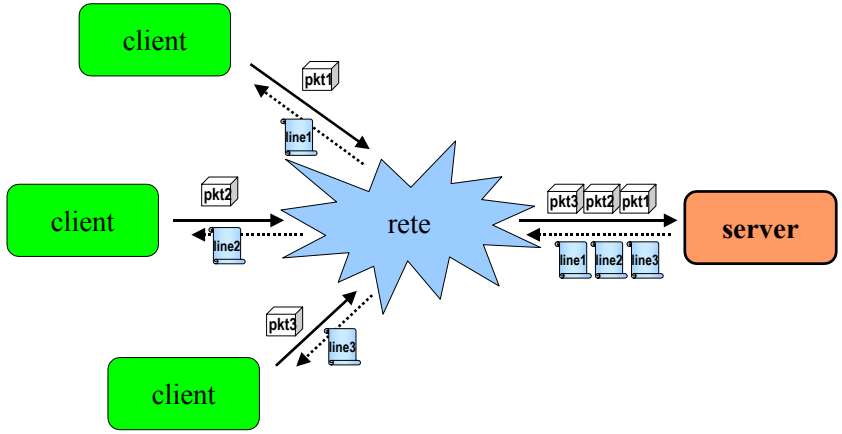
Esercitazione:
Socket Java senza connessione

Sviluppare un'applicazione C/S in cui:

- il client invia al server pacchetti contenenti **il nome del file** e il **numero della linea** del file che vuole ricevere, che sono richiesti dall'utente usando l'input da console. Si 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**;

- 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.
  Dopo 3 minuti senza ricezione di richieste il server termina.

Client e Server Datagram



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 iport)
```

Se il pacchetto deve essere ricevuto basta definire il contenuto:

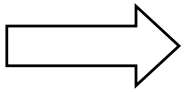
```
public DatagramPacket(byte ibuf[], int ilength)
```

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, astraendo 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 ed eventuale settaggio opzioni:

```
socket = new DatagramSocket();
socket.setxxx(…);
```

2. Interazione da console con l'utente:

```
BufferedReader stdIn =
   new BufferedReader(
       new InputStreamReader(System.in));
  System.out.print("Domanda… ");
  String richiesta = stdIn.readLine();
```

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

```
packetOUT = DatagramUtility.buildPacket
       (addr, LineServer.PORT, richiesta);
```

4. Invio del pacchetto al server:

```
socket.send(packetOUT);
```

5. Attesa del pacchetto di risposta:

```
packetIN = new DatagramPacket
                    (buf, buf.length);
socket.receive(packetIN);
```

6. Estrazione delle informazioni dal pacchetto ricevuto:

```
risposta =
   DatagramUtility.getContent(packetIN);
```


Schema di soluzione: il Server

1. Creazione socket:

```
socket = new DatagramSocket(PORT);
```

2. Attesa del pacchetto di richiesta:

```
packet = new DatagramPacket
                    (buf, buf.length);
socket.receive(packet);
```

3. Estrazione delle informazioni dal pacchetto ricevuto:

```
String richiesta =
       DatagramUtility.getContent(packet);
StringTokenizer st =
       new StringTokenizer(richiesta);
nomeFile = st.nextToken();
numLinea = Integer.parseInt(st.nextToken());
```

4. Creazione del pacchetto di risposta con la linea richiesta:

```
String linea =
   LineUtility.getLine(nomeFile, numLinea);
packet =
   DatagramUtility.buildPacket(
            mittAddr, mittPort, linea);
```

5. Invio del pacchetto al client:

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

Datagram Utility

```
public class DatagramUtility {
```

Metodo per creare un pacchetto da una stringa:

```
 static protected DatagramPacket buildPacket
   (InetAddress addr, int port, String msg)
                          throws IOException{

  ByteArrayOutputStream boStream =
                new ByteArrayOutputStream();
  DataOutputStream doStream =
              new DataOutputStream(boStream);

  doStream.writeUTF(msg);

  byte[] data = boStream.toByteArray();

  return new DatagramPacket(data,
                  data.length, addr, port);
  }
```

Metodo per recuperare una stringa da un pacchetto:

```
 static protected String getContent
     (DatagramPacket dp) throws IOException{

  ByteArrayInputStream biStream =
    new ByteArrayInputStream(dp.getData(),
                          0, dp.getLength());
  DataInputStream diStream =
              new DataInputStream(biStream);
  String msg = diStream.readUTF();
  return msg;
 }
}
```


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++)
              in.readLine();
        if ((linea = in.readLine()) == null)
            linea = "Linea non trovata";
        }
      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:

```
 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) {

     DatagramSocket socket=null;
     try{
          socket = new DatagramSocket();
          socket.setSoTimeout(30000);
     }
     catch (SocketException e) {
       e.printStackTrace();
       System.exit(1);
     }


     InetAddress addr=null;
     try{
      if (args.length == 0)
          addr = InetAddress.getByName(null);
      else
       addr = InetAddress.getByName(args[0]);
      }
      catch(UnknownHostException e){
            e.printStackTrace();
            System.exit(2);
         }


     BufferedReader stdIn =
      new BufferedReader(
           new InputStreamReader(System.in));
     String richiesta=null;
```

```
System.out.print("\n^D(Unix)/^Z(Win)+invio
per uscire, solo invio per continuare: ");

 try{

   while (stdIn.readLine()!=null){

     try{
       System.out.print("Nome del file? ");
       String nomeFile = stdIn.readLine();
       System.out.print("Numero linea? ");
       int numLinea =
        Integer.parseInt(stdIn.readLine());
        richiesta = nomeFile+" "+numLinea;
     }
     catch (Exception e) {
       e.printStackTrace();
       System.out.print("\n^D(Unix)/^Z(Win)
            +invio per uscire,
              solo invio per continuare: ");
       continue;
      }



     try{
      DatagramPacket packetOUT =
        DatagramUtility.buildPacket
        (addr, LineServer.PORT, richiesta);
      socket.send(packetOUT);
     }
     catch (IOException e){/*…come sopra */}
```



```
   DatagramPacket packetIN=null;

   try{
     byte[] buf = new byte[256];
      packetIN =
        new DatagramPacket(buf, buf.length);
        socket.receive(packetIN);
   }
    catch (IOException e){/*…come sopra */}
    try{
      String risposta=null;
      risposta =
        DatagramUtility.getContent(packetIN);
      System.out.println("Risposta: " +
                                    risposta);
     }
     catch (IOException e){/*…come sopra */}

System.out.print("\n^D(Unix)/^Z(Win)+invio
per uscire, solo invio per continuare: ");
       }// while
 }
 catch(Exception e){
    e.printStackTrace();
  }
System.out.println("LineClient: termino..");
        socket.close();}
}
```

Line Server

```
public class LineServer {

 public static final int PORT = 4445;

public static void main(String[] args) {

 DatagramSocket socket = null;

 try {
    socket = new DatagramSocket(PORT);
 }
 catch (SocketException e) {
    e.printStackTrace();
    System.exit(1);
 }

 try{

   while (true) {
      System.out.println("\n
            In attesa di richieste...");

    DatagramPacket packet=null;
    InetAddress mittAddr=null;
    int mittPort=0;

    try{
      byte[] buf = new byte[256];
      packet =
        new DatagramPacket(buf, buf.length);
      socket.receive(packet);
      mittAddr = packet.getAddress();
      mittPort = packet.getPort();
    }
```



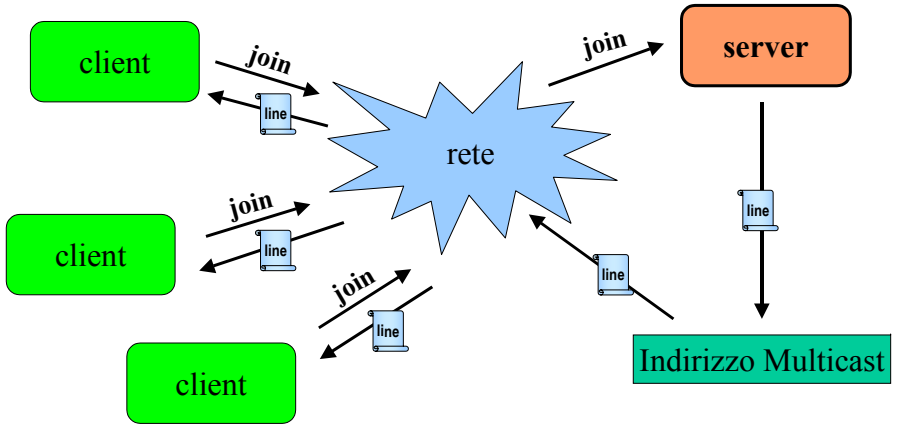
```
    catch(IOException e){
       e.getMessage());
       e.printStackTrace();
       continue;
    }

  String nomeFile;
  int numLinea;

  try{
    String richiesta =
         DatagramUtility.getContent(packet);
   StringTokenizer st =
            new StringTokenizer(richiesta);
   nomeFile = st.nextToken();
   numLinea =
         Integer.parseInt(st.nextToken());
  }
  catch(Exception e){/*… come sopra */}

 try{
   String linea =
      LineUtility.getLine(nomeFile, numLinea);
   packet =
          DatagramUtility.buildPacket(
                mittAddr, mittPort, linea);
        socket.send(packet);
  }
  catch(IOException e){/*… come sopra */}

 } // while
}
catch(Exception e){
   e.printStackTrace();
}
System.out.println("LineServer: termino..");
socket.close();
  }
}
```

Socket multicast

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

- i client non inviino più nessun pacchetto di richiesta al server ma si **associno al gruppo** cui il server invia periodicamente le linee di un certo file, ne ricevano alcune (per es. 5), le stampino, poi si dissocino dal gruppo

- il server **invii periodicamente** (per es. ogni 5 secondi), una linea di un certo file ad un **prefissato indirizzo multicast** (230.0.0.1). Terminato il file, il server interrompa gli invii notificando l'interruzione del servizio mediante l'invio di un messaggio di congedo



Schema di soluzione: il Client Multicast

1. Creazione socket ed eventuale settaggio opzioni:

```
socket = new
  MulticastSocket(MulticastServer.PORT);
socket.setxxx(…);
```

2. Adesione al gruppo multicast:

```
address =
       InetAddress.getByName("230.0.0.1");
socket.joinGroup(address);
```

3. Attesa del pacchetto di risposta:

```
packet = new DatagramPacket
                        (buf, buf.length);
socket.receive(packet);
```

4. Estrazione delle informazioni dal pacchetto ricevuto:

```
risposta =
   DatagramUtility.getContent(packet);
```

5. Uscita dal gruppo multicast

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

Schema di soluzione: il Server Multicast

1. Creazione socket:

```
socket = new DatagramSocket(PORT);
```

2. Creazione del gruppo:

```
group = InetAddress.getByName("230.0.0.1");
```

3. Creazione del pacchetto da inviare:

```
String linea = LineUtility.getNextLine(in);
DatagramPacket packet =
             DatagramUtility.buildPacket
                        (group, PORT, linea);
```

4. Invio del pacchetto a tutto il gruppo:

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


Line Multicast Client

```
public class MulticastClient {

 public static void main(String[] args) {


  MulticastSocket socket=null;
  try{
   socket = new
     MulticastSocket(MulticastServer.PORT);
   socket.setSoTimeout(20000);
  }
  catch (IOException e) {
    e.printStackTrace();
   System.exit(1);
  }



 InetAddress address=null;
   try{
     address =
        InetAddress.getByName("230.0.0.1");
     socket.joinGroup(address);
   }
   catch (IOException e) {/*… come sopra */}


   DatagramPacket packet;
   for (int i = 0; i < 5; i++) {

    System.out.println("\n
         In attesa di un datagramma... ");
     byte[] buf = new byte[256];
    packet =
       new DatagramPacket(buf, buf.length);
     try{ socket.receive(packet); }
```

```
     catch (IOException e) {
         e.printStackTrace();
         continue;
      }

     try{
      String linea=null;
      linea =
        DatagramUtility.getContent(packet);
        System.out.println(
              "Linea ricevuta: " + linea);
      }
      catch (IOException e) {/*…come sopra*/
       }
} //for

System.out.println("\nUscita dal gruppo");
   try{
     socket.leaveGroup(address);
   }
   catch (IOException e) {
     e.printStackTrace();
   }
  System.out.println("MulticastClient:
                                  termino");
   socket.close();
    }
}
```


Line Multicast Server

```
public class MulticastServer {

  public static final int PORT = 4446;

     public static final String
                    FILE = "saggezza.txt";
     static BufferedReader in = null;
     static boolean moreLines = true;


public static void main(String[] args) {

   long WAIT = 1000;
   int count=0;
   DatagramSocket socket = null;

   try {
      socket = new DatagramSocket(PORT);
      System.out.println("Socket: " +
                                     socket);
    }
   catch (SocketException e) {
      e.printStackTrace();
      System.exit(1);
   }

// associazione di uno stream di input al
file da cui estrarre le linee
   try {
     in = new BufferedReader(
                new FileReader(FILE));
      System.out.println("File "+FILE+
                                  " aperto");
   }
   catch (FileNotFoundException e)
         {/*… come sopra */}
```

```
// creazione del gruppo
InetAddress group=null;
    try{
       group =
          InetAddress.getByName("230.0.0.1");
    }
     catch (UnknownHostException e)
         {/*… come sopra */}

while (moreLines) {
  count++;
  byte[] buf = new byte[256];
  String linea =
            LineUtility.getNextLine(in);
  try {
     DatagramPacket packet =
            DatagramUtility.buildPacket(
                        group, PORT, linea);
      socket.send(packet); }
   catch (Exception e) {
       e.printStackTrace();
       continue; }

 try {
  Thread.sleep((long)(Math.random()*WAIT));
 } catch (InterruptedException e) { }
} // while

 System.out.println("File terminato");
 System.out.println("MulticastServer: termino…");

socket.close(); }
}
```

NOTA: Il server 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.**