione della linea
                String dString = null;
                if (in == null) dString = new Date().toString();
                else dString = getNextLine();
                if (dString!=null) {
                  buf = dString.getBytes();

                  // "impacchettamento" e invio della linea al gruppo
                  DatagramPacket packet = new DatagramPacket(buf, buf.length, group,
                                                                          PORT);
                  server.socket.send(packet);
                  System.out.println("Sending line # "+ count);
                }

                  // attesa tra un invio e l'altro...
                try {
                    Thread.sleep((long)(Math.random() * WAIT));
                } catch (InterruptedException e) { }
             } catch (IOException e) {
                e.printStackTrace();
                moreLines = false;
            }
       }
     System.out.println("No more lines. Goodbye.");
     System.out.println("MulticastServer: closing...");
     server.socket.close();
    }
}
```

```java
// MulticastClient.java


import java.io.*;
import java.net.*;
import java.util.*;


public class MulticastClient {


    public static void main(String[] args) throws IOException {

         // creazione della socket multicast
        MulticastSocket socket = new MulticastSocket(MulticastServer.PORT);
        socket.setSoTimeout(20000);
        InetAddress address = InetAddress.getByName("230.0.0.1");
        System.out.println("MulticastClient: started");
         System.out.println("Socket: "+ socket);


        // adesione al gruppo associato all'indirizzo multicast
        socket.joinGroup(address);
        System.out.println("Joining to "+ address);


        DatagramPacket packet;

        // ricezione di alcune linee
        for (int i = 0; i < 5; i++) {

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

            String received = new String(packet.getData());
            System.out.println("Line got: " + received);
        }

         // uscita dal gruppo e chiusura della socket
        System.out.println("Leaving group...");
        socket.leaveGroup(address);
        System.out.println("MulticastClient: closing...");
        socket.close();
    }

}
```

NOTA:

Il server multicast, ereditando dal server non multicast, usa una DatagramSocket per fare il broadcasting del pacchetto anziché usare una MulticastSocket. L'importante è che siano multicast l'indirizzo del pacchetto e la socket usata dal client per riceverlo.

Prove:

1. aprire 2 o più console sulla stessa macchina:
   - java LineServer
   - java LineClient
   - java LineClient
   - ecc.

   Modificare LineClient (200 cicli di richiesta)
   - ⇨ terminazione linee del file e scatto del timeout

2. aprire 2 o più console su macchine diverse:
   - java LineServer
   - java LineClient indMacchinaServer
   - java LineClient indMacchinaServer
   - ecc.

3. aprire 2 o più console sulla stessa macchina:
   - java MulticastServer
   - java MulticastClient
   - java MulticastClient
   - ecc.

4. aprire 2 o più console su macchine diverse:
   - java MulticastServer
   - java MulticastClient
   - java MulticastClient
   - ecc.

…

Ulteriori proposte:

I pacchetti inviati dai client potrebbero non essere vuoti ma contenere informazioni, sotto forma di stringa, che il server deve estrarre per svolgere il servizio, per esempio:

- numero di righe da estrarre dal file
- nome del file da cui estrarre la riga, supponendo di averne a disposizione più d'uno; in questo caso l'apertura del file e l'associazione allo stesso dello stream di lettura deve essere fatto soltanto dopo la ricezione della richiesta dal client
- indirizzo del destinatario a cui deve essere inviata la linea (o le linee); diversamente dal caso delle socket stream, non è necessario che il destinatario sia connesso o in grado di accettare una connessione, bastano indirizzo e porta, che anziché essere estratti dai rispettivi campi del pacchetto dovranno essere ricavati dal messaggio, scritto secondo determinate convenzioni (es. "host porta numLinee", ogni informazione separata da uno spazio)
- …