Esercitazioni di base sulle Socket in Java:

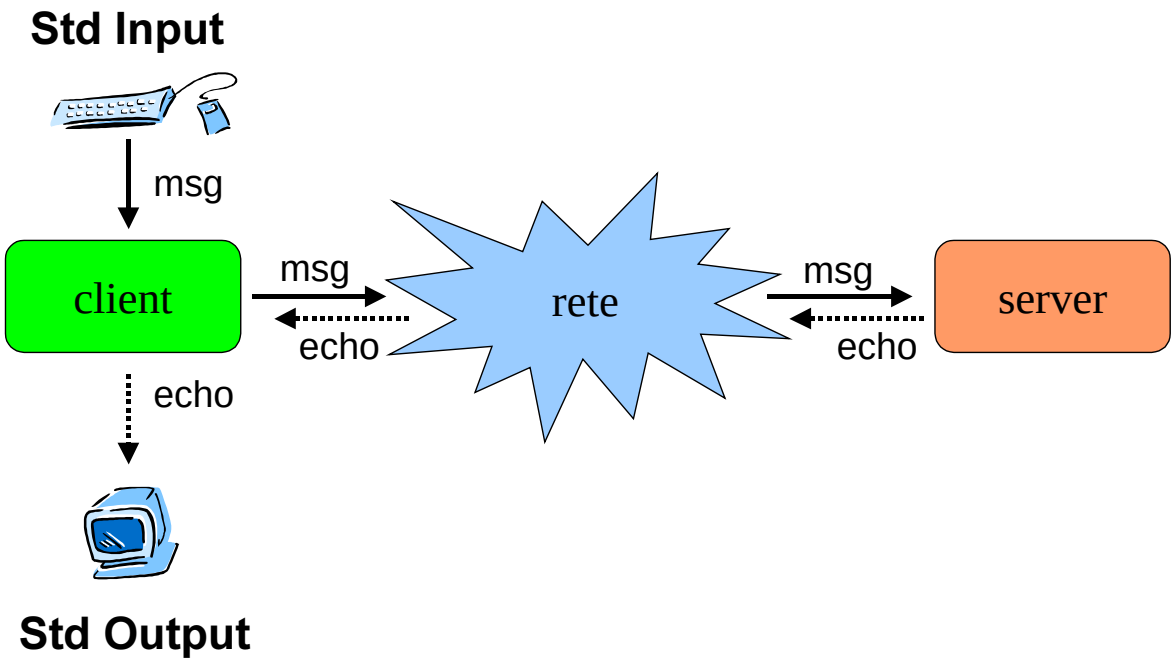
Socket di tipo stream

a) Echo Client e Server

Sviluppare un'applicazione C/S in cui:

- il server attende una connessione da parte del client (su **java.net.ServerSocket**), usa la socket (**java.net.Socket**) prodotta dalla connessione per creare uno stream di input e uno stream di output (con gli opportuni filtraggi…), e ripete (echo) sul secondo tutto ciò che legge dal primo finché non riceve la linea che contiene solo la stringa "END", dopodiché chiude la connessione;

- il client si connette al server (con **java.net.Socket**), crea uno stream di output, attraverso cui inviare linee di testo, ricevute da tastiera (**java.lang.System.in**), al server, e uno stream di input da cui ricevere ciò che il server invia.



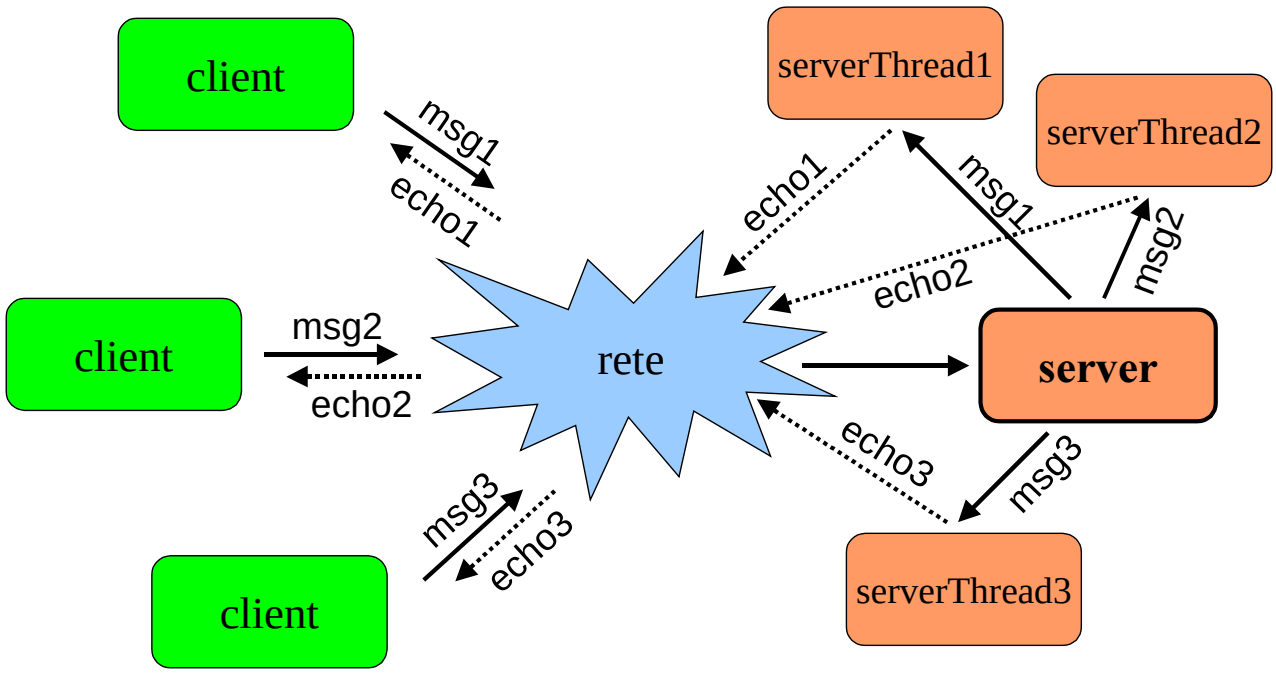
a) Echo Client e Server

```java
// EchoServer.java


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


public class EchoServer {

public static final int PORT = 1050; // porta al di fuori del range 1-1024 !

public static void main(String[] args) throws IOException {
  ServerSocket serverSocket = new ServerSocket(PORT);
  System.out.println("EchoServer: started ");
  System.out.println("Server Socket: " + serverSocket);
  Socket clientSocket=null;
  BufferedReader in=null;
  PrintWriter out=null;
  try {
    // bloccante finchè non avviene una connessione
    clientSocket = serverSocket.accept();
    System.out.println("Connection accepted: "+ clientSocket);

    // creazione stream di input da clientSocket
    InputStreamReader isr = new InputStreamReader(clientSocket.getInputStream());
    in = new BufferedReader(isr);

    // creazione stream di output su clientSocket
    OutputStreamWriter osw = new OutputStreamWriter(clientSocket.getOutputStream());
     BufferedWriter bw = new BufferedWriter(osw);
    out = new PrintWriter(bw, true);

    //ciclo di ricezione dal client e invio di risposta
    while (true) {
      String str = in.readLine();
      if (str.equals("END")) break;
      System.out.println("Echoing: " + str);
      out.println(str);
    }
  }
  catch (IOException e) {
    System.err.println("Accept failed");
    System.exit(1);
  }
  // chiusura di stream e socket
  System.out.println("EchoServer: closing...");
  out.close();
  in.close();
  clientSocket.close();
  serverSocket.close();
 }
} // EchoServer
```

```java
// EchoClient.java

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


public class EchoClient {

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

  /* Lanciando il programma senza argomenti si ottiene il local loopback IP address,
  per testarlo in locale (client e server sulla stessa macchina), altrimenti
  si possono passare da linea di comando l'indirizzo o il nome della macchina remota */

  InetAddress addr;
  if (args.length == 0) addr = InetAddress.getByName(null);
  else addr = InetAddress.getByName(args[0]);

  Socket socket=null;
  BufferedReader in=null, stdIn=null;
  PrintWriter out=null;


  try {
    // creazione socket
    socket = new Socket(addr, EchoServer.PORT);
    System.out.println("EchoClient: started");
     System.out.println("Client Socket: "+ socket);

    // creazione stream di input da socket
    InputStreamReader isr = new InputStreamReader( socket.getInputStream());
    in = new BufferedReader(isr);

    // creazione stream di output su socket
    OutputStreamWriter osw = new OutputStreamWriter( socket.getOutputStream());
    BufferedWriter bw = new BufferedWriter(osw);
    out = new PrintWriter(bw, true);

    // creazione stream di input da tastiera
    stdIn = new BufferedReader(new InputStreamReader(System.in));
    String userInput;

    // ciclo di lettura da tastiera, invio al server e stampa risposta
    while (true){
       userInput = stdIn.readLine();
      out.println(userInput);
       if (userInput.equals("END")) break;
      System.out.println("Echo: " + in.readLine());
     }
  }
  catch (UnknownHostException e) {
    System.err.println("Don't know about host "+ addr);
    System.exit(1);
  } catch (IOException e) {
      System.err.println("Couldn't get I/O for the connection to: " + addr);
      System.exit(1);
    }

  System.out.println("EchoClient: closing...");
  out.close();
  in.close();
  stdIn.close();
  socket.close();
  }
} //EchoClient
```

b) Echo Client e Server versione multithreaded

Modificare il programma sviluppato nella prima parte dell'esercitazione in modo da rendere il server in grado di gestire più richieste (per esempio quelle di 10 client che inviano ciascuno 10 messaggi) in maniera concorrente; per semplicità eliminare l'interazione con la console utilizzando client che inviano messaggi "automatici".



b) Echo Client e Server versione multithreaded

```
// EchoMultiServer.java

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

class ServerThread extends Thread {
  private static int counter = 0;
  private int id = ++counter;

  private Socket socket;
  private BufferedReader in;
  private PrintWriter out;

  public ServerThread(Socket s) throws IOException {
    socket = s;
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    OutputStreamWriter osw = new OutputStreamWriter(socket.getOutputStream());
    out = new PrintWriter(new BufferedWriter(osw), true);
    start();
    System.out.println("ServerThread "+id+": started");

  }

public void run() {
  try {
    while (true) {
    String str = in.readLine();
    if (str.equals("END")) break;
    System.out.println("ServerThread "+id+": echoing -> " + str);
    out.println(str);
    }
  System.out.println("ServerThread "+id+": closing...");
  } catch (IOException e) {}
  try {
    socket.close();
  } catch(IOException e) {}
 }
} // ServerThread


public class EchoMultiServer {
  static final int PORT = 1050;


  public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket(PORT);
    System.out.println("EchoMultiServer: started");
    System.out.println("Server Socket: " + serverSocket);

    try {
      while(true) {
        // bloccante finchè non avviene una connessione:
        Socket clientSocket = serverSocket.accept();
        System.out.println("Connection accepted: "+ clientSocket);

        try {
          new ServerThread(clientSocket);
        } catch(IOException e) {
            clientSocket.close();
          }
      }
    }
    catch (IOException e) {
      System.err.println("Accept failed");
      System.exit(1);
    }
    System.out.println("EchoMultiServer: closing...");
    serverSocket.close();
  }

} // EchoMultiServer
```

```java
// EchoMultiClient.java
// Client che testa l'EchoMultiServer lanciando più clienti.

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

class ClientThread extends Thread {
  private Socket socket;
  private BufferedReader in;
  private PrintWriter out;
  private static int counter = 0;
  private int id = counter++;
  private static int threadcount = 0;

  public static int threadCount() {
    return threadcount;
  }

  public ClientThread(InetAddress addr) {
    threadcount++;

    try {
      socket = new Socket(addr, EchoMultiServer.PORT);
      System.out.println("EchoClient n° "+id+": started");
       System.out.println("Client Socket: "+ socket);

    } catch(IOException e) {}
    // Se la creazione della socket fallisce non è necessario fare nulla

    try {
      InputStreamReader isr = new InputStreamReader(socket.getInputStream());
      in = new BufferedReader(isr);
      OutputStreamWriter osw = new OutputStreamWriter(socket.getOutputStream());
      out = new PrintWriter(new BufferedWriter(osw), true);
      start();
    } catch(IOException e1) {
      // in seguito ad ogni fallimento la socket deve essere chiusa, altrimenti
      // verrà chiusa dal metodo run() del thread
      try {
        socket.close();
      } catch(IOException e2) {}
    }
  }


  public void run() {
    try {
      for(int i =0;i <10; i++) {
        out.println("client "+id +" msg "+i);
        System.out.println("Msg sent: client "+id+" msg"+i);
        String str = in.readLine();
        System.out.println("Echo: "+str);
      }
      out.println("END");
     } catch(IOException e) {}
    try {
     System.out.println("Client "+id+" closing...");
       socket.close();
    } catch(IOException e) {}
    threadcount--;
  }

} // ClientThread

public class EchoMultiClient {
  static final int MAX_THREADS = 10;

  public static void main(String[] args) throws IOException,InterruptedException {
    InetAddress addr;
    if (args.length == 0) addr = InetAddress.getByName(null);
    else addr = InetAddress.getByName(args[0]);

    while(true) {
      if (ClientThread.threadCount() < MAX_THREADS)
        new ClientThread(addr);
      Thread.currentThread().sleep(1000);
    }
  }
} // EchoMultiClient
```

Prove:

1. aprire 2 console sulla stessa macchina:
   - java EchoServer
   - java EchoClient

2. aprire 2 console su macchine diverse:
   - java EchoServer
   - java EchoClient indMacchinaServer

3. aprire 2 console sulla stessa macchina:
   - java EchoMultiServer
   - java EchoMultiClient

4. aprire più console sulla stessa macchina:
   - java EchoMultiServer
   - java EchoClient
   - java EchoClient
   - java EchoClient
   - …

5. aprire più console su macchine diverse:
   - java EchoMultiServer
   - java EchoClient indMacchinaServer
   - java EchoClient indMacchinaServer
   - java EchoClient indMacchinaServer
   - …

…

Ulteriori proposte:

Variare il programma in modo che:

- il server mandi tanti echo quanti indicato dal client, per es.:
  - client invia msg, server salva msg e chiede quante volte deve essere ripetuto, client invia numero, server invia il numero di echo richiesti;
  - client invia msg e numero insieme, decidendo una convenzione in base alla quale il server, facendo un parsing di ciò che riceve, possa stabilire qual è il numero di echo richiesti e dove inizia e finisce il msg (es. “numero///msg”, oppure “numero msg”)
- il server mandi l’echo dove indicato dal client (in pratica il server inoltra il messaggio ricevuto); questo implica stabilire una connessione con il destinatario indicato dal client, e ci sono due possibilità:
  - il destinatario è un altro server, quindi il server che ha ricevuto il msg diventa a sua volta client di un altro server;
  - il destinatario è un altro client con cui il client che invia il msg si era in precedenza accordato e che quindi ha già stabilito una connessione con il server ed è in attesa dell’echo;
- …