Definiamo le variabili da utilizzare nella classe.

```
1  import java.io.*;
2  import java.net.*;
3  public class ClientStr {
4    String nomeServer ="localhost";
5    int portaServer   = 6789;
6    Socket miosocket;
7    BufferedReader tastiera;
8    String stringaUtente;
9    String stringaRicevutaDalServer;
10   DataOutputStream outVersoServer;
11   BufferedReader inDalServer;
```

Quindi definiamo il metodo che stabilisce la connessione con il server:

```
25   public Socket connetti(){
26     System.out.println("2 CLIENT partito in esecuzione ...");
27     try
28     {
29       // per l'input da tastiera
30       tastiera = new BufferedReader(new InputStreamReader(System.in));
31       // crea un socket
32       miosocket = new Socket(nomeServer,portaServer);
33       // miosocket = new Socket(InetAddress.getLocalHost(), 6789);
34       // associa due oggetti al socket per effettuare la scrittura e la lettura
35       outVersoServer = new DataOutputStream(miosocket.getOutputStream());
36       inDalServer    = new BufferedReader(new InputStreamReader (miosocket.getInputStream()));
37     }
38     catch (UnknownHostException e){
39         System.err.println("Host sconosciuto"); }
40     catch (Exception e)
41     {
42       System.out.println(e.getMessage());
43       System.out.println("Errore durante la connessione!");
44       System.exit(1);
45     }
46     return miosocket;
47   }
```

Abbiamo creato un socket mediante il costruttore:

```
miosocket = new Socket(nomeserver,portaserver);
```

Trattandosi di server locale avremmo anche potuto utilizzare la seguente istruzione:

```
miosocket = new Socket(InetAddress.getLocalHost(), 6789);
```

Per le comunicazioni I/O abbiamo definito tre oggetti, il primo della classe BufferedReader per effettuare l'input da tastiera e gli altri due, rispettivamente, della classe DataOutputStream e BufferedReader per scrivere e leggere sul socket.

Definiamo un secondo metodo che effettua la conversazione:

- leggiamo la stringa inserita dall'utente e la scriviamo sullo stream del miosocket: quindi ci mettiamo in attesa della risposta del server mediante la inDalserver.readLine();

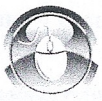
- quando riceviamo la risposta la visualizziamo sullo schermo e terminiamo l'elaborazione chiudendo la porta con miosocket.close().

```
12  public void comunica() {
13    try
14    {
15      System.out.println("4 ...  inserisci la stringa da trasmettere al server:"+'\n');
16      stringaUtente = tastiera.readLine();
17      //la spedisco al server
18      System.out.println("5 ... invio la stringa al server e attendo ...");
19      outVersoServer.writeBytes( stringaUtente+'\n'); // OUT PUT CERTA STRINGA NEL CANALE
20      //leggo la risposta dal server
21      stringaRicevutaDalServer=inDalServer.readLine();
22      System.out.println("8 ... risposta dal server "+'\n'+stringaRicevutaDalServer );
23      // chiudo la connessione
24      System.out.println("9 CLIENT: termina elaborazione e chiude connessione" );
25      miosocket.close(); //CHIUDO CONNESSIONE CLIENT + SERVER
26    }
27    catch (Exception e)
28    {
29      System.out.println(e.getMessage()); // GUARDA PRIMA
30      System.out.println("Errore durante la comunicazione col server!");
31      System.exit(1);
32    }
33  }
34
```

Concludono il metodo le istruzioni del costrutto catch che ci segnalano gli eventuali errori.

Il main è il seguente:

```
58  public static void main(String args[]) {
59    ClientStr cliente = new ClientStr(); // CLIENT OGGETTO CLIENTSTR E STRINGA CLIENTE
60    cliente.connetti();
61    cliente.comunica();
62  }
```



Prova adesso!

Sulla base del client sopra descritto, scrivi un tuo client predisposto per connettersi a un server sempre presente sul tuo pc che trasmette e riceve una stringa.
La verifica della connessione verrà fatta al termine della successiva esercitazione di laboratorio oppure utilizzando come server ServerStr.class che puoi scaricare dal sito www.hoepliscuola.it nella cartella materiali nella sezione riservata al presente volume (file UD-SOCKET.rar).

ESERCITAZIONI DI LABORATORIO 2

JAVA SOCKET: REALIZZAZIONE DI UN SERVER TCP

Realizzazione di un server

In questa esercitazione realizzeremo il server corrispondente al client definito nella precedente esercitazione: dopo aver definito le variabili:

```
1  import java.io.*;
2  import java.net.*;
3  import java.util.*;
4
5  public class ServerStr
6  {
7    ServerSocket server       = null;
8    Socket client             = null;
9    String stringaRicevuta    = null;
10   String stringaModificata = null;
11   BufferedReader   inDalClient;
12   DataOutputStream outVersoClient;
```

Definiamo un metodo che effettua la connessione: per prima cosa si deve creare un oggetto ServerSocket che indichi il numero di porta sulla quale si mette in ascolto (nel nostro esempio, 6789) e mediante il metodo accept() aspetta un client.

```
13
14   public Socket attendi()
15   {
16     try
17     {
18       System.out.println("1 SERVER partito in esecuzione ...");
19       // creo un server sulla porta 6789
20       server = new ServerSocket(6789);
21       // rimane in attesa di un client
22       client = server.accept();
23       // chiudo il server per inibire altri client
24       server.close();
25       //associo due oggetti al socket del client per effettuare la scrittura e la lettura
26       inDalClient = new BufferedReader(new InputStreamReader (client.getInputStream()));
27       outVersoClient = new DataOutputStream(client.getOutputStream());
28     }
```

L'esecuzione dell'istruzione successiva avviene solamente se si è instaurata la connessione col client su tale porta: prima di iniziare la comunicazione, il server deve chiudere il ServerSocket per inibire ad altri client il collegamento, in quanto quell'indirizzo è già utilizzato (e ora noi stiamo implementando una comunicazione Unicast).

Conclude il metodo la gestione dell'errore di connessione:

```
29      catch (Exception e)
30      {
31        System.out.println(e.getMessage());
32        System.out.println("Errore durante l'istanza del server !");
33        System.exit(1);
34      }
35      return client;
36    }
```

Il metodo che effettua la comunicazione è molto simile a quello descritto per il client: dopo il saluto di benvenuto, il server si pone in attesa di lettura di una stringa dal canale di "input dal client"; al suo arrivo, la converte in maiuscolo e la trasmette al client.
Quindi si commiata dal client e chiude la connessione terminando la sua elaborazione.

```
38    public void comunica()
39    {
40      try
41      {
42        // rimango in attesa della riga trasmessa dal client
43        System.out.println("3 benvenuto client, scrivi una frase e la trasformo in maiuscolo. Attendo ...");
44        stringaRicevuta = inDalClient.readLine();
45        System.out.println("6 ricevuta la stringa dal cliente : "+stringaRicevuta);
46
47        //la modifico e la rispedisco al client
48        stringaModificata=stringaRicevuta.toUpperCase();
49        System.out.println("7 invio la stringa modificata al client ...");
50        outVersoClient.writeBytes(stringaModificata+'\n');
51
52        //termina elaborazione sul server : chiudo la connessione del client
53        System.out.println("9 SERVER: fine elaborazione  ... buona notte!");
54        client.close();
55      }
```

Il main è il seguente:

```
63
64    public static void main(String args[]) {
65      ServerStr servente = new ServerStr();
66      servente.attendi ();
67      servente.comunica();
68    }
```

Mandiamo ora in esecuzione le due classi: dato che utilizziamo la medesima macchina per entrambe le funzioni, dovremo attivare due istanze di BlueJ: una che ci permetterà di mandare in esecuzione il server e l'altra per il client.

L'output si presenta in due finestre diverse: per meglio comprendere la corretta sequenza di esecuzione delle operazioni queste sono state numerate in ordine progressivo. Ricordiamo di mandare in esecuzione prima il server altrimenti il client termina immediatamente con il seguente messaggio: