

Reti e Laboratorio III

Modulo Laboratorio III

AA. 2022-2023

docente: Laura Ricci

laura.ricci@unipi.it

Lezione 5

InetAddress

Stream Sockets for clients

13/10/2022

NETWORK APPLICATIONS

alcune “killer network applications”

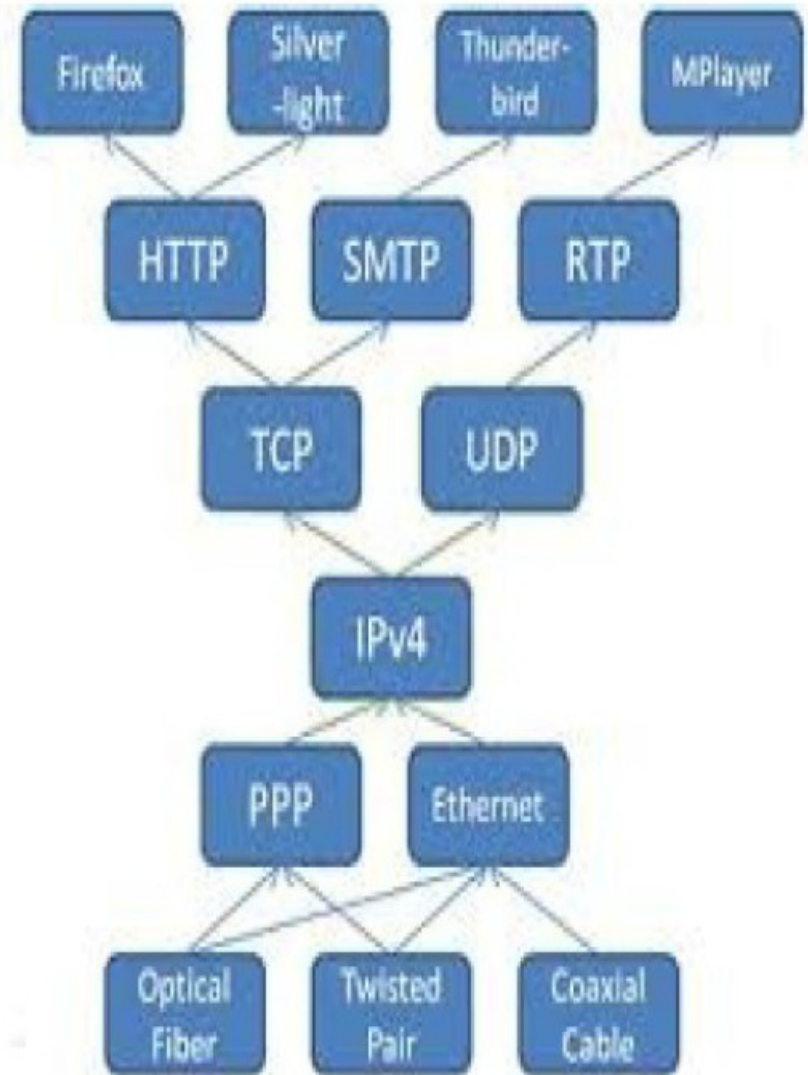
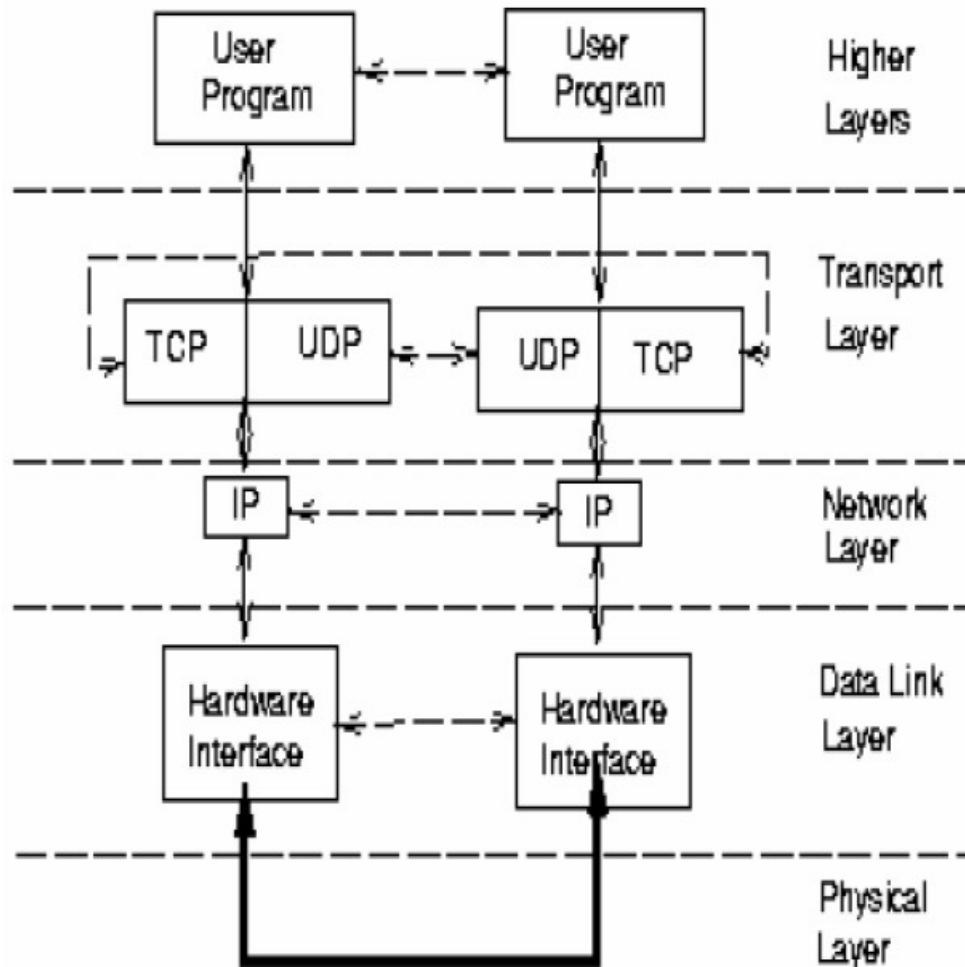
- web browser
- SSH
- email
- social networks
- teleconferences (Skype, Zoom, GoToMeeting, Meet, Teams,...)
- program development environments: GIT
- collaborative work: Overleaf
- multiplayer games: War of Warcraft
- P2P File sharing: Bittorrent
- blockchain: cryptocurrencies (Bitcoin), supply chain,...
- metaverse, e molte altre....

scopo del corso è mettervi in grado di sviluppare una semplice applicazione di rete.

NETWORK APPLICATIONS

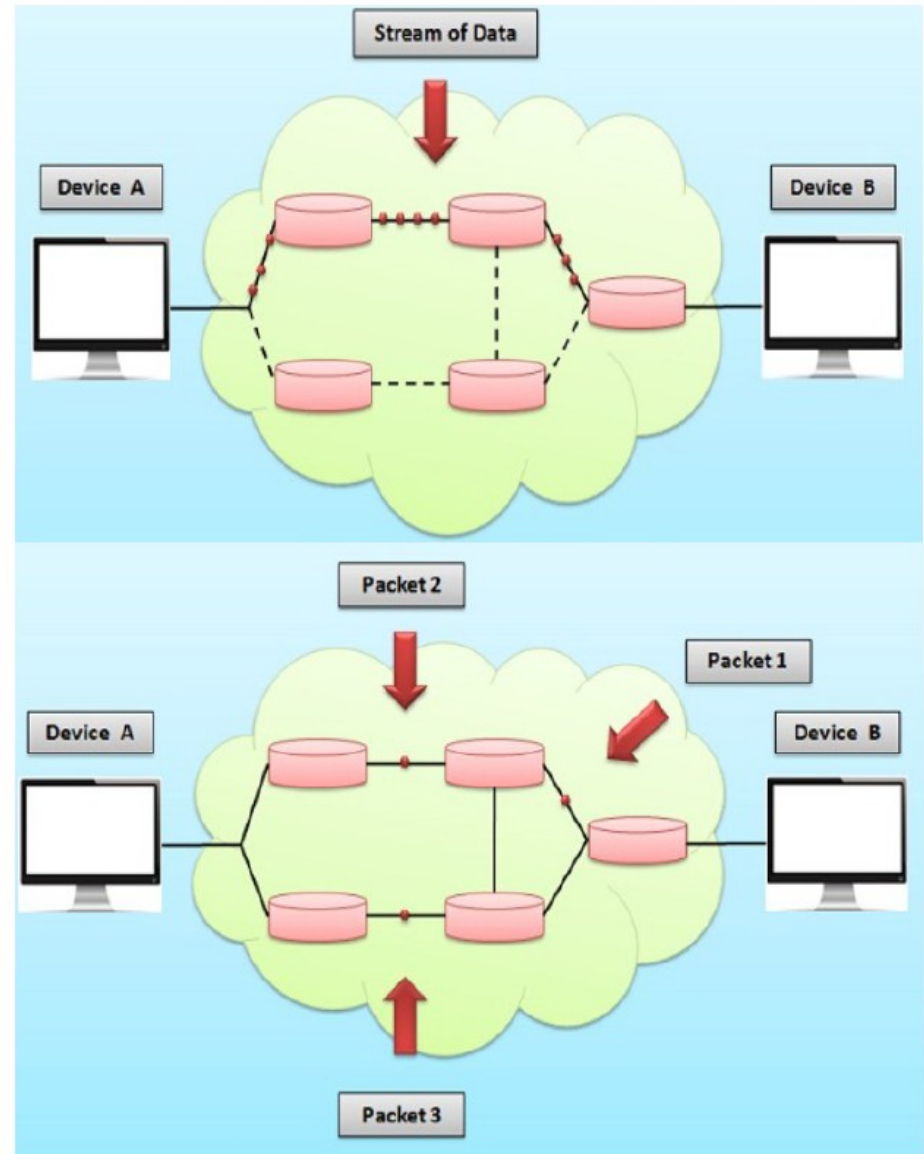
- due o più **processi** (non thread!) in esecuzione su **hosts diversi**, distribuiti geograficamente sulla rete, **comunicano** e **cooperano** per realizzare una funzionalità globale:
- **cooperazione**: scambio informazioni utile per perseguire l'obiettivo globale, quindi implica comunicazione
- **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, Transmission 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
 - datagramsocket



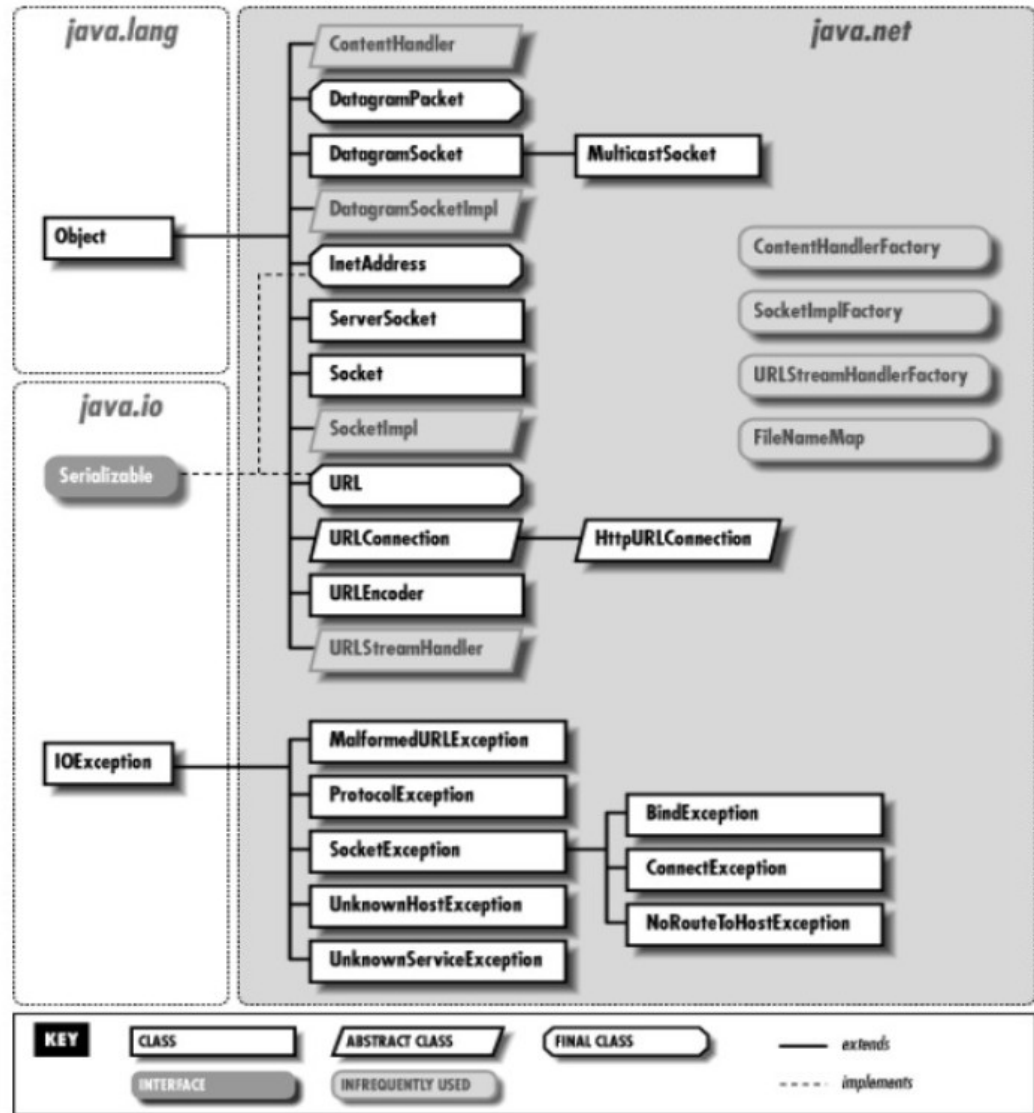
JAVA.NET: NETWORKING IN JAVA

connection-oriented

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

connectionless

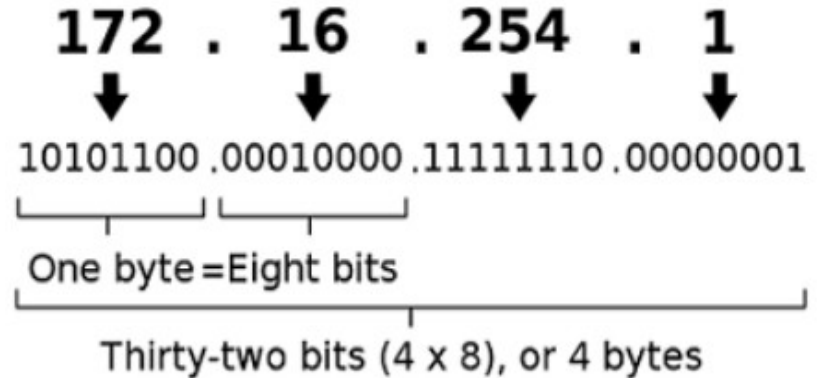
- simmetrici: sia per il client che per il server
 - datagramSocket
 - datagramPacket



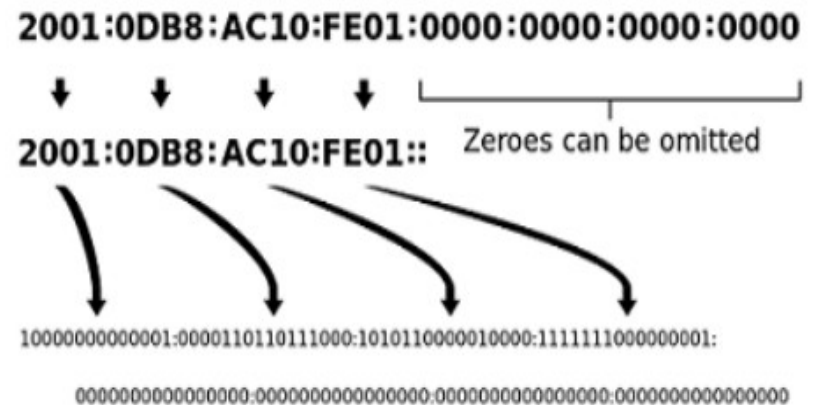
IP (INTERNET PROTOCOL) ADDRESS

- **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

An IPv4 address (dotted-decimal notation)



An IPv6 address (in hexadecimal)



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 e IPv6
- usata per incapsulare in un unico oggetto di tipo `InetAddress` sia
 - 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
```

- `getLocalHost()` restituisce l' `InetAddress` del local host

```
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 address"); }}
```



INETADDRESS: CACHING

- i metodi descritti **effettuano caching** dei nomi/indirizzi risolti
 - l'accesso al DNS è una 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 a causa di 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 UnknownHostException`
 - `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)
- 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`)

`boolean 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 InetAddressInstance {

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

        InetAddress ia1 = InetAddress.getByName("www.google.com");

        byte [] address = ia1.getAddress();

        System.out.println(Arrays.toString(address));

        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());

        System.out.println(InetAddress.getByAddress(new byte[]{127,0,0,1}).isLoopbackAddress());

        System.out.println(InetAddress.getByAddress(new byte[] {(byte)225,(byte)255,(byte)255,
                                                                (byte)255}).isMulticastAddress());}}
```



```
$ Java InetAddressInstance
[-114, -6, -76, -124]
142.250.180.132
www.google.com
true
false
false
true
true
```


UN PROGRAMMA UTILE: SPAM CHECKER

- diversi servizi monitorano gli spammers: [real-time black-hole lists](#) (RTBLs)
 - ad esempio: sb1.spamhaus.org
 - mantengono una lista di indirizzi IP che risultano, probabilmente, degli spammers
- per identificare se un indirizzo IP corrisponde ad uno spammer:
 - inversione dei bytes dell'indirizzo IP
 - concatena il risultato a sb1.spamhaus.org
 - esegui un DNS look-up
 - la query ha successo se e solo se l'indirizzo IP corrisponde ad uno spammer
- SpamCheck richiede a sb1.spamhaus.org se un indirizzo IPv4 è uno spammer noto
 - es una query DNS su `17.34.87.207.sb1.spamhaus.org` ha successo se l'indirizzo è uno spammer

UN PROGRAMMA UTILE: SPAM CHECKER

```
import java.net.*;

public class SpamCheck {
    public static final String BLACKHOLE = "sbl.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."); }}}
    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 23.45.65.88 141.250.89.99 127.0.0.2

23.45.65.88 appears legitimate.
141.250.89.99 appears legitimate.
127.0.0.2 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:

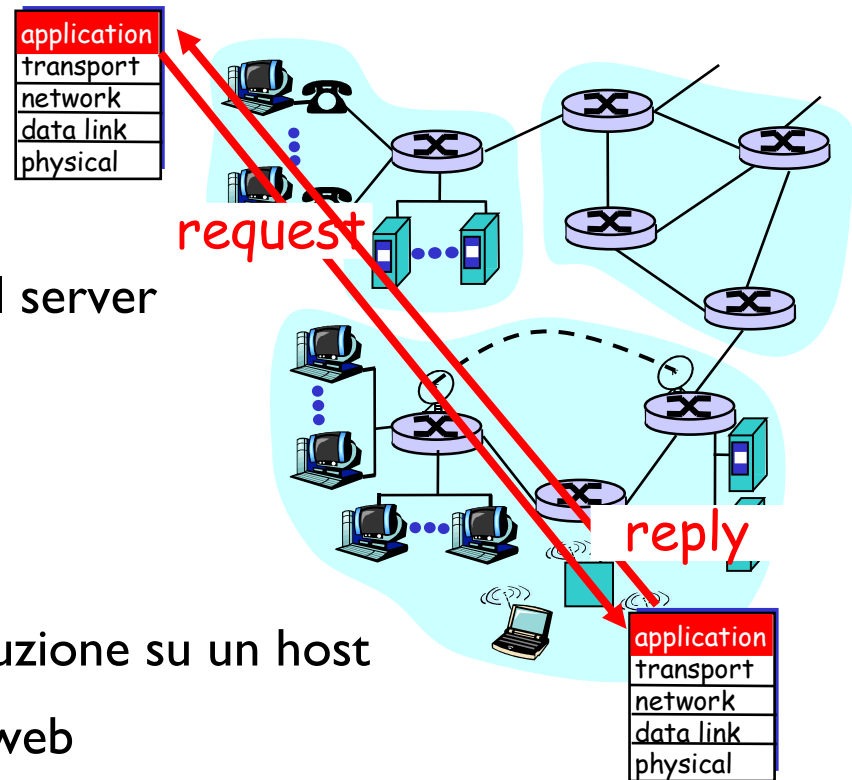
- un software che sfrutta servizi forniti dal server

web client browser

e-mail client mail-reader

server:

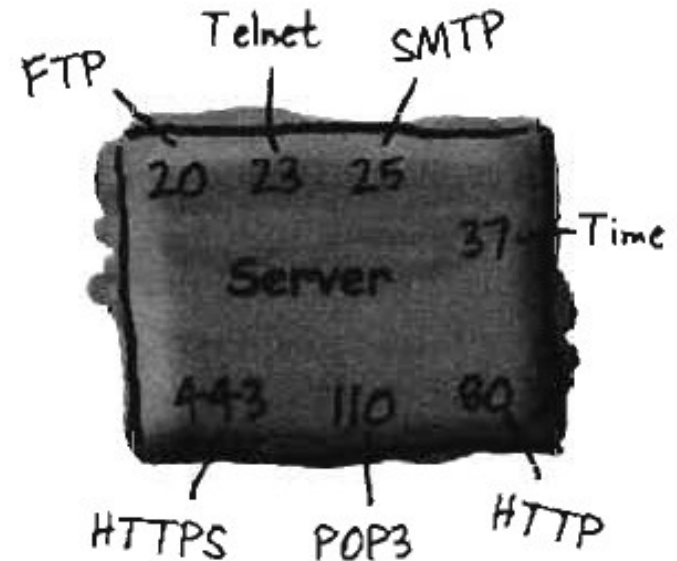
- istanza di un particolare servizio in esecuzione su un host
- ad esempio: server Web invia la pagina web richiesta, mail server consegna la posta al client



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.

Well-known TCP port numbers
for common server applications



A server can have up to 65536
different server apps running,
one per port

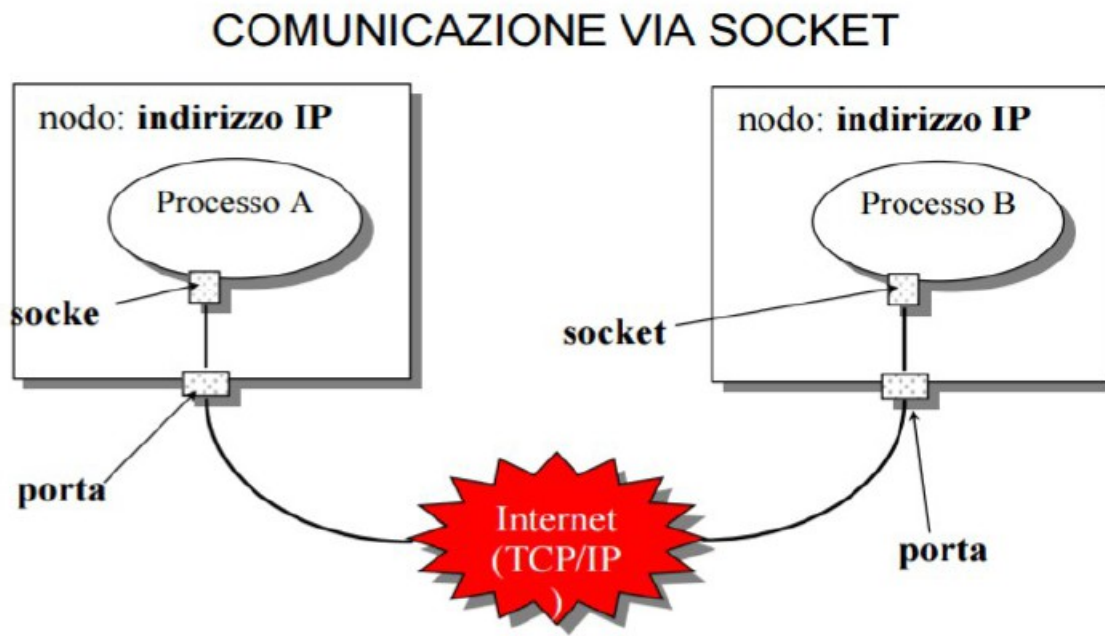
CONNETTERSI AD UN SERVIZIO

- 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**



COME IL CLIENT ACCEDE AD UN SERVIZIO

- 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,

MODELLARE UNA CONNESSIONE MEDIANTE STREAM

- una volta stabilita una connessione tra client e server devono scambiarsi dei dati. La connessione è modellata **come uno stream**.
- associare uno stream di input o di output ad un socket:

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

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

- invio di dati: client/server leggono/scrivono dallo/sullo stream
 - un byte/una sequenza di bytes
 - dati strutturati/oggetti. In questo caso è necessario associare dei filtri agli stream
- ogni valore scritto sullo stream di output associato al socket viene copiato nel *Send Buffer* del livello TCP
- ogni valore letto dallo stream viene prelevato dal *Receive Buffer* del livello TCP

INTERAGIRE CON IL 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

DAYTIME PROTOCOL (RFC 867)

- aprire una connessione sulla porta 13, verso il servizio `time.nist.gov` (NIST: National Institute of Standards and Technology)

```
$ telnet time.nist.gov 13
Trying 129.6.15.28...
Connected to time.nist.gov.
Escape character is '^]'.
```

```
56375 13-03-24 13:37:50 50 0 0 888.8 UTC(NIST) *
Connection closed by foreign host.
```

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

- JJJJJ: Modified Julian 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)

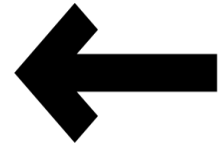
DAYTIME PROTOCOL CLIENT

```
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 }}}}}
            setSoTimeout(<ms>): setta un timeout sul socket
            • previene attese indeterminate di risposte dal server
            • solleva SocketTimeoutException (è una
              IOException)
```

DAYTIME CON TRY WITH RESOURCES

```
try { socket = new Socket(hostname, 13);  
    //read from the socket  
} catch (IOException ex)  
    {System.out.println("could not connect to time.nist.gov");}
```

```
finally {  
    if (socket != null) {  
        try {  
            socket.close();  
        } catch (IOException ex) { // ignore }}}}
```

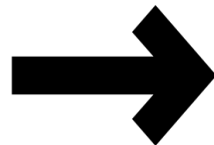


Java 6- : rilascio esplicito
delle risorse

Java 7+: autochiusura tramite

try with resources

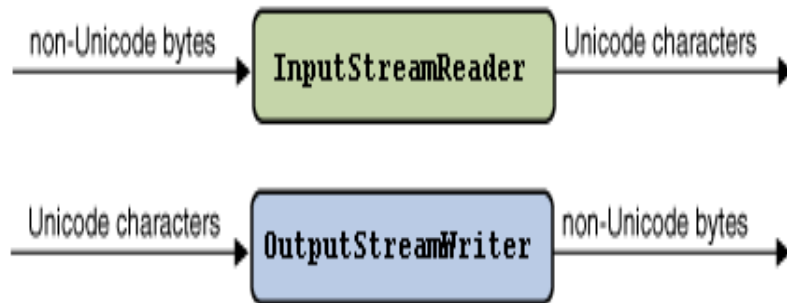
clausola finally non necessaria



```
try (Socket socket = new Socket("time.nist.gov", 13))  
    { //read from the socket  
    } catch (IOException ex)  
    { System.out.println("could not  
        connect to time.nist.gov");}
```

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);  
    }  
    .....  
}
```

TRY WITH RESOURCES

- introdotto in JAVA 7, aggiornato in JAVA 9
- chiusura sistematica ed automatica delle risorse usate da un programma
- un blocco try con uno o più argomenti tra parentesi.
 - argomenti = risorse che devono essere chiuse quando il try block termina
 - le variabili che rappresentano le risorse non devono essere riutilizzate
- suppressed exceptions:
 - quando si verificano delle eccezioni sia nel blocco try-with-resources sia durante la chiusura, la JVM sopprime l'eccezione generata nella chiusura automatica.
- generalizzazione: implementazione della AutoCloseable interface

TRY WITH RESOURCES

- una certa risorsa è chiusa “automaticamente”, dopo che è stata utilizzata
 - risorsa: file, stream, reader o socket
 - tecnicamente ogni oggetto che implementi l'interfaccia AutoClosable

```
try (FileWriter w = new FileWriter("file.txt")) {  
    w.write("Hello World"); }  
  
// w.close() is called automatically
```

- in questo esempio, `w.close()` viene chiamata indipendentemente dal fatto che la write sollevi o meno una eccezione
- concettualmente simile ad aggiungere `w.close()` in un blocco `finally`
- possibile usare più risorse in un blocco `try with resources`, vengono chiuse in senso inverso rispetto all'ordine con cui sono state dichiarate

TRY WITH RESOURCES: ECCEZIONI

- nel seguente esempio

```
try (FileWriter w = new FileWriter("file.txt")) {  
    w.write("Hello World"); }  
    // w.close() is called automatically
```

- una eccezione può essere sollevata nei seguenti statement
 - `new FileWriter("file.txt")`
 - `w.write("Hello World")`
 - implicitamente da `w.close()`
- eccezione sollevata nel costruttore: nessun oggetto da chiudere, si propaga la eccezione senza eseguire la `write()`



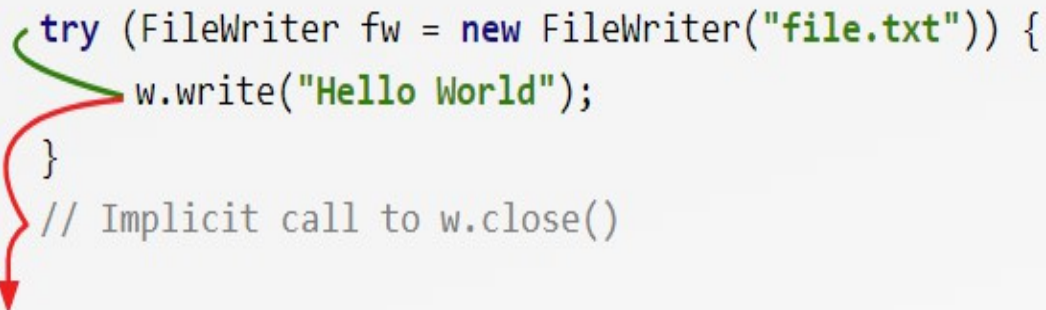
```
try (FileWriter w = new FileWriter("file.txt")) {  
    w.write("Hello World");  
}  
    // no call to w.close()
```

TRY WITH RESOURCES: ECCEZIONI

- nel seguente esempio

```
try (FileWriter w = new FileWriter("file.txt")) {  
    w.write("Hello World"); }  
    // w.close() is called automatically
```

- eccezione sollevata nella write() : viene invocato w.close(), poi si propaga l'eccezione



```
try (FileWriter fw = new FileWriter("file.txt")) {  
    w.write("Hello World");  
}  
// Implicit call to w.close()
```

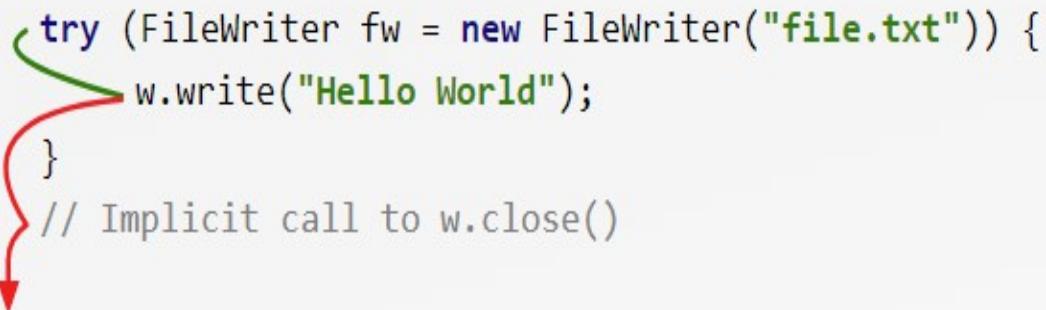
The diagram illustrates the execution flow of a try-with-resources block. A green arrow points from the opening curly brace of the try block to the `w.write("Hello World");` statement. A red arrow originates from the closing curly brace, indicating the point where an exception is thrown. This red arrow points down to the line `// Implicit call to w.close()`, showing that resource cleanup occurs before the exception is propagated out of the try block.

TRY WITH RESOURCES: ECCEZIONI

- nel seguente esempio

```
try (FileWriter w = new FileWriter("file.txt")) {  
    w.write("Hello World"); }  
    // w.close() is called automatically
```

- eccezione sollevata nella write() : viene invocato w.close(), poi si propaga l'eccezione



The diagram illustrates the flow of an exception in a try-with-resources block. A green arrow points from the `try` statement to the `w.write("Hello World");` line, indicating that an exception is thrown at this point. A red arrow then points from the `w.write` line down to the `// Implicit call to w.close()` line, showing that the resource is closed before the exception is propagated out of the try block.

```
try (FileWriter fw = new FileWriter("file.txt")) {  
    w.write("Hello World");  
}  
    // Implicit call to w.close()
```

TRY WITH RESOURCES: ECCEZIONI

- nel seguente esempio

```
try (FileWriter w = new FileWriter("file.txt")) {  
    w.write("Hello World"); }  
    // w.close() is called automatically
```

- eccezione sollevata nella chiamata implicita alla close() : viene propagata la eccezione



```
try (FileWriter fw = new FileWriter("file.txt")) {  
    w.write("Hello World");  
}  
// Implicit call to w.close()
```

The diagram illustrates the flow of an exception. A green bracket on the left groups the try block and the closing brace. A green arrow points from the closing brace down to the line indicating the implicit call to `w.close()`. A red arrow then points from this line down to the bottom of the slide, representing the propagation of the exception.

TRY WITH RESOURCES: SUPPRESSED EXCEPTIONS

- nel seguente esempio

```
try (FileWriter w = new FileWriter("file.txt")) {  
    w.write("Hello World"); }  
    // w.close() is called automatically
```

- cosa accade se la `w.write()` solleva un'eccezione ed anche la chiamata implicita alla `w.close()` la solleva?
- la prima eccezione “vince” sulla seconda e la seconda viene soppressa



TRY WITH RESOURCES: SUPPRESSED EXCEPTIONS

```
import java.io.*;

public class trywithresources
{
    public static void main (String args[]) throws IOException {
        try(FileInputStream input = new FileInputStream(new File("immagine.jpg"));
            BufferedInputStream bufferedInput = new BufferedInputStream(input))
        {
            int data = bufferedInput.read();
            while(data != -1){
                System.out.print((char) data);
                data = bufferedInput.read();
            }
        }
    }
}
```

- risolve il problema delle “suppressed exceptions”
 - eccezioni possono essere sollevate nel blocco try, oppure nel blocco finally,
 - un'eccezione rilevata nella finally sopprimerebbe l'eccezione rilevata nel blocco try
- con il try with resources viene propagata l'eccezione rilevata nel blocco try

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 = new InetSocketAddress ("time.nist.gov", 13);  
    socket.bind(conect(address));  
    // utilizza il socket  
} catch (IOException ex) {System.out.println(err); }}
```

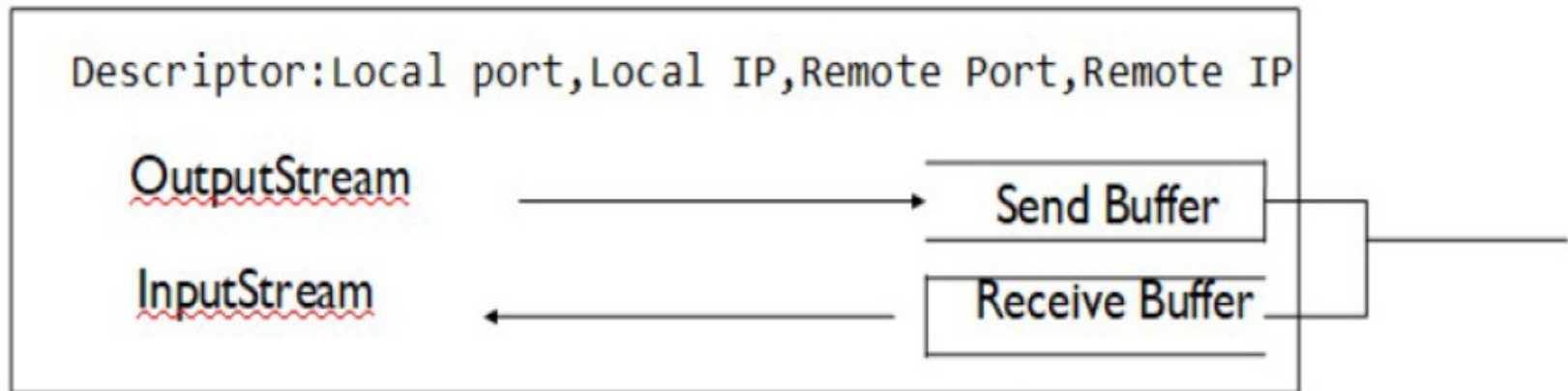
- scritture successive sollevano una IOException
- InetSocketAddress: costruttori

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

REPERIRE INFORMAZIONI SU UN SOCKET

- metodi getter

<code>public InetAddress getInetAddress()</code>	}	indirizzo e porta
<code>public int getPort()</code>		host remoto
<code>public InetAddress getLocalAddress()</code>	}	indirizzo e porta
<code>public int getLocalPort()</code>		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.