Reti e Laboratorio III
Modulo Laboratorio III
AA. 2025-2026

docente: Laura Ricci

laura.ricci@unipi.it

Lezione 5
InetAddress
Stream Sockets for clients

24/10/2025

NETWORK APPLICATIONS

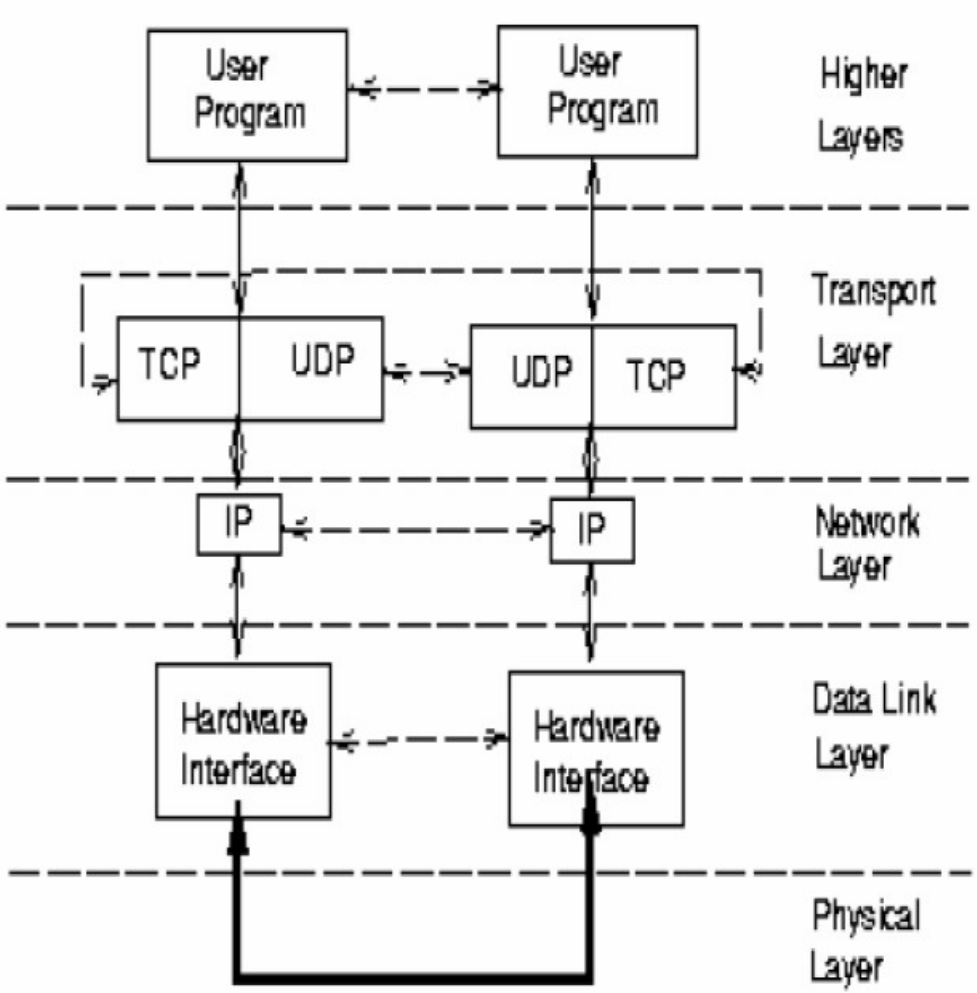
alcune "network killer applications"

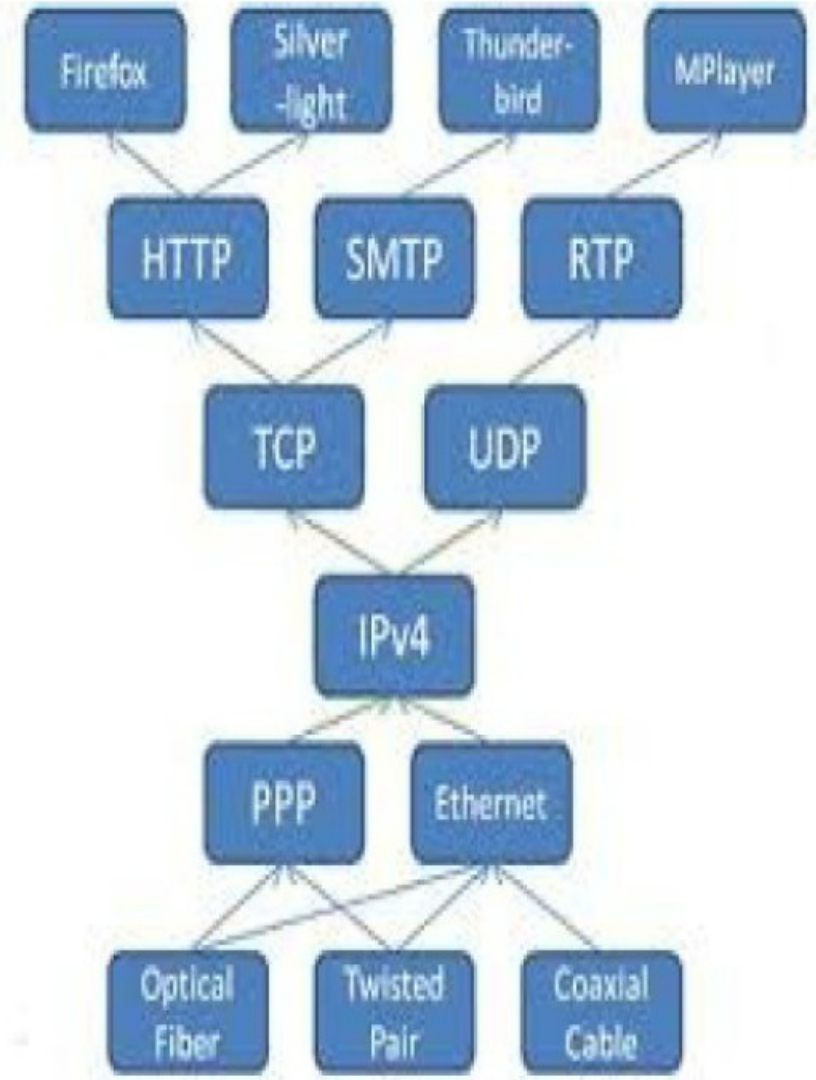
- web browser
- SSH
- email
- social networks
- teleconferences (Skype, Zoom, Meet, Teams,...)
- program development environments: GIT
- collaborative work: Overleaf
- multiplayer games: War of Warcraft
- P2P File sharing: Bittorrent
- blockchain: cryptocurrencies (Bitcoin, Ethereum)
- Decentralized Finance, borrow, lending, flash loans
- Play to earn

NETWORK APPLICATIONS

- due o più processi (non thread!) in esecuzione su hosts diversi, distribuiti geograficamente sulla rete, comunicano e cooperano per realizzare una funzionalità globale:
  - cooperazione: scambio informazioni utile per perseguire l'obiettivo globale, quindi implica comunicazione
  - comunicazione: utilizza protocolli, ovvero insieme di regole che i partners devono seguire per comunicare correttamente
- in questo corso utilizzeremo i protocolli di livello trasporto:
  - connection-oriented: `TCP, Trasmission Control Protocol`
  - connectionless: `UDP, User Datagram Protocol`

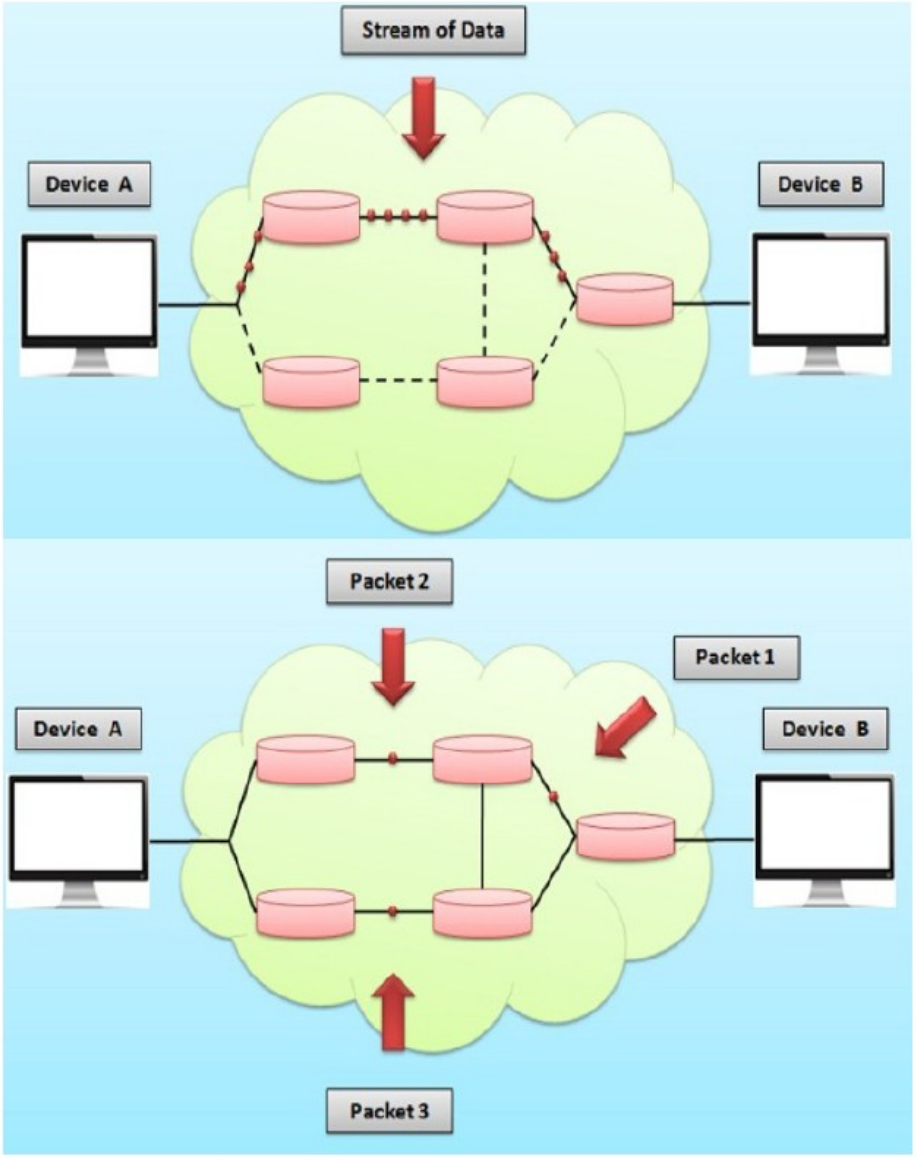
NETWORK LAYERS: DAL MODULO DI TEORIA





TIPI DI COMUNICAZIONE

- Connection Oriented (TCP)
  - come una chiamata telefonica
  - una connessione stabile (canale di comunicazione dedicato) tra mittente e destinatario
  - stream socket
- Connectionless (UDP)
  - come l'invio di una lettera
  - non si stabilisce un canale di comunicazione dedicato
  - ogni messaggio viene instradato in modo indipendente dagli altri
  - datagram socket



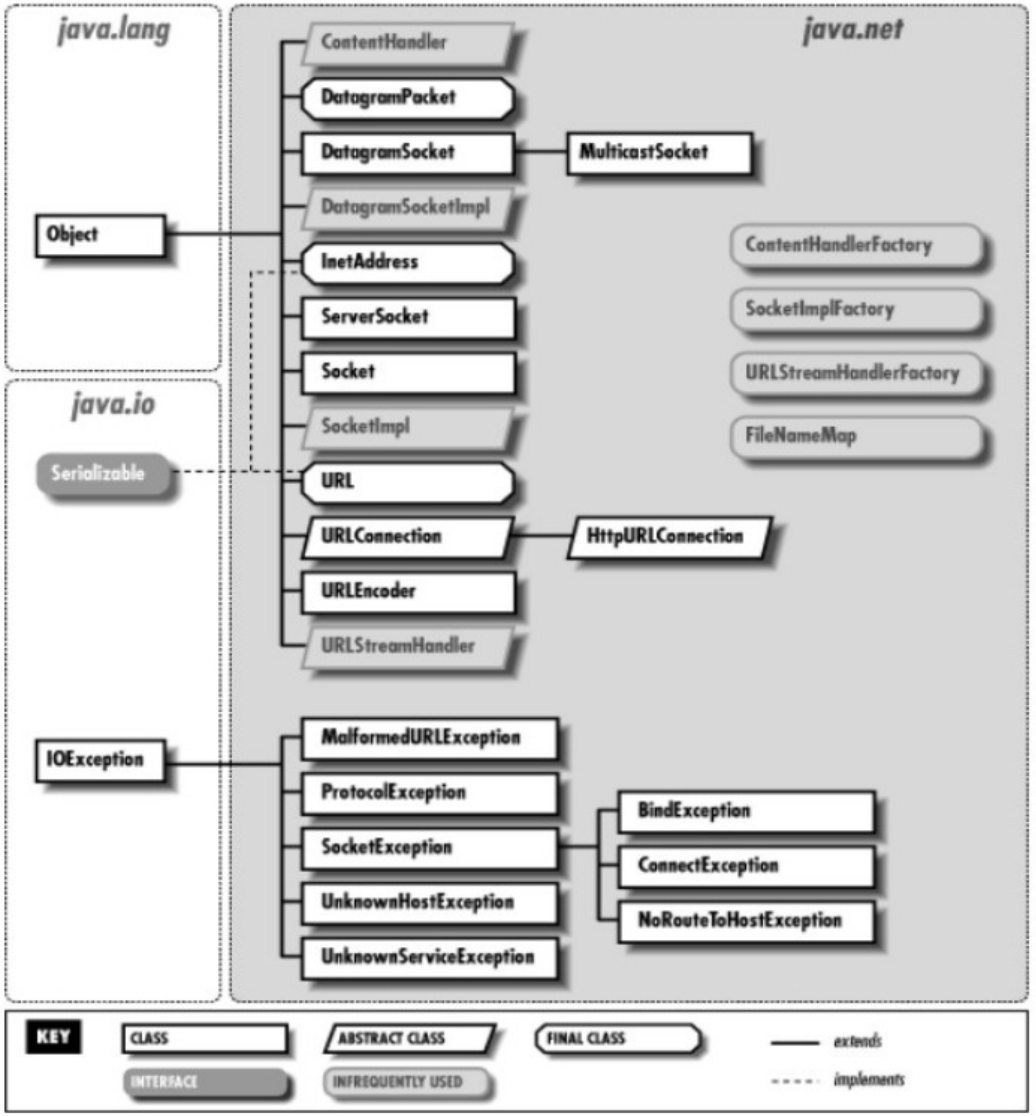
JAVA.NET: NETWORKING IN JAVA

connection-oriented

- connessione modellata come stream
- asimmetrici
  - client side: `Socket` class
  - server side:
    - `ServerSocket` class
    - `Socket` class

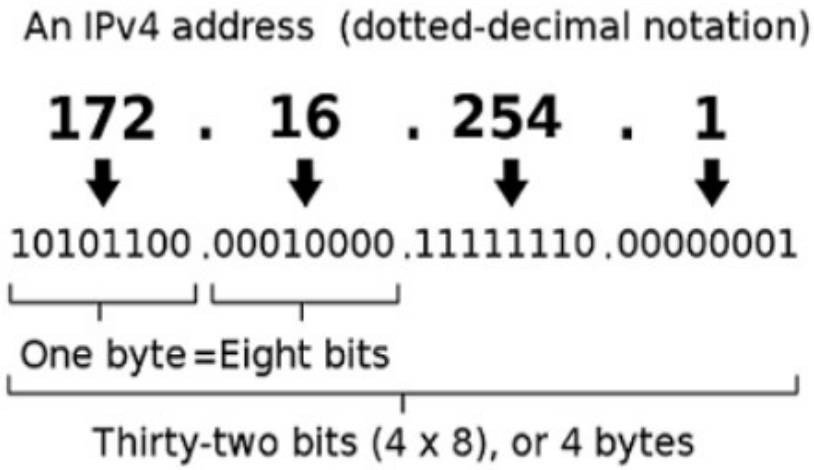
connectionless

- basata su invio di pacchetti
- simmetrici: sia per il client che per il server
  - `DatagramSocket`
  - `DatagramPacket`

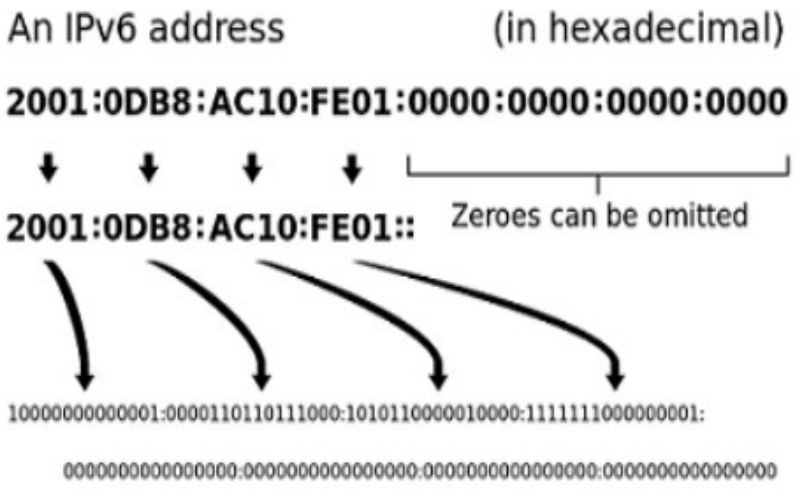


IP (INTERNET PROTOCOL) ADDRESS

- `IPV4, 4 bytes:` 2^{32} indirizzi
  - dotted quad form
  - ogni byte interpretato come un numero decimale senza segno
  - alcuni indirizzi riservati, loopback address: `127.0.0.0`, broadcast `255.255.255.255`



- `IPV6, 16 bytes:` 2^{128} indirizzi,
  - 8 blocchi di 4 cifre esadecimali



DOMAIN NAMES

- gli indirizzi `IP` semplificano l'elaborazione effettuata dai routers, ma sono poco leggibili per gli utenti della rete
- soluzione
  - assegnare un nome simbolico unico ad ogni host della rete
  - utilizzare uno spazio di nomi gerarchico

    `fujih0.cli.di.unipi.it`

(host `fuji` presente nell'aula H alla postazione `0`, nel dominio `cli.di.unipi.it`)

  - livelli della gerarchia separati dal punto
  - nomi interpretati da destra a sinistra
  - un nome può essere mappato a più indirizzi `IP`
- indirizzi a lunghezza fissa verso nomi a lunghezza variabili
- Domain Name System (DNS) traduce nomi in indirizzi `IP`

LA CLASSE INETADDRESS

```
public class InetAddress extends Object implements Serializable
```

- può gestire sia indirizzi IPv4 che IPv6
- usata per incapsulare in un unico oggetto di tipo `InetAddress`
  - l'indirizzo IP numerico: `byte[] address`
  - il nome di dominio per quell'indirizzo: `String`
- la classe non contiene alcun costruttore,
- allora, come posso creare oggetti di tipi `InetAddress`?
  - si utilizza una factory con metodi statici
  - i metodi si connettono al DNS per risolvere un hostname, ovvero trovare l'indirizzo IP ad esso corrispondente: necessaria una connessione di rete
  - possono sollevare `UnKnownHostException`, se non riescono a risolvere il nome dell'host

LA CLASSE INETADDRESS

getByName() lookup dell'indirizzo di un host

```
import java.net.*;
public class FindIP {
 public static void main (String[] args) {
    try {
        InetAddress address = InetAddress.getByName("www.unipi.it");
        System.out.println(address);
        } catch (UnknownHostException ex) {
                        System.out.println("Could not find www.unipi.it"); }} }
```

```
$ java FindIP
www.unipi.it/131.114.21.42
```

getLocalHost() lookup dell'indirizzo locale

```
import java.net.*;
public class MyAddress {
  public static void main (String [] args)
      {try {
         InetAddress address = InetAddress.getLocalHost();
         System.out.println(address);
           } catch (UnknownHostException ex) {
                   System.out.println("Could not find this computer's address"); }}}
```

```
$java MyAddress
DESKTOP-R5C46F3/192.168.1.196
```

LA CLASSE INETADDRESS

- `getAllByName()` lookup di tutti gli indirizzi di un host

```
import java.net.*;
public class FindAllIP {
 public static void main (String[] args) {
  try { InetAddress [] addresses = InetAddress.getAllByName("www.repubblica.it");
    for(InetAddress address:addresses)
        { System.out.println(address); }
 } catch (UnknownHostException ex) {
                    System.out.println("Could not find www.repubblica.it");}}}
```

```
$ java FindAllIP
www.repubblica.it/18.66.196.45
www.repubblica.it/18.66.196.118
www.repubblica.it/18.66.196.94
www.repubblica.it/18.66.196.112
```

INETADDRESS: CACHING

- i metodi descritti effettuano caching dei nomi/indirizzi risolti
  - l'accesso al DNS è un'operazione potenzialmente molto costosa
  - nomi risolti con i dati nella cache, quando possibile (di default: per sempre)
  - anche i tentativi di risoluzione non andati a buon fine in cache
- permanenza dati nella cache:
  - 10 secondi se la risoluzione non ha avuto successo, spesso il primo tentativo di risoluzione fallisce per un time out...
  - tempo illimitato altrimenti
  - problemi: indirizzi dinamici
- controllo dei tempi di permanenza in cache

```
java.security.Security.setProperty
                        ("networkaddress.cache.ttl","0");
```

- per i tentativi non andati a buon fine: networkaddress.cache.negative.ttl

CACHING DI INDIRIZZI IP: "UNDER THE HOOD"

```
import java.net.InetAddress; import java.net.UnknownHostException;
import java.security.*;
public class Caching {
   public static final String CACHINGTIME="0";
   public static void main(String [] args) throws InterruptedException
     {Security.setProperty("networkaddress.cache.ttl",CACHINGTIME);
      long time1 = System.currentTimeMillis();
      for (int i=0; i<1000; i++){
        try {System.out.println(
             InetAddress.getByName("www.cnn.com").getHostAddress());}
        catch (UnknownHostException uhe)
                { System.out.println("UHE");} }
        long time2 = System.currentTimeMillis();
        long diff=time2-time1; System.out.println("tempo trascorso e'"+diff);}}
```

CACHINGTIME=0 tempo trascorso è 545

CACHINGTIME=1000 tempo trascorso è 85

INETADDRESS: FACTORY METHODS

- metodi statici di una classe che restituiscono oggetti di quella classe
- i seguenti metodi contattano il DNS per la risoluzione di indirizzo/hostname

```
static InetAddress getLocalHost() throws UnknownHostEception
static InetAddress getByName (String hostname) throws UnknownHostException
static InetAddress [] getAllByName (String hostName) throws UnknownHostException
static InetAddress getLoopBackAddress()
```

- i seguenti metodi statici costruiscono oggetti di tipo `InetAddress`, ma non contattano il `DNS` (utile se `DNS` non disponibile e conosco indirizzo/host)
- non danno nessuna garanzia sulla correttezza di hostname/IP, `UnknownHostException` sollevata solo se l'indirizzo è malformato

```
static InetAddress getByAddress(byte IPAddr[]) throws UnknownHostException
static InetAddress getByAddress (String hostName, byte IPAddr[])
                                                  throws UnknownHostException
```

INETADDRESS: INSTANCE METHODS

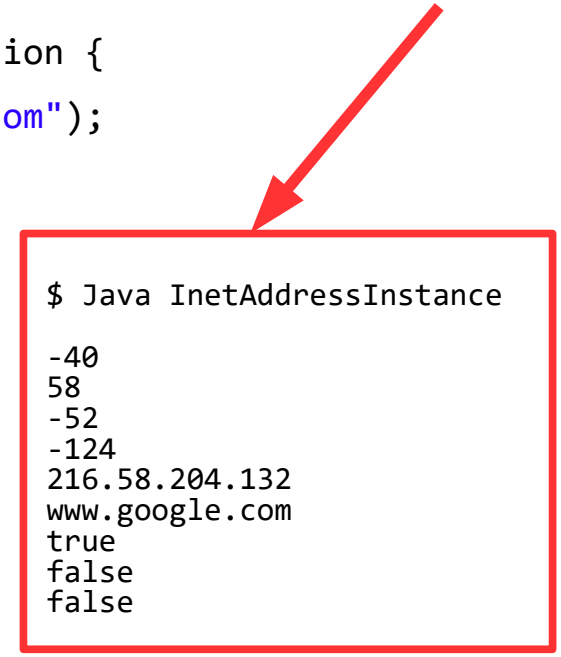
- la classe `InetAddress` ha moltissimi "metodi di istanza" che possono essere utilizzati sull'istanza di un oggetto `InetAddress` (costruito con uno dei metodi della `Factory`)

```
boolen equals(Object other)
byte [] getAddress()
String getHostAddress()
String getHostName()
boolean isLoopBackAddress()
boolean isMulticastAddress()
boolean isReachable()
String toString ()
```

…. e molto altri (vedere le `API`)

INETADDRESS: INSTANCE METHODS

```
import java.net.*; import java.util.Arrays; import java.io.*;
public class InetAddressIstance {
public static void main (String[] args) throws IOException {
  InetAddress ia1 = InetAddress.getByName("www.google.com");
  byte [] address = ia1.getAddress();
  for (int i=0; i<address.length; i++)
      {System.out.println(address[i]);};
  System.out.println(ia1.getHostAddress());
  System.out.println(ia1.getHostName());
  System.out.println(ia1.isReachable(1000));
  System.out.println(ia1.isLoopbackAddress());
  System.out.println(ia1.isMulticastAddress());
```



osservare:

- `GetAddress` restituisce gli ottetti dell'indirizzo
  - come signed byte
  - nella posizione `0` del vettore viene memorizzato il byte dell'ottetto più significativo (rappresentazione `big endian`, byte più significativo per primo)
- `GetHostAddress` restituisce l'indirizzo IP

SPAM CHECKER CON INETADDRESS

- diversi servizi monitorano gli spammers: real-time black-hole lists (RTBLs)
  - ad esempio: `zen.spamhaus.org`
  - mantengono una lista di indirizzi IP che risultano, probabilmente, degli spammers
- per identificare se un indirizzo IP corrisponde ad uno spammer:
  - invertire i bytes dell'indirizzo IP
  - concatenare il risultato alla stringa `zen.spamhaus.org`
  - esegui un DNS look-up con il nome di dominio ottenuto
  - la query ha successo se e solo se l'indirizzo IP corrisponde ad uno spammer
  - es una query DNS su `45.113.0.203.sbl.spamhaus.org` ha successo se l'indirizzo `203.0.113.45` è uno spammer

SPAM CHECKER CON INETADDRESS

```
import java.net.*;
public class SpamCheck {
  public static final String BLACKHOLE = "zen.spamhaus.org";
  public static void main(String[] args) throws  UnknownHostException
       { for (String arg: args) {
                if (isSpammer(arg)) {
                    System.out.println(arg + " is a known spammer.");
                   } else {
                    System.out.println(arg + " appears legitimate.");  }}}

                    // continua nella slide successiva
```

SPAM CHECKER CON INETADDRESS

```
private static boolean isSpammer(String arg) {
  try { InetAddress address = InetAddress.getByName(arg);
        byte [ ] quad = address.getAddress();
        String query = BLACKHOLE;
        for (byte octet : quad) {
            int unsignedByte = octet < 0 ? octet + 256 : octet;
            query = unsignedByte + "." + query;
        }
        InetAddress.getByName(query);
        return true;
   } catch (UnknownHostException e) { return false; }}}
```

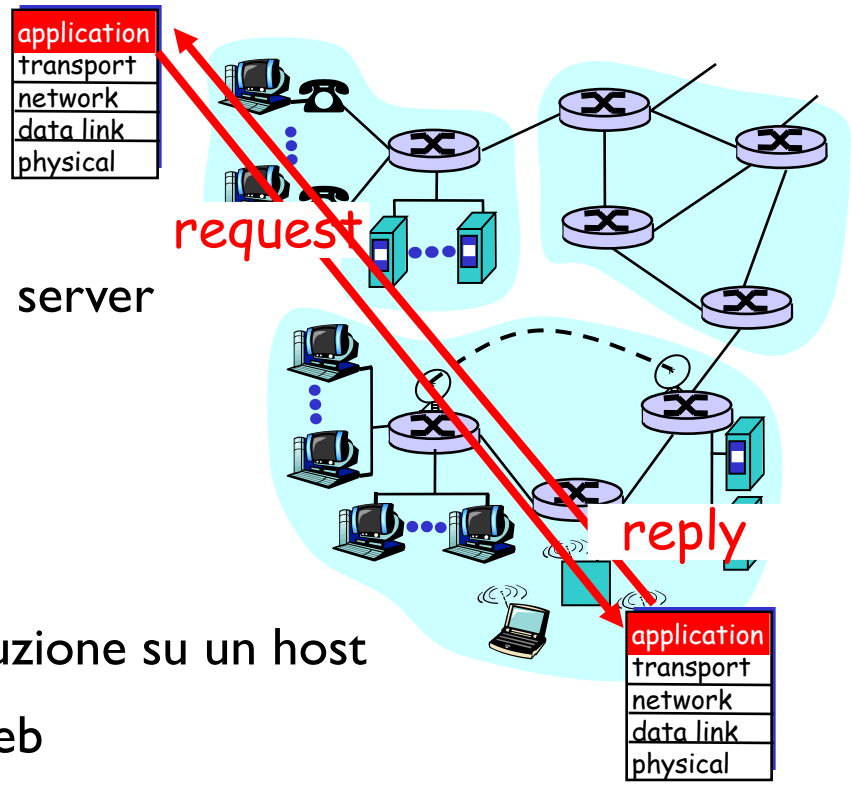
```
$java SpamCheck  23.45.65.88 141.250.89.99
203.0.113.45

23.45.65.88 appears legitimate.
141.250.89.99 appears legitimate.
203.0.113.45 is a known spammer
```

IL PARADIGMA CLIENT/SERVER

servizio:

- software in esecuzione su una o più macchine
- fornisce l'astrazione di un insieme di operazioni



client:

- un software che sfrutta servizi forniti dal server

  web client browser

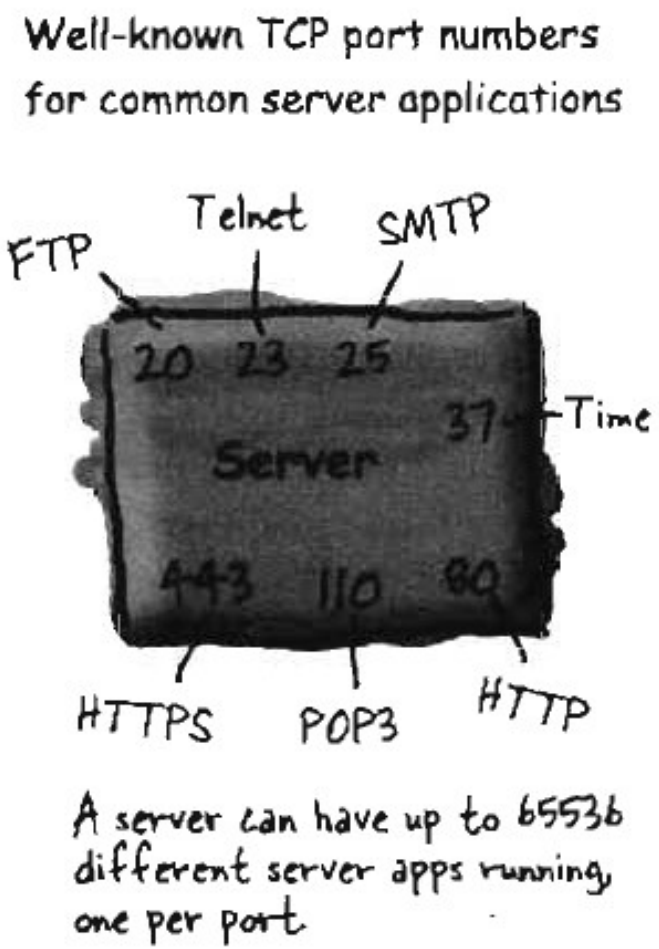
  e-mail client mail-reader

server:

- istanza di un particolare servizio in esecuzione su un host
- ad esempio: server Web invia la pagina web richiesta, mail server consegna la posta al client

IDENTIFICARE I SERVIZI

- occorre specificare:
  - l'host, tramite indirizzo `IP` (la rete all'interno della quale si trova l'host + l'host all'interno della rete)
  - la porta individua un servizio tra I tanti servizi (es: `e-mail`, `ftp`, `http`,...) attivi su un host
- ogni servizio individuato da una porta
  - intero tra `1` e `65535` (per `TCP` ed `UDP`)
  - non un dispositivo fisico, ma un'astrazione per individuare i singoli servizi (processi)
- porte `1-1023`: riservate per well-known services.

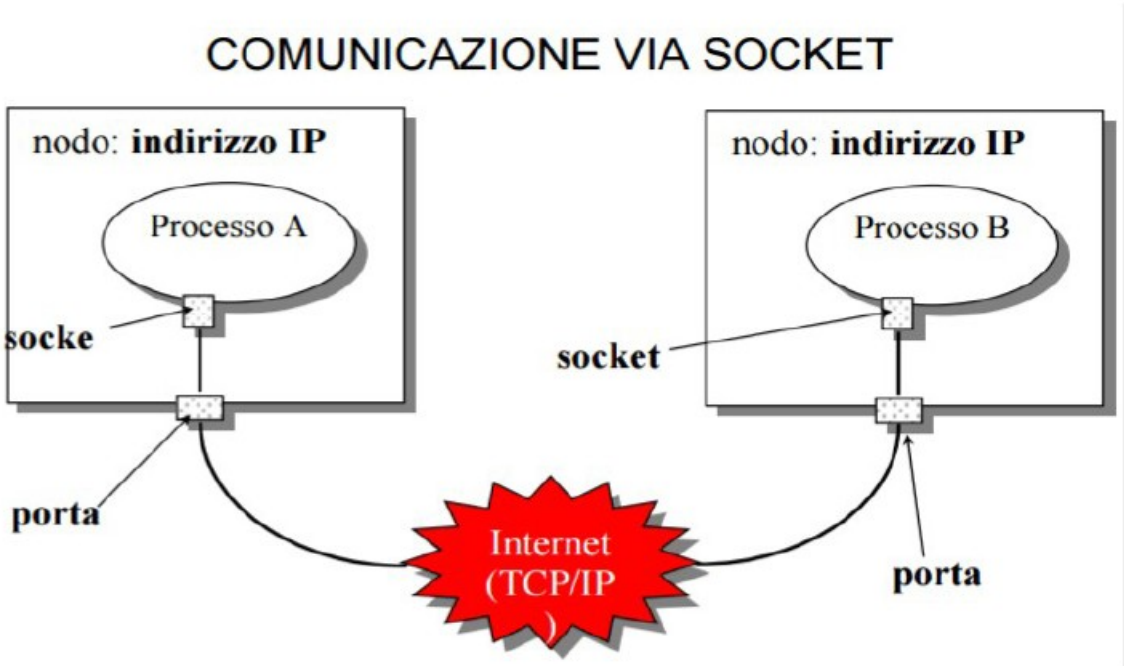


CONNETTERSI AD UN SERVIZIO

- socket: uno standard per connettere dispositivi distribuiti, diversi, eterogenei
- termine utilizzato in tempi remoti in telefonia
  - la connessione tra due utenti veniva stabilita tramite un operatore
  - l'operatore inseriva fisicamente i due estremi di un cavo in due ricettacoli (sockets)
  - un socket per ogni utente

SOCKET: UNO "STANDARD" DI COMUNICAZIONE

- una presa "standard" a cui un processo si può collegare per spedire dati
- un endpoint sull'host locale di un canale di comunicazione da/verso altri hosts
- introdotti in `Unix BSD 4.2`
- collegati ad una porta locale



SOCKET LATO CLIENT

- per usufruire di un servizio, il client apre un socket individuando
  - host + porta che identificano il servizio
  - invia/riceve messaggi su/da uno stream
- in JAVA: `java.net.Socket:` usa codice nativo per comunicare con lo stack TCP locale

```
public socket(InetAddress host, int port) throws IOException
```

- crea un socket su una porta effimera e tenta di stabilire, tramite esso, una connessione con l'host individuato da `InetAddress`, sulla porta `port`.
- se la connessione viene rifiutata, lancia una eccezione di IO

```
public socket (String host, int port) throws
                                    UnKnownHostException, IOException
```

come il precedente, l'host è individuato dal suo nome simbolico: interroga automaticamente il DNS)

PORT SCANNER

- ricerca quale delle prime 1024 porte di un host è associata ad un servizio

```
import java.net.*;
import java.io.*;
public class LowPortScanner {
  public static void main(String[] args) {
    String host = args.length > 0 ? args[0] : "localhost";
    for (int i = 1; i < 1024; i++) {
      try {
          Socket s = new Socket(host, i);
          System.out.println("There is a server on port " + i + " of " + host);
          s.close();
      } catch (UnknownHostException ex) {
                  System.err.println(ex);
                  break;
      } catch (IOException ex) {
      // must not be a server on this port
     }}}}
```

```
$java LowPortScanner

There is a server on port 80  of localhost
There is a server on port 135 of localhost
There is a server on port 445 of localhost
There is a server on port 843 of localhost
```

PORT SCANNER: ANALISI

- il client richiede un servizio tentando di creare un socket su ognuna delle prime 1024 porte di un host
  - nel caso in cui non vi sia alcun servizio attivo, il socket non viene creato e viene invece sollevata un'eccezione
- il programma precedente effettua 1024 interrogazioni al DNS, una per ogni socket che tenta di creare, impiega molto tempo
- come ottimizzare il programma? utilizzare un diverso costruttore

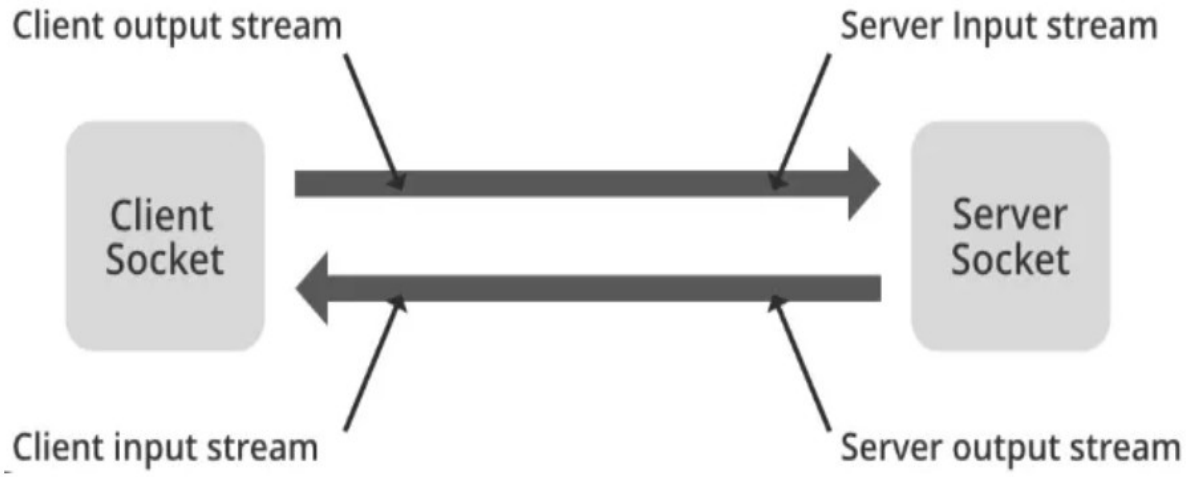
```
public Socket(InetAddress host, int port) throws IOException
```

  - viene utilizzato l' `InetAddress` invece del nome dell'host per costruire i sockets
  - costruire l'`InetAddress` invocando `InetAddress.getByName` una sola volta, prima di entrare nel ciclo di scanning

STREAM PER CONNESSIONI TCP

- una volta stabilita una connessione con il server, il client scambia dati con esso
- la connessione è modellata come uno stream.
- stream (o flusso): JAVA utilizza principalmente il concetto di stream per la gestione dell'I/O
  - I/O: una delle parti più complesse di un linguaggio
- astrazioni JAVA per l'I/O
  - Stream
  - File: per manipolare descrittori di files
  - Channels (NIO)
- perchè gli stream sono importanti per questo corso?
  - le connessioni TCP possono essere modellate in JAVA con streams
  - saranno utilizzati anche per la generazione di pacchetti UDP

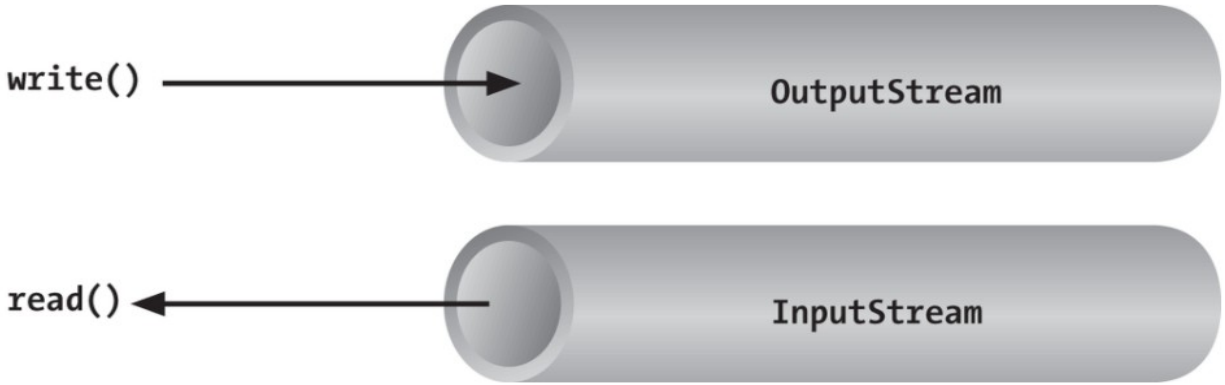
STREAM PER CONNESSIONI TCP



- socket: endpoint per inviare/ricevere dati
  - astrazione che "maschera" complessità della rete
- stream: astrazione che modella la connessione effettuata tramite un socket TCP

L'ASTRAZIONE DEGLI STREAM

- uno stream rappresenta una connessione tra un programma JAVA ed un dispositivo esterno (file, buffer di memoria, connessione di rete,...)
- un flusso di informazione di lunghezza illimitata



- un "tubo" tra una sorgente ed una destinazione (dal programma ad un dispositivo e viceversa)
- l'applicazione inserisce/legge dati ad/da un capo dello stream
- i dati fluiscono da/verso l'altra estremità

JAVA STREAMS: CARATTERISTICHE GENERALI

- accesso sequenziale
- mantengono l'ordinamento FIFO
- one way: read only oppure write only (a parte i file ad accesso random)
  - se un programma ha bisogno di dati in input ed output, è necessario aprire due stream, uno in input, l'altro in output
- bloccanti: quando un'applicazione legge un dato dallo stream (o lo scrive) si blocca finchè l'operazione non è completata
- non è richiesta una corrispondenza stretta tra letture/scritture
  - un'unica scrittura inietta 100 bytes sullo stream
  - i byte vengono letti con due read successive 'all'altro capo dello stream', la prima legge 20 bytes, la seconda 80 bytes.

CREARE GLI STREAM PER LA CONNESSIONE

- per scambiare dati su una connessione associata a un `Socket`, `Java` fornisce al client due stream principali:
  - `InputStream`: per leggere i dati che arrivano dal server.
  - `OutputStream`: per scrivere dati verso il server.
- associare stream di input o di output ad un oggetto di tipo socket:

```
public InputStream  getInputStream  () throws IOException
public OutputStream getOutputStream () throws IOException
```

- `InputStream in = socket.getInputStream();`
  - restituisce un `InputStream` (stream di input) associato al socket.
  - attraverso questo stream il client può leggere i dati che arrivano dal server
  - i dati sono letti in byte, molto a basso livello
  - spesso si usano dei wrapper che consentono di aggiungere funzionalità allo stream di base e quindi di gestire i dati a più alto livello

INTERAGIRE CON UN SERVER TRAMITE SOCKET

- client implementato in `JAVA`, server in qualsiasi altro linguaggio
  - aprire un socket `sock` sulla porta su cui è attivo il servizio
  - utilizzare gli stream per la comunicazione con il servizio
- occorre conoscere il protocollo ed il formato dei dati scambiati, che sono codificati in un formato interscambiabile
  - testo
  - `JSON`
  - `XML`
- possibile conoscere il formato dei dati scambiati interagendo con il server tramite il protocollo `telnet`

CITAZIONI IN LIBERTA': IL SERVIZIO QOTD

- servizio: `Quote of the Day` (`QOTD`)
- porta: `17 TCP` (standard per `QOTD`)
- server: `djxmmx.net`
- restituisce una citazione di testo casuale quando un client si connette: una citazione testuale per ogni connessione, poi chiude la connessione.
- funzionamento
  - client: qualsiasi programma TCP che si connette alla porta 17 del server (`telnet`, `netcat`, `Java` socket, `Python` socket ecc.).
  - il servizio non richiede nessun input dal client: appena il client si connette, il server invia una citazione scelta in modo casuale e chiude la connessione
- permette di testare client TCP o imparare a leggere dati da un server remoto senza bisogno di inviare comandi complessi.

CITAZIONI IN LIBERTA': IL SERVIZIO QOTD

```
package Citations;
import java.io.*;
import java.net.Socket;
public class CitationsClient {
    public static void main(String[] args) {
        String host = "djxmmx.net";
        int port = 17;
        Socket socket=null;
        try
           {
            socket = new Socket(host, port);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            System.out.println("Connesso a djxmmx.net! Ecco la tua citazione del giorno:\n");
            String line;
            while ((line = in.readLine()) != null) {
                System.out.println(line); }
            System.out.println("\nFine della citazione. Connessione chiusa dal server.");
           } catch (IOException e) {
                    e.printStackTrace(); }
        try
           { socket.close();
           } catch (IOException e)
                   {e.printStackTrace();}}}
```

VOGLIO UNA LISTA DI CITAZIONI !

```
package CitationsList;
import java.io.*; import java.net.Socket;

public class CitationsList {
    public static void main(String[] args) {
        String host = "djxmmx.net";
        int port = 17;   // QOTD standard TCP
        int repeat = 10; // quante citazioni vuoi ricevere
        int delay = 3000; // millisecondi tra le citazioni (3 secondi)
        Socket socket=null;
        for (int i = 1; i <= repeat; i++) {
          try
             {
              socket = new Socket(host, port);
              BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))
              System.out.println("\nCitazione #" + i + ":");
              String line;
              while ((line = in.readLine()) != null) {
                    System.out.println(line);
                }
            } catch (IOException e) { e.printStackTrace(); }

    // continua nella slide successiva
```

VOGLIO UNA LISTA DI CITAZIONI !

```
            // continua dalla slide precedente
            // Attesa prima della prossima citazione
            try {
              Thread.sleep(delay);
            } catch (InterruptedException e) {
              Thread.currentThread().interrupt(); }
        }
        System.out.println("\nFine delle citazioni.");
        try {
            socket.close();
        } catch (Exception e) {}
    }
}
```

HALF CLOSED SOCKETS

- `close( )`: chiusura del socket in entrambe le direzioni
- half closure: chiusura del socket in una sola direzione
  - `shutdownInput()`
  - `shutdownOutput()`
- in molti protocolli: il client manda una richiesta al server e poi attende la risposta

```
try ( Socket connection = new Socket("www.somesite.com", 80)){
      Writer out = new OutputStreamWriter(
                              connection.getOutputStream(), "8859_1");
      out.write("GET / HTTP 1.0\r\n\r\n");;
      out.flush();
      connection.shutdownOutput();
      // read the response
    } catch (IOException ex) { ex.printStackTrace(); }
```

- scritture successive sollevano una `IOException`

COSTRUZIONE SOCKET SENZA CONNESSIONE

- costruttore senza argomenti e connessione successiva

```
try {
    Socket socket = new Socket();
    // setta opzioni Socket, ad esempio timeout
     SocketAddress address= new InetSocketAddress ("time.nist.gov", 13);
     socket.bind(address);
    // utilizza il socket
    } catch (IOException ex)
                {System.out.println(err); }}
```

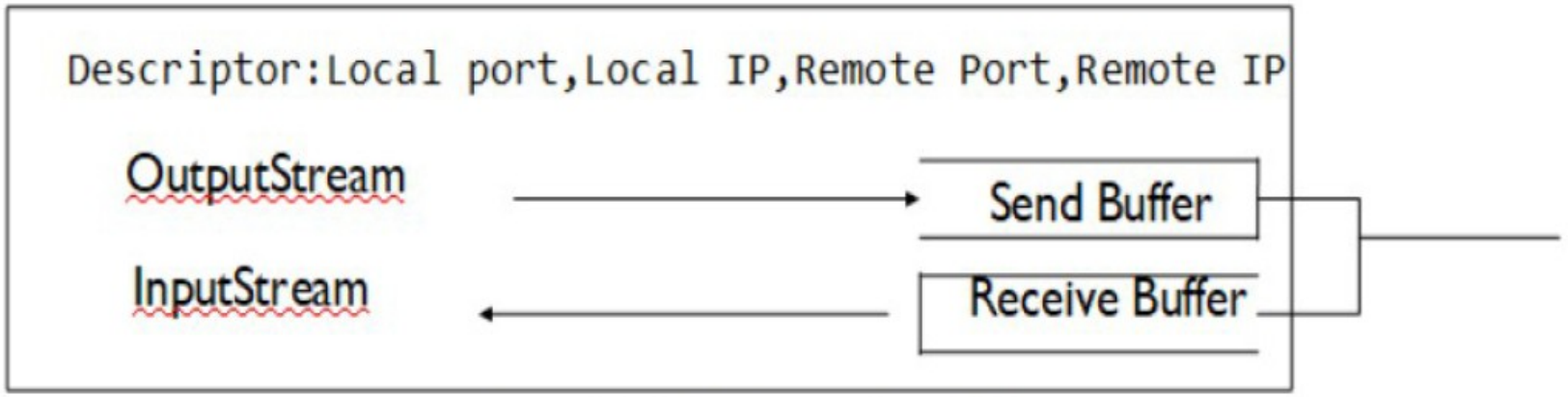
- InetSocketAddress: costruttori

```
public InetSocketAddress (InetAddress address, int port);
public InetSocketAddress(String host, int port);
public InetSocketAddress (int port);
```

REPERIRE INFORMAZIONI SU UN SOCKET

- metodi getter

```
public InetAddress getInetAddress()   } indirizzo e porta
public int getPort()                  } host remoto
public InetAddress getLocalAddress(   } indirizzo e porta
public int getLocalPort()             } host locale
```



Struttura del Socket TCP

REPERIRE INFORMAZIONI SU UN SOCKET

```
import java.net.*;
import java.io.*;
public class SocketInfo {
public static void main(String [] args)
{ for (String host: args) {
   try {
     Socket theSocket = new Socket (host, 80);
     System.out.println("Connected to "+theSocket.getInetAddress()
           +" on port"+ theSocket.getPort()+ " from port "
           + theSocket.getLocalPort() + " of"
           + theSocket.getLocalAddress());
   } catch(UnknownHostException ex) {
            System.out.println("I cannot find"+host);}
    catch(SocketException ex) {
            System.out.println("Could not connect to"+host);}
    catch(IOException ex) { System.out.println(ex);}}}}
```

```
$ java SocketInfo www.repubblica.it www.google.com
Connected to www.repubblica.it/18.66.196.94
on port 80 from port 56261 of/192.168.1.146
Connected to www.google.com/142.250.180.164
on port 80 from port 56262 of/192.168.1.146
```

RIASSUNTO

identificazione di un servizio con cui comunicare, occorre individuare:

- la rete all'interno della quale si trova l'host su cui è in esecuzione il processo
- l'host all'interno della rete
- il processo in esecuzione sull'host

- rete ed host: identificati da di Internet Protocol, mediante indirizzi IP
- processo: identificato da una porta, rappresentata da un intero da `0` a `65535`
- ogni comunicazione è quindi individuata dalla seguente 5-upla:
  - il protocollo (TCP o UDP)
  - l'indirizzo IP del computer locale (client *`sky3.cm.deakin.edu.au, 139.130.118.5`*)
  - la porta locale esempio: `5101`
  - l'indirizzo del computer remoto (server *res.cm.deakin.edu.au* `139.130.118.102`),
  - la porta remota: `5100` {tcp, `139.130.118.102, 5100, 139.130.118.5, 5101`}

ASSIGNMENT 5

Il log file di un web server contiene un insieme di linee, con il seguente formato:

```
150.108.64.57 - - [15/Feb/2001:09:40:58 -0500] "GET / HTTP 1.0" 200 2511
```

in cui:

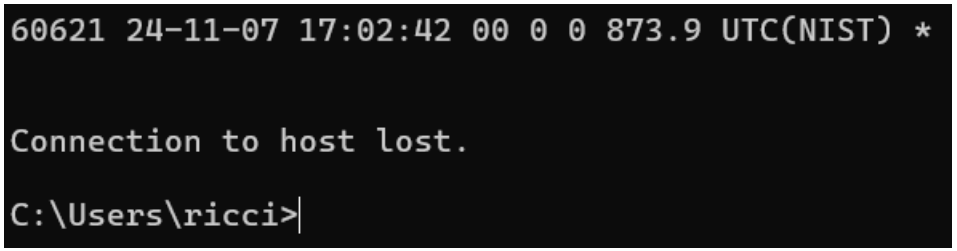
- `150.108.64.57` indica l'host remoto, in genere secondo la dotted quad form
- [data]
- "HTTP request"
- status
- bytes sent
- eventuale tipo del client "Mozilla/4.0....."

- scrivere un'applicazione `Weblog` che prende in input il nome del log file (che sarà fornito) e ne stampa ogni linea, in cui ogni indirizzo `IP` è sostituito con l'hostname
- sviluppare due versioni del programma, la prima single-threaded, la seconda invece utilizza un thread pool, in cui il task assegnato ad ogni thread riguarda la traduzione di un insieme di linee del file. Confrontare i tempi delle due versioni.

DAYTIME PROTOCOL (RFC 867)

- aprire una connessione sulla porta 13, di (NIST: National Institute of Standards and Technology) verso il servizio DayTime

```
> telnet time.nist.gov
```



Format: JJJJJ YY-MM-DD HH:MM:SS TT L H msADV UTC(NIST) OTM

- JJJJJ: Modified Julinan Date (days since Nov 17, 1858)
- TT: 00 means standard time and 50 means daylight savings time
- L: indicates whether a leap second will be added (1) or subtracted (2)
- H: health of the server (0: healthy; 1: up to 5 seconds off; …)
- msADV: how long (ms) it estimates it's going to take for the response to return
- UTC (NIST): time-zone constant string
- OTM: almost a constant (an asterisk)

DAYTIME PROTOCOL CLIENT

```
import java.io.*; import java.net.*;
public class TimeClient {
  public static void main(String[] args) {
    String hostname = args.length > 0 ? args[0] : "time.nist.gov";
    Socket socket = null;
    try {
      socket = new Socket(hostname, 13);
      socket.setSoTimeout(15000);
      InputStream in = socket.getInputStream();
      StringBuilder time = new StringBuilder();
      InputStreamReader reader = new InputStreamReader(in, "ASCII");
      for (int c = reader.read(); c != -1; c = reader.read()) {
           time.append((char) c);  }
      System.out.println(time);
        } catch (IOException ex) { System.out.println("could not connect to
                    time.nist.gov");
     } finally {
      if (socket != null) {
        try {
          socket.close();
        } catch (IOException ex) {// ignore }}}}}
```

Stampa:

```
60621 24-11-07 17:06:20 00 0 0 933.5 UTC(NIST) *
```

DAYTIME PROTOCOL CLIENT

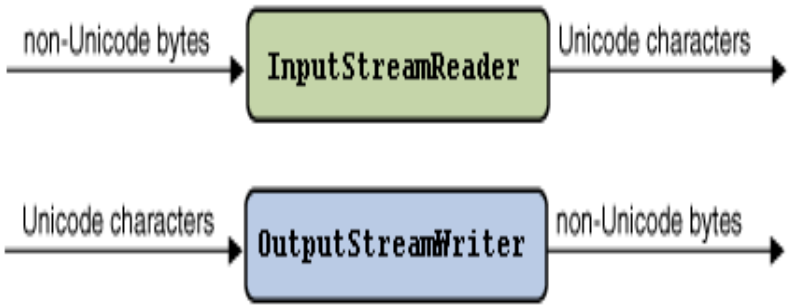
```
import java.io.*; import java.net.*;
public class TimeClient {
  public static void main(String[] args) {
    String hostname = args.length > 0 ? args[0] : "time.nist.gov";
    Socket socket = null;
    try {
      socket = new Socket(hostname, 13);
      socket.setSoTimeout(15000);
      InputStream in = socket.getInputStream();
      StringBuilder time = new StringBuilder();
      InputStreamReader reader = new InputStreamReader(in, "ASCII");
      for (int c = reader.read(); c != -1; c = reader.read()) {
           time.append((char) c);  }
      System.out.println(time);
        } catch (IOException ex) { System.out.println("could not connect to
                    time.nist.gov");
     } finally {
      if (socket != null) {
        try {
          socket.close();
        } catch (IOException ex) {// ignore }}}}}
```

NOTA: `setSoTimeout(<ms>)`: setta un timeout sul socket

- previene attese indeterminate di risposte dal server
- solleva `SocketTimeoutException` (è una `IOException`)

DAYTIME: LEGGERE CARATTERI

- utilizza `InputStreamReader`
- istanziato su un `InputStream`
- parametro
  - codifica dei caratteri presenti sullo stream di byte (`ASCII`, `UTF-8`, `UTF-16`,...)
- traduce caratteri esterni nella codifica interna `Unicode`




```
                    …...
InputStream in =
          socket.getInputStream();
StringBuilder time = new
          StringBuilder();
InputStreamReader reader = new
    InputStreamReader(in, "ASCII");
for (int c=reader.read();c != -1;
        {
     time.append((char) c);  }
                    …...
```