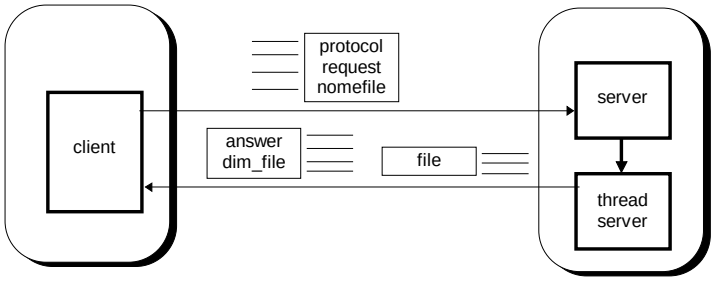
Architettura Client-Server

1.il client ***manda una richiesta*** al server
2.il server (in attesa) ***riceve*** la richiesta
3.il server ***esegue*** il servizio richiesto (*generando un thread concorrente*)
4.il server ***manda una risposta*** ed eventualmente dei dati
5.il client ***riceve*** la risposta ed eventualmente i dati

Necessità di accordarsi su un ***protocollo di richiesta/risposta***



Socket in Java

Distinzione dei ruoli

Classi
- `java.net.Socket`
- `java.net.ServerSocket`

Socket
- implementa una generica socket
- connessione di tipo stream

ServerSocket
- implementa una socket che consente la ***ricezione di richieste*** di servizio

Input e output
- attraverso *stream*
- metodi per ottenere *stream di basso livello* che possono essere utilizzati *incapsulati in stream di livello più alto*

Classe Socket

Metodi

- `public` **`Socket`**`(String` **`host`**`, int` **`port`**`)`

crea una socket di tipo ***stream*** e la connette alla porta `port` dell'host `host`

- `public InputStream` **`getInputStream`**`()`
- `public OutputStream` **`getOutputStream`**`()`

ritornano uno ***stream di input (output)*** per leggere dalla (scrivere sulla) socket

- `public synchronized void` **`setSoTimeout`**`(int timeout)`

imposta un ***timeout*** per le operazioni di lettura dalla socket (0 = attesa infinita)

- `public synchronized void` **`close`**`()`

chiude la socket

Anche

`java.net.DatagramSocket`
per connessioni di tipo ***datagram***

Classe ServerSocket

Metodi

- `public` **`ServerSocket`**`(int` **`port`**`)`

crea una socket server sulla porta `port`, che accetta fino a 50 richieste di servizio
se `port` = 0, viene scelta una porta libera

- `public` **`ServerSocket`**`(int` **`port`**`, int` **`backlog`**`, InetAddress` **`bindAddr`**`)`

crea una socket server sulla porta `port`, collegata all'indirizzo `bindAddr` ed accetta fino a `backlog` richieste di servizio

- `public Socket` **`accept`**`()`

mette la ***socket in condizioni di ascolto***, in attesa di richieste di connessione; restituisce una socket per colloquiare con il richiedente

- `public synchronized void` **`close`**`()`

chiude la socket

Implementazione di un server HTTP

Entità
1. richiesta
2. risposta
3. server
4. thread che esegue il servizio (per avere concorrenza)
5. client

Classi
1. Request
2. Answer
3. ServerHTTP
4. ServerThread
5. ClientHTTP

Definizioni di **costanti**
- classe HTTP

Classi inserite nel package **serverHTTP**

Definizione di costanti

```
package serverHTTP;

public class HTTP {
  // porta su cui il server è in ascolto
  static final int HTTP_PORT = 5557;

  // protocolli
  static final int HTTP = 0;
  static final int FTP = 1;
  static final int TELNET = 2;
  static final int NNTP = 3;

  // comandi HTTP
  static final int GET = 0;
  static final int IF_NOT_MODIFIED = 1;

  // risposte HTTP
  static final int FILE_NOT_FOUND = 0;
  static final int FILE_NOT_MODIFIED = 1;
  static final int REQ_OK = 2;

  // dimensione del buffer
  static final int PACCHETTO = 100;}
```

Richiesta e risposta

```
package serverHTTP;

public class Request implements java.io.Serializable
{
  public int protocol;
  public int request;
  public String nomefile;

  public Request(int p, int r, String n)
  { protocol = p; request = r; nomefile = n;}
}
```

```
package serverHTTP;

public class Answer implements java.io.Serializable
{
  public int answer;
  public long dim_file;

  public Answer(int a, long d)
  { answer = a; dim_file = d; } }
```


Server HTTP - 1

```
package serverHTTP;

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

public class ServerHTTP
{
  static final int MAX_REQ = 5;

  public static void main(String args[])
  {
    try
    {
      // crea e connette la socket
      ServerSocket so = new ServerSocket(HTTP.HTTP_PORT, MAX_REQ);
      System.out.print("Server HTTP: inizializzato sulla porta ");
      System.out.println(so.getLocalPort());
```

Server HTTP - 2

```
    while(true)
       {
         // si mette in ascolto
         Socket newso = so.accept();
         System.out.println("Server HTTP: ricevuta richiesta");

         // esegue un thread in modo parallelo
         new ServerThread(newso).start();
       }
     } catch (IOException e) {System.out.println("ERRORE: " + e);}
  }
} // class ServerHTTP
```


Esecutore del servizio - 1

```
package serverHTTP;

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

class ServerThread extends Thread
{
  private Socket s;
  private Request richiesta;
  private Answer risposta;

  public ServerThread(Socket s)
  { this.s = s;}

  public void run()
  {
    byte buffer[] = new byte[HTTP.PACCHETTO];
    try
    {
      // ottiene gli stream di I/O dalla socket
      ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
      ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
```

Esecutore del servizio - 2

```
      // legge la richiesta
      richiesta = (Request)ois.readObject();
      System.out.println(richiesta.toString());

switch (richiesta.protocol)
      {
        case HTTP.HTTP:
          switch (richiesta.request)
          {
            case HTTP.GET:
              File f = new File(richiesta.nomefile);

              // controlla che il file esista
              if (f.exists())
              {
                Answer risposta = new Answer(HTTP.REQ_OK, f.length());
                oos.writeObject(risposta);
                oos.flush();
                DataOutputStream dos = new
   DataOutputStream(s.getOutputStream());
                FileInputStream fis = new FileInputStream(f);
```


Esecutore del servizio - 3

```
                // manda il contenuto del file
                for (int i=0; i<(risposta.dim_file/HTTP.PACCHETTO); i++)
                {
                  fis.read(buffer);
                  dos.write(buffer);
                }
                fis.read(buffer, 0, (int)risposta.dim_file%HTTP.PACCHETTO);
                dos.write(buffer, 0,
   (int)risposta.dim_file%HTTP.PACCHETTO);
                fis.close();
              }
              else // il file non esiste
              {
                Answer risposta = new Answer(HTTP.FILE_NOT_FOUND, 0);
                oos.writeObject(risposta);
              }
            break; // GET
```

Esecutore del servizio - 4

```
            case HTTP.IF_NOT_MODIFIED:
              // altre istruzioni...
            break; //IF_NOT_MODIFIED
          } // switch request
        } // switch protocol
        s.close();   //chiudo socket
    } catch (Exception e) {System.out.println("ERRORE: "+e);}
  } // run()
} // class ServerThread
```


Client - 1

```
package serverHTTP;

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

public class ClientHTTP
{
  public static void main(String args[])
  {
    String nomeurl, nomeserver, nomefile;

    try
    {
      while (true)
      {
        System.out.print("Dammi l'indirizzo URL: ");
        BufferedReader r = new BufferedReader(new
   InputStreamReader(System.in));
        nomeurl = r.readLine();
        nomeserver = nomeurl.substring(0, nomeurl.indexOf('/'));
        nomefile = nomeurl.substring(nomeurl.indexOf('/') + 1);
        System.out.println("Server: " + nomeserver + "; file: " +
   nomefile);
```

Client - 2

```
        // crea una socket e la connette a host e porta
        Socket s = new Socket(nomeserver, HTTP.HTTP_PORT);

        // ottiene gli stream di I/O dalla socket
        ObjectOutputStream oos = new
   ObjectOutputStream(s.getOutputStream());
        ObjectInputStream ois = new ObjectInputStream(s.getInputStream());

        // invia la richiesta
        Request richiesta = new Request(HTTP.HTTP, HTTP.GET, nomefile);
        oos.writeObject(richiesta);

        // legge la risposta
        Answer risposta = (Answer)ois.readObject();

        if (risposta.answer == HTTP.REQ_OK)
        {
          byte buffer[] = new byte[HTTP.PACCHETTO];

          // ottiene un stream di dati in input
          DataInputStream dis = new DataInputStream(s.getInputStream());
```


Client - 3

```
          // legge i byte in arrivo dalla socket
          for (int i=0; i<(risposta.dim_file/HTTP.PACCHETTO); i++)
          {
            dis.read(buffer);
            System.out.print(new String(buffer));
          }
          dis.read(buffer, 0, (int)risposta.dim_file%HTTP.PACCHETTO);
          System.out.println(new String(buffer, 0,
   (int)risposta.dim_file%HTTP.PACCHETTO));

        }
        else
        {
          System.out.println("File " + nomefile + " non trovato");
        }
        s.close();
      } // fine while
    } catch (Exception e) {System.out.println("ERRORE: " + e);}
  }
} // class ClientHTTP
```