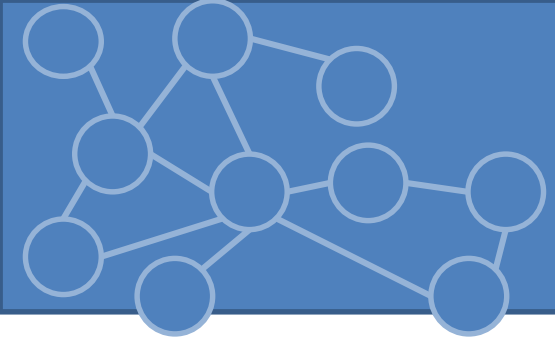
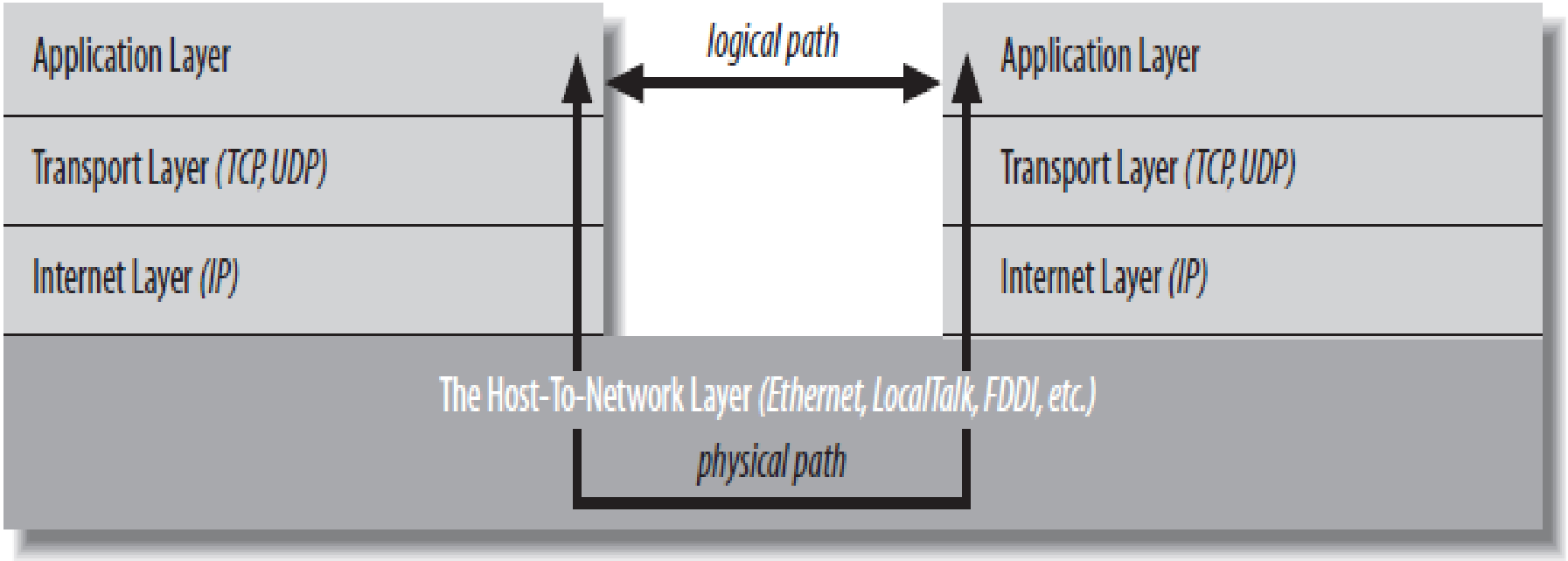


Laboratorio Reti di Calcolatori

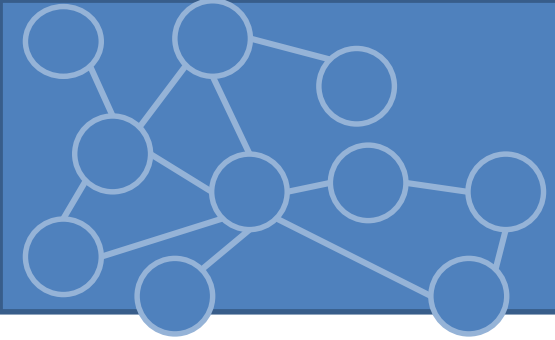
Laurea Triennale in Comunicazione Digitale

Anno Accademico 2013/2014

- Diversi modelli di livelli di rete, Java si focalizza su modello TCP(UDP)/IP



- Viene creato un percorso logico tra i livelli applicativi dei due host che comunicano

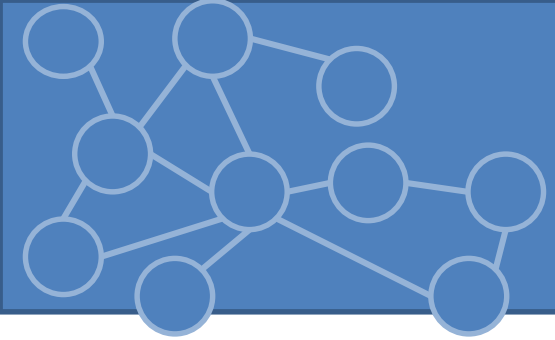


IP, TCP e UDP

- Vantaggi di IP
  - Robustezza: più cammini tra due nodi qualsiasi e instradamento su cammini validi
  - Aperto e indipendente dalla piattaforma

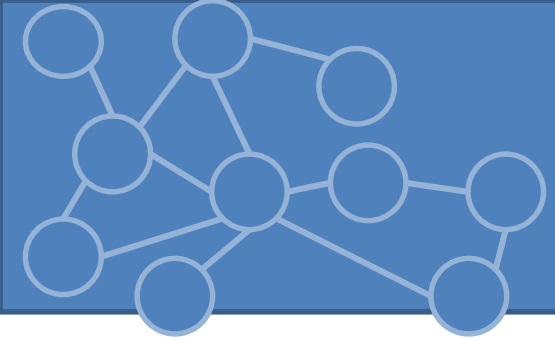
$\Rightarrow$ Pacchetti che costituiscono uno stesso stream possono 'prendere' strade diverse + ordine di partenza può non essere rispettato all'arrivo.

- TCP
  - Ritrasmissione in caso di pacchetti persi o corrotti
  - Stessa sequenza di trasmissione e ricezione
- Se ordine non importante o perdita di pacchetti non corrompe lo stream si utilizza UDP
  - Video e audio streams + correzione dell'errore a livello applicazione



Indirizzi IP

- Come sviluppatore Java si richiede solo conoscenza degli indirizzi
  - In IPv4 indirizzo è un numero di 4 byte nel formato dotted quad. Ogni byte è unsigned
  - Header del pacchetto include indirizzo del destinatario e del mittente
  - In IPv6 indirizzo di 16 byte. Scritti in 8 blocchi di 4 cifre esadecimali divisi da :
  - Formato misto
- DNS: mapping tra nomi human-friendly e indirizzi IP. Java ammette sia nomi sia indirizzi IP che vengono trattati dalla classe *java.net.InetAddress*
- Per molti client l'indirizzo è dinamico (DHCP) => controllo indirizzo IP corrente ogni volta che ne ho bisogno (evitare caching)



InetAddress

java.net.InetAddress è una rappresentazione di un indirizzo IP.

- Socket, ServerSocket, URL, DatagramSocket, DatagramPacket

Include hostname e IP.

Creazione

Nessun costruttore pubblico ma

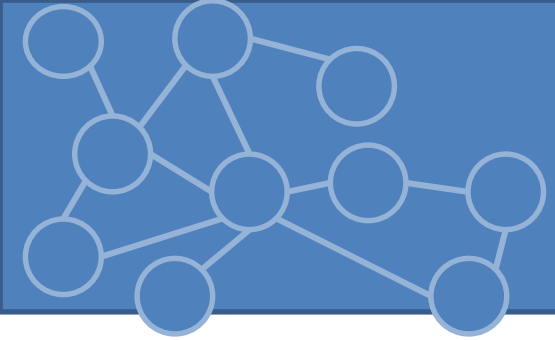
public static InetAddress ***getByName***(String hostName) throws UnknownHostException

public static InetAddress[] ***getAllByName***(String hostName) throws UnknownHostException

public static InetAddress ***getLocalHost***( ) throws UnknownHostException

Possono connettersi al DNS locale per reperire informazioni necessarie

- DNS lookup costoso: InetAddress mette risultati in **cache** sia positivi sia negativi
  - **networkaddress.cache.ttl**, **networkaddress.cache.negative.ttl** specificano il numero di secondi di permanenza di una entry nella cache (-1 = infinito)



public static InetAddress **getByAddress**(byte[] address) throws UnknownHostException

public static InetAddress **getByAddress**(String hostName, byte[] address) throws UnknownHostException

Non controllano indirizzo usando DNS

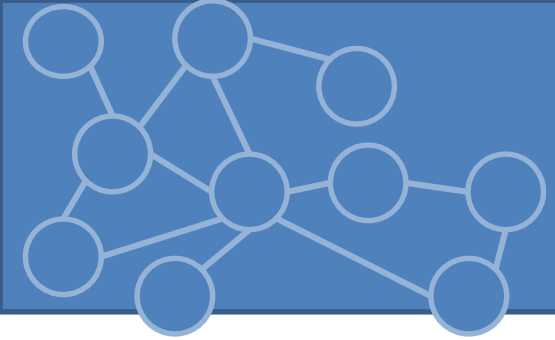
- Nessuna garanzia che host esista o che il mapping IP/hostname sia attendibile.

getByName(String hostname)

Hostname è il nome dell'host che devo risolvere usando il DNS

Solleva una UnknownHostException se l'host non viene trovato

```
try {
InetAddress address = InetAddress.getByName("www.google.it");
System.out.println(address);
}
catch (UnknownHostException ex) {
System.out.println("www.google.it non trovato");
}
```



Posso passare una stringa contenente la forma puntata dell'indirizzo IP

- Crea un oggetto *InetAddress* per l'IP richiesto senza controllare nel DNS
- DNS lookup viene richiesta quando hostname è richiesto da *getHostName*() o dal metodo *toString()*

getAllByName(String hostname)

Alcuni hostname sono associati a più di un indirizzo IP

Restituisce un array che contiene gli indirizzi IP associati all'hostname

Solleva un *UnknownHostException*

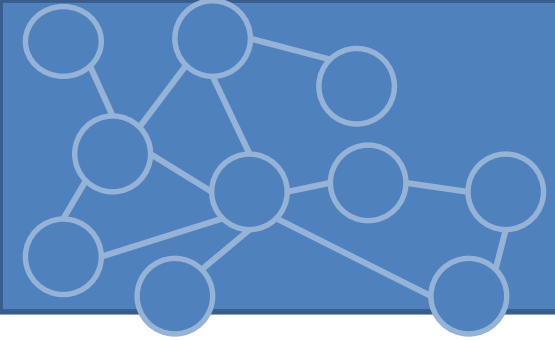
getByAddress(byte[] address)

getByAddress(String hostname, byte[] address)

Creo un oggetto *InetAddress* ma non viene effettuato un DNS lookup

getLocalhost()

Restituisce l'*InetAddress* della macchina su cui è stato invocato



getHostName()

Restituisce una stringa che contiene il nome dell'host con indirizzo IP rappresentato dall'oggetto *InetAddress*

- Utile quando ho solo indirizzo IP

getHostAddress()

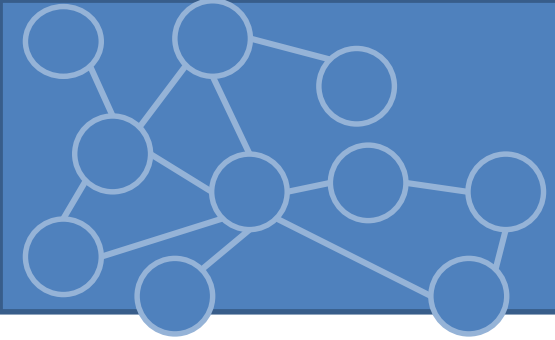
Retituisce una stringa contenente il formato puntato dell'indirizzo IP

getAddress()

Restituisce indirizzo IP come un **array di byte**. Il byte più significativo è il primo byte nell'array

- int unsignedByte = signedByte < 0 ? signedByte + 256 : signedByte;
- Per determinare il tipo di IP (es: IPv4 o IPv6)

Codice: IPClassi.java



isReachable(NetworkInterface, ttl, timeout)

- Permette di testare se un nodo particolare è raggiungibile da host corrente
- Si connette alla porta echo su host remoto, se host remoto risponde entro il timeout (in ms) metodo restituisce true
- IOException se errori nella rete o nella connessione

equals(Object)

- Uguali se corrispondono a stesso IP (codice: Esempio2.java)

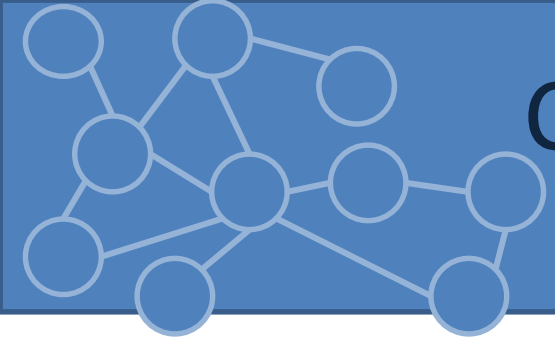
toString()

- hostname/xxx.xxx.xxx.xxx

Porte

- Multitasking è la norma, ma un solo indirizzo IP
  - uso mail contemporaneamente a FTP e traffico web
- Ogni host con un indirizzo IP ha 65535 porte che possono essere allocate per diversi servizi.
  - HTTP su 80. Web server 'ascolta' su porta 80 in attesa di possibili connessioni.
- Il ricevente controlla la porta a cui pacchetto è indirizzato e invia il pacchetto al programma che ascolta su quella porta.
- Tra 1 e 1023 sono riservate per servizi conosciuti

Porta	Servizio	Porta	Servizio	Porta	Servizio
20/21	FTP	23	Telnet	110	POP3
22	SSH	25	SMTP	143	IMAP



Concetti base per il Web
URI

- **URI (Uniform Resource Identifier)**: stringa di caratteri che identifica una risorsa usando una particolare sintassi.
  - Risorsa: file su server, mail, news message, libro, persona, host.
- URI **assoluto**:

  schema:<parte dipendente dallo schema>

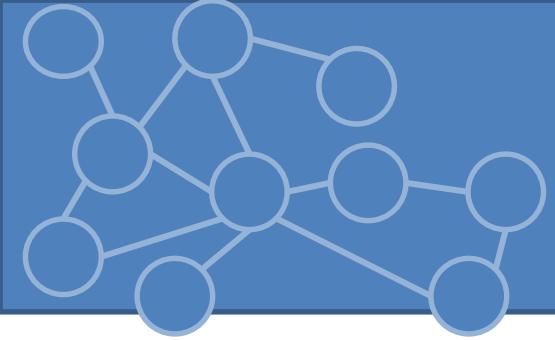
  – schema: data, file, ftp, http, mailto, news, telnet, urn.
- La sintassi della seconda parte dipende dallo schema, molti hanno una struttura gerarchica.

  //autorità/percorso?query

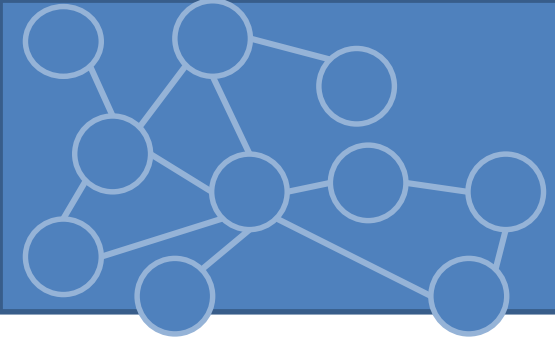
  Autorità: responsabile della risoluzione del resto dell'URI

  Se autorità è un host Internet posso specificare username e porte.

  Percorso: stringa che autorità usa per determinare quale risorsa è identificata. Può essere gerarchico.



- Parte specifica dello schema composta da caratteri ASCII, caratteri NON ASCII vengono sostituiti da % e dal codice esadecimale del carattere. Un URL trasformato viene detto x-www-form-urlencoded.
  - Caratteri non ASCII codificati usando UTF-8 e facendo escaping di ogni byte
  - @ e / devono essere codificati usando escaping se assumono un ruolo diverso da quello specificato dallo schema

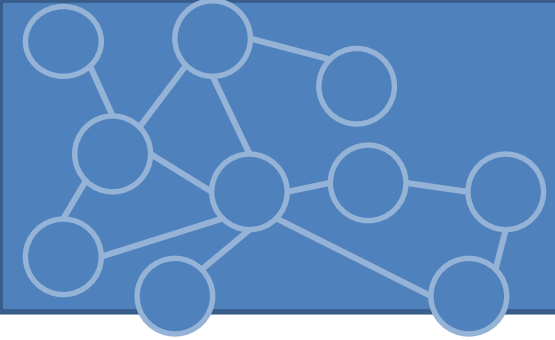


URL

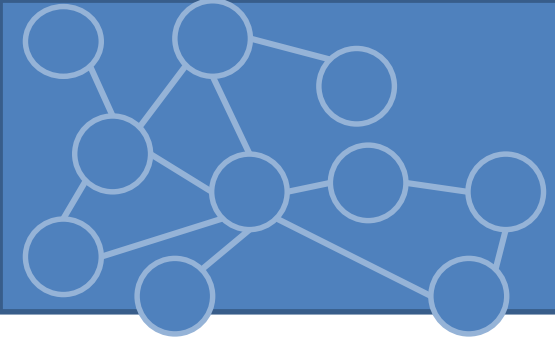
- URL identifica la location di una risorsa in Internet specificando:
  - Protocollo di accesso al server, nome del server e location del file sul server.

protocol://username@hostname:porta/path/filename?query#fragment

- **protocol**: altro modo per indicare schema.
- **hostname**: nome del server che rende disponibile la risorsa. Sia name sia indirizzo IP.
- **username**: opzionale
- **porta**: opzionale
- **path**: directory particolare sul server specificato. È relativo alla document root del server (server rende visibile solo la directory dei contenuti).
- **filename**: file nella directory. Se omesso è il server che decide quale inviare (es. index.html)
- **query**: parametri aggiuntivi per il server.
- **fragment**: si riferisce ad una parte particolare di una risorsa remota



- Molte informazioni nell'URL probabilmente le stesse per altri URL nello stesso documento.
- URL può ereditare il protocollo, l'hostname e il percorso dal documento padre = URL relativi. Ogni parte mancante è ereditata dal documento che contiene URL
  - Se link inizia con / è relativo alla document root
- Vantaggi:
  1. Sposto un albero di documenti senza modificare il codice
  2. Risparmio caratteri e tempo ☺



URL

java.net.URL è astrazione di un Uniform Resource Locator.

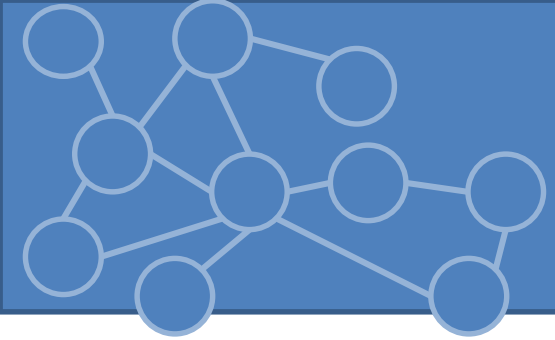
È final, non può essere estesa !

- URL è oggetto con campi che includono lo schema, l'hostname, la porta, il percorso, la query e l'identificatore del frammento.
- Una volta creato non può essere modificato.

Creazione URL

- 6 costruttori. Scelta dipende dalle informazioni che ho a disposizione alla costruzione.

  Sollevano *MalformedURLException* se si crea un URL da un protocollo non supportato o *MalformedURLException* se c'è un errore sintattico (codice: UrlEsempi.java)



URL(String url)

```
try {
    URL u = new URL("http://www.google.it");
}
catch (MalformedURLException ex) {
    System.out.println("");
}
```

URL(String protocol, String hostname, String file)

- La porta viene settata a -1 (porta di defaul del protocollo)
- File deve iniziare con un / e include percorso e nome del file e frammento (opzionale)
- Altro costruttore permette di settare la porta

URL(URL base, URL relative)

- Costruisce un URL assoluto partendo da una URL base e da un URL relativo
- Eventuale filename viene rimosso quando attacco il relativo alla base
- Utile per iterare su file nella stessa directory

Dividere un URL

getProtocol

Stringa che contiene lo schema dell'URL

getHost()

Stringa con l'hostname dell'URL

getPort()

Se non specifico porta restituisce un -1 corrispondente a quella di default del protocollo

getDeafulPort()

Restituisce la porta di default del protocollo quando non specificata nell'URL. -1 se non c'è porta di default

getFile()

Il percorso dal primo / fino all'ultimo carattere prima di #

getPath()

Non include la query

getRef()

Restituisce il frammento

getQuery , getUserInfo, getAuthority

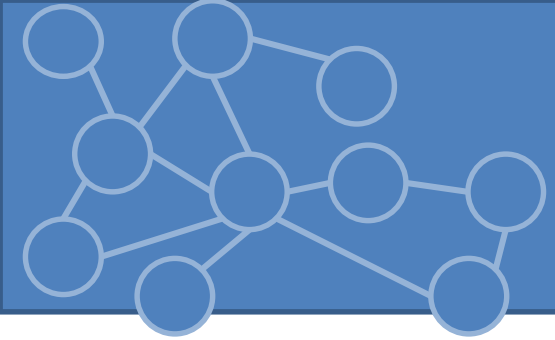
Acquisire dati da un URL

InputStream openStream()

- Si connette alla risorsa referenziata da URL e restituisce un *InputStream* da cui i dati possono essere letti
  - Contenuto grezzo del file, non interpretato

```
URL u = new URL("http://www.google.it");
InputStream is = u.openStream();
int c;
while((c = is.read())!=-1) System.out.write(c);
```

  - Come codificare il file non è dato dal metodo ma bisogna scorrere il file in ricerca di un header.



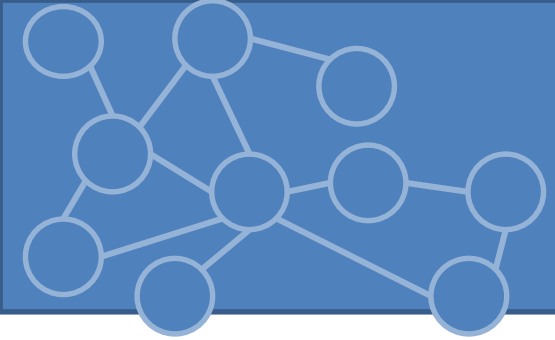
URLConnection openConnection()

- Apre una socket verso l'URL specificato. ***URLConnection*** rappresenta una connessione aperta verso una risorsa di rete
  - *IOException* se chiamata fallisce
- *URLConnection* ha il pieno accesso a tutto ciò che viene inviato dal server. Posso accedere a tutti i metadati del protocollo.
- *URLConnection* permette di leggere/scrivere verso URL

Object getContent()

- Acquisisce i dati riferiti da URL e tenta di trasformarli in un qualche oggetto
- Agisce in base al campo Content-type nel MIME header dei dati che riceve dal server.
- Overload con array[] di Class cerca di restituire il contenuto secondo la preferenza delle classi imposta dall'array di Class

Codice:UrlStream.java



URLEncoder

Caratteri usabili negli URL sono:

- A-Z, a-z, 0-9, -_.!*'(),
- /&?@#;$+=% possono essere usati ma per uno scopo preciso. Se sono nel filename devono essere codificati

- Ogni carattere è convertito in byte e ogni byte è scritto come % e due cifre esadecimali
  - Spazio codificato con + e + codificato come %2B
  - /&?@#$= vengono codificati quando sono usati come parte del nome

URLEncoder codifica una stringa secondo le precedenti specifiche

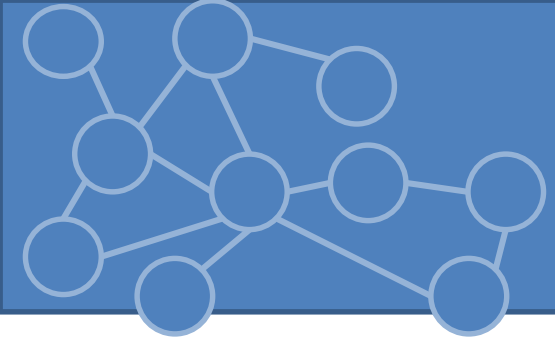
String encode(String s, String encoding)

- Meglio usare UTF-8 come codifica

- Uso più comune è preparare le stringhe di query per comunicare parametri al server usando la GET

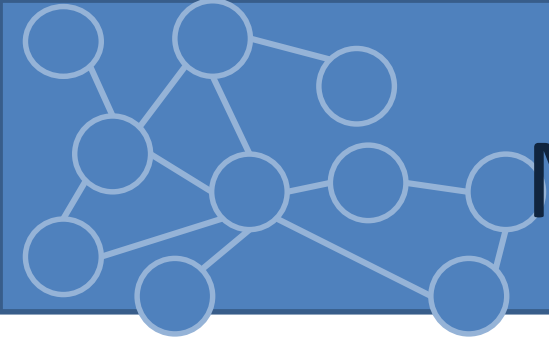
URLEncoder non distingue se = e & usati all'interno della query

Codice:EncodingUrl.java



MIME

- MIME: standard aperto per inviare dati con formati diversi nelle email attraverso Internet.
- Standard per descrivere il contenuto di un file e gestire il dato che contiene.
- Supporta più di 100 tipi di contenuto classificati in tipi e sottotipi. Il sottotipi specifica nel dettaglio il tipo di dato
- Web servers usano MIME per identificare il tipo di dato che stanno inviando, mentre clients lo usano per indicare quali tipi possono accettare.
- Posso specificare sottotipi non standard usando il prefisso x.



Modello Client/Server

- Maggior parte programmazione di rete si basa su paradigma client/server. Dati ed elaborazione pesante su server (hardware performante), interfaccia utente e elaborazione più leggera su client (hardware meno performante e più economico)
- Server mette a disposizione un servizio e si mette in attesa di un client che richiede il servizio stesso
- Diversi tipi di server: file o database, application

Client/Server

- Server sempre attivo, client non sempre
- Clients non comunicano direttamente tra loro

P2P

- Comunicazione diretta tra peers che non appartengono a fornitore di servizio
- Ibridizzazione
- scalabili

Modelli di servizio

MODELLO DI SERVIZIO: Comportamento di un server al fine di erogare un servizio

- Iterativo, Concorrente e Multithread

Server Iterativo

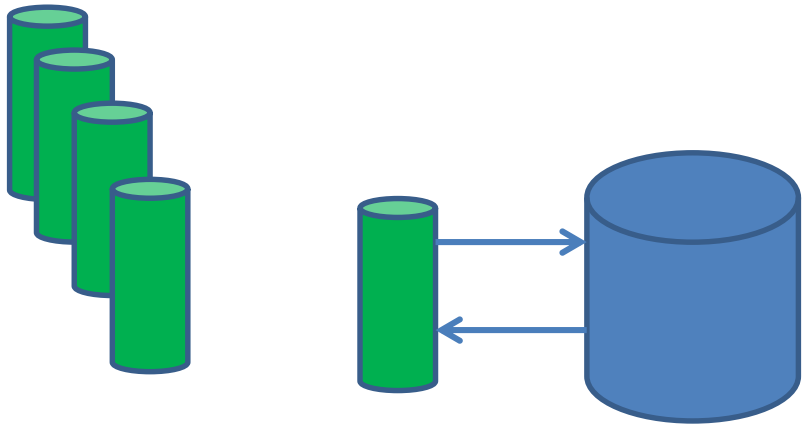
Il client invia una richiesta e

1. Server libero e richiesta soddisfatta subito
2. Server occupato
    1. Coda non è piena
    2. Coda piena e richiesta rifiutata

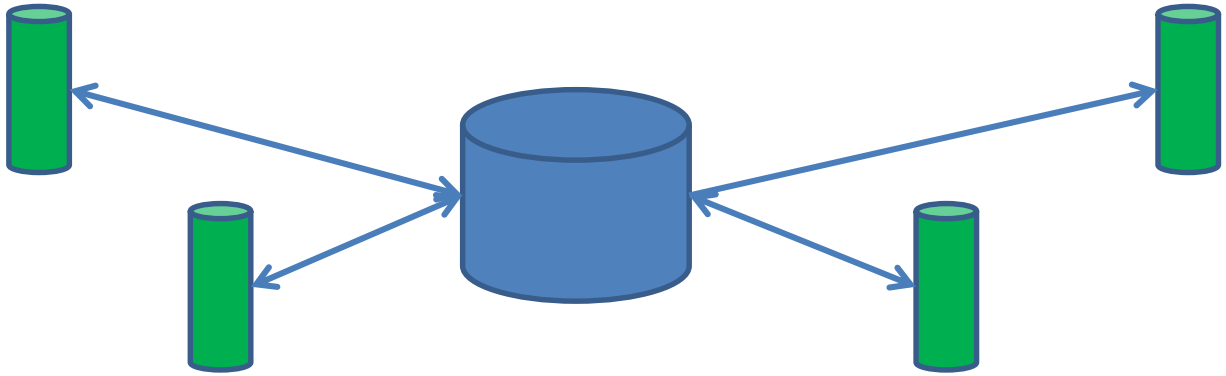
Coda gestita da SO

Starvation

Attacchi Denial of Service (DOS)



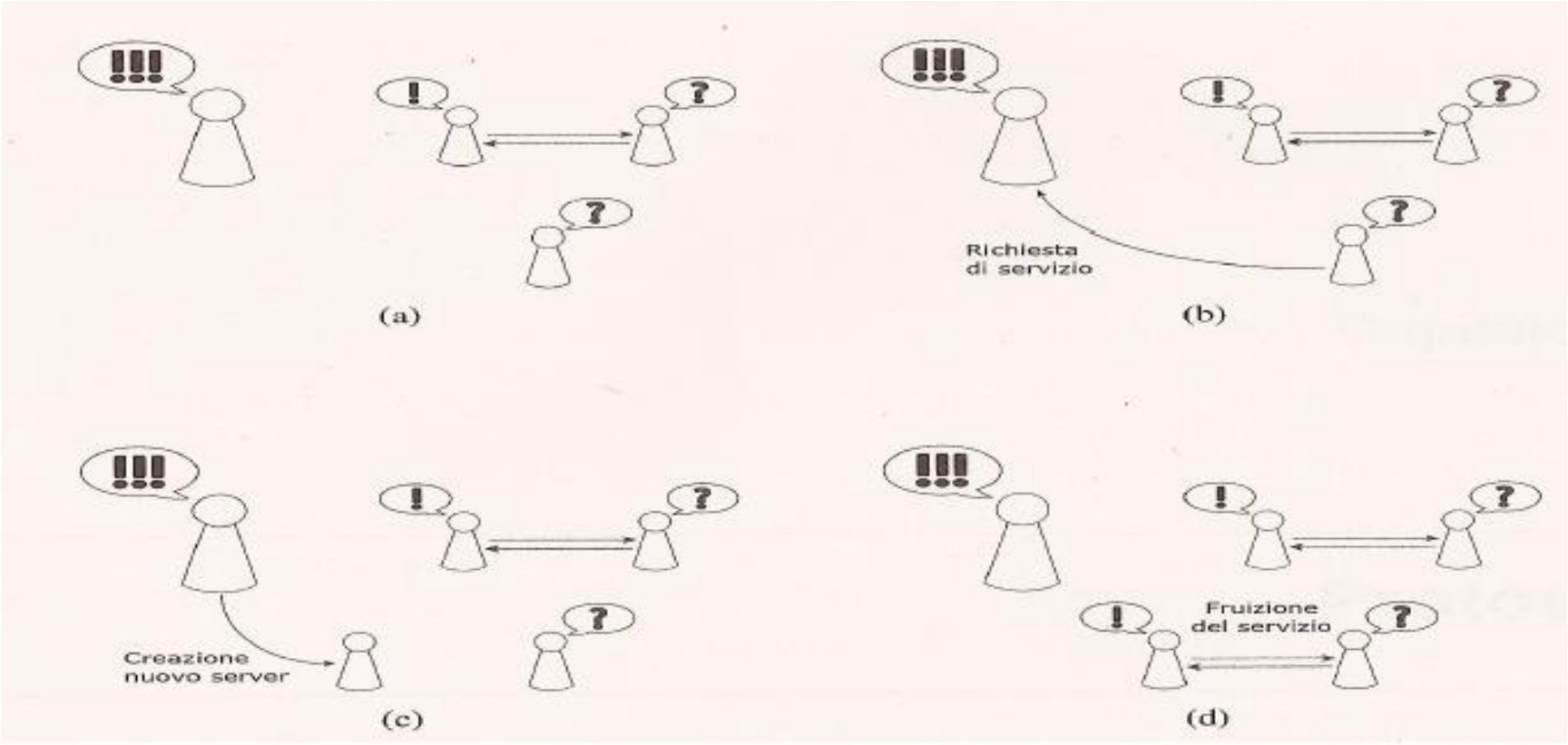
Servizio Concorrente



Parallelismo nel mantenere i rapporti ma non nell'inviare o ricevere i dati

- Limitazioni sul numero di client secondo vincoli del SO
- Un singolo server che mantiene molti canali di comunicazioni aperti simultaneamente
- Traccia dei contesti associati alle varie connessioni
- Poco scalabile

Servizio Multithread: più flussi esecutivi paralleli (processo o thread) nella fornitura del servizio



Socket

- **SOCKET**: interfaccia all'infrastruttura di comunicazione che il kernel del SO mette a disposizione dei processi per accedere ai servizi del livello di trasporto dello stack ISO-OSI

Tratto la connessione di rete come un qualsiasi stream.
Proteggere programmatore dai dettagli di basso livello

Elementi per definire una socket

Socket connessa: relazione diretta con altra socket e si stabilisce un rapporto mittente – destinatario.

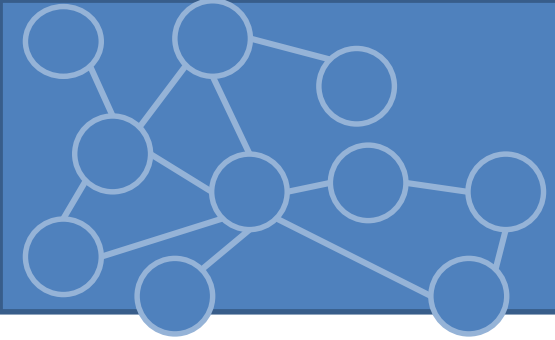
Socket non connessa: invio messaggi sulla rete fornendo ogni volta l'indirizzo della destinazione finale (può variare tra le spedizioni)

Creazione Socket

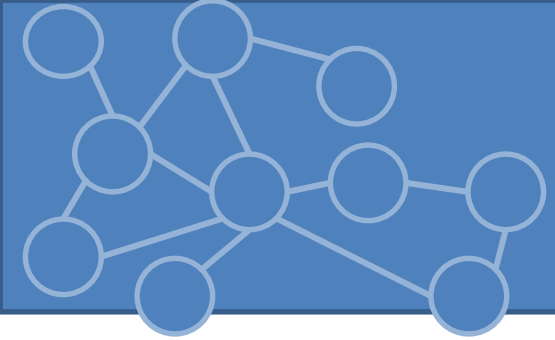
- **DOMINIO:** specifica il sistema di indirizzamento e condiziona la modalità con cui identifico e raggiungo la socket.
  - Corrispondenza con il protocollo di livello di rete utilizzato per l'instradamento Domini più comuni

    UNIX: non accede alla rete, riceve e manda msg tramite un file su disco

    Internet e Internet 6: socket identificata con IP + numero porta (32+16 bit)
  - Numero di domini supportati dipende da kernel SO e linguaggio di programmazione.



- **Modalità di trasferimento dati**
  - **Byte-stream**: dati fruiti in maniera sequenziale => sequenza di byte che genera un flusso continuo. Preserva ordine di invio e garantisce l'arrivo di tutte le info. SOLO connessa
  - **Datagram**: dati suddivisi in unità indipendenti e affidati a livello rete. Il ricevente vede un insieme di pacchetti che dovrà interpretare. No garanzia su ordine né arrivo di tutti i messaggi. Connessa e non
- **Protocollo**
  - **TCP: byte-stream**
  - UDP: datagram



Binding

Socket DEVE essere identificabile in maniera univoca per essere raggiungibile e poter creare un canale di comunicazione.

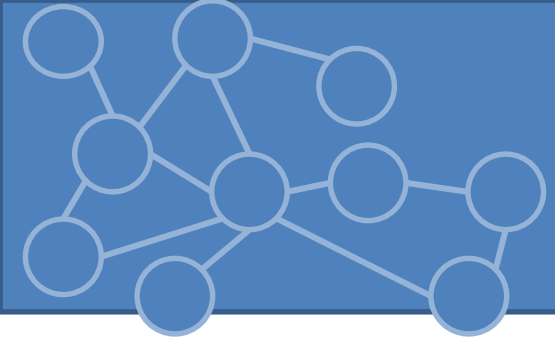
BINDING: associazione socket ad indirizzo di trasporto

- Esplicito: richiesta dell'utente di associare socket ad un indirizzo di trasporto + SO verifica che indirizzo sia disponibile (associazione livello porta e IP se possibile)
  - Binding su porta 0: SO assegna a socket il primo indirizzo non ancora allocato
- Implicito: SO in autonomia quando istituisco il canale o al primo invio dati (datagram)

Necessario per tutte e due le socket

- Ricevente: binding esplicito
- Mittente: scelta esplicito/implicito

Porte datagram e byte-stream sono disgiunte per il binding => standard associa separatamente servizi associati a TCP e UDP



Trasferimento dati

Ricevente: socket che viene contattata per creare un canale di comunicazione

Byte-stream: dichiarata disponibile per ricevere richieste di connessione

- No usata per chiamate verso altre socket, no invio e ricezione dati
- Quando contattata genera ogni volta una nuova socket su cui effettuare scambio dati
- Stessa porta ma indirizzi diversi lato mittente

Datagram: subito pronto per ricezione dati, può associarsi al mittente come conseguenza della ricezione del primo pacchetto

Mittente: creare socket e chiedere al sistema di associarla alla controparte

TRASFERIMENTO DATI

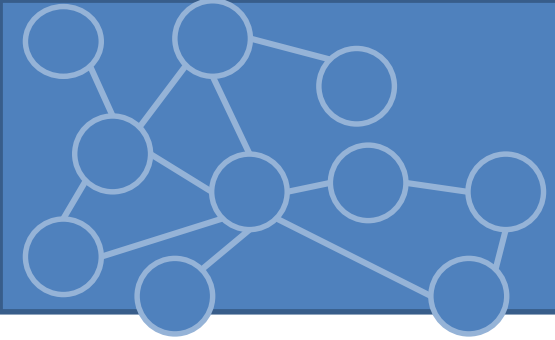
Connesse: non specifico ogni volta la destinazione, info implicitamente e permanentemente svolta durante l'associazione

Non connesse: ogni volta specifico indirizzo di trasporto. Lato ricevente ho info dell'indirizzo di trasporto del mittente. Stessa socket usata per scambiare info con un numero di entità in rete.

CHIUSURA

Esplicita: richiesta (Scelta consigliata soprattutto a livello server) .

Implicita: SO al termine del processo.



Socket - costruttori

java.net.Socket definisce una socket byte-stream TCP

- Alla base di URL, URLConnection, Applet e JEditorPane

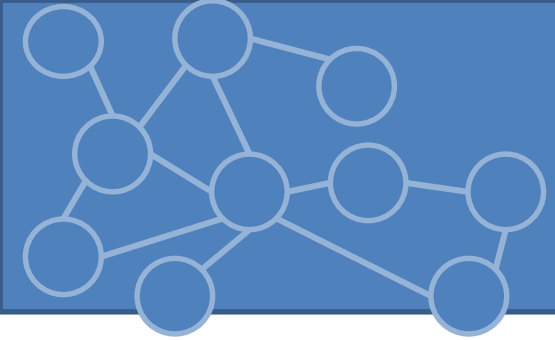
- Utilizza un'istanza della classe SocketImpl che comunica con lo stack TCP del SO
- Fornisce in sostanza streams => lettura e scrittura come i soliti stream

CREAZIONE SOCKET

Socket(String host, int port) codice:CreaSocket.java

Crea una socket TCP verso l'host specificato sulla porta specificata e tenta di connettersi all'host remoto

- Se resolver non riesce a risolvere l'hostname solleva *UnknownHostException*
- Se socket non può essere aperta solleva *IOException*



Socket(InetAddress host, int port)

Simile al precedente ma utilizza un InetAddress

- Solleva solo un IOException

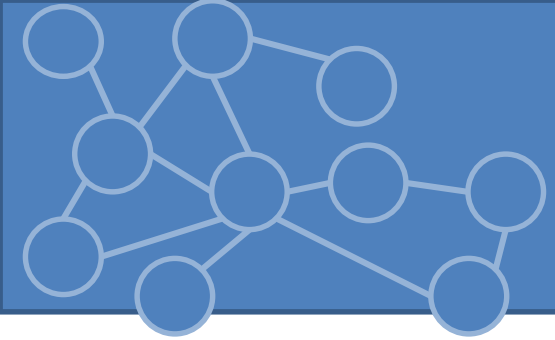
Utile per aprire più socket sullo stesso host in modo efficiente (invoco il DNS una sola volta)

Socket(String host, int port, InetAddress interface, int localPort)

Si connette all'host e alla porta specificati nei primi due argomenti dall'interfaccia e dalla porta locali specificati dai rimanenti argomenti

- Router/firewall

– Solleva IOException e UnknownException e BindException



Metodi di get

InetAddress getInetAddress()

IP dell'hot remoto a cui si è connessi, se connessione è chiusa restituisce IP a cui host era connesso.

Int getPort

Porta dell'host remoto a cui socket è connessa

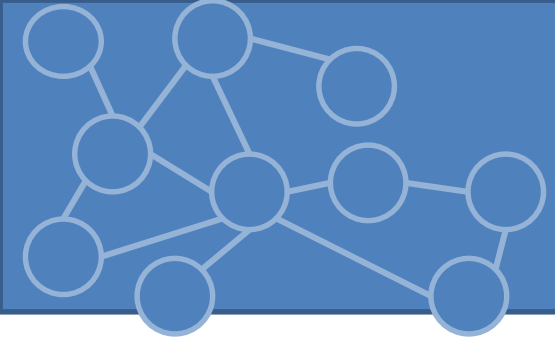
Int getLocalPort

Porta dell'host local è cui Socket è connessa. Porta locale è assegnata da SO a runtime tra le porte disponibili.

InetAddress getLocalAddress()

Indirizzo IP dell'interfaccia a cui la Socket è legata

Codice:CreaSocket.java



InputStream getInpuStream()

Restituisce un input stream da cui posso leggere i dati che host remoto invia. Solitamente viene filtrato e bufferizzato per questioni di efficienza

OutputStream getOutput() codice:Esempio5/6.java

Restituisce un output stream per inviare dati all'host remoto. Meglio filtrare e bufferizzare

close() codice:Esempio7.java

Una socket si chiude automaticamente quando uno dei due stream si chiude, il prg termina o quando oggetto viene eliminato da garbage collector. Brutta idea assumere che il sistema chiuda le socket per noi. Meglio chiudere esplicitamente, meglio se in un blocco **finally**

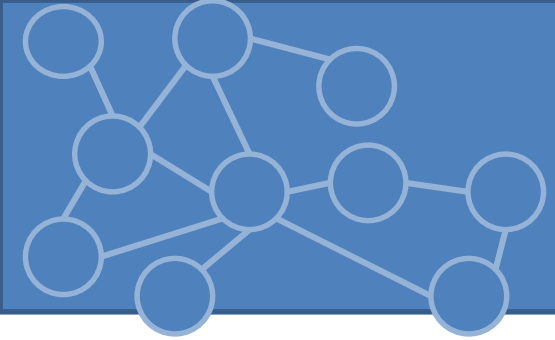
isClosed()

True se la socket è stata chiusa. Se la socket non è mai stata connessa restituisce false (comportamento controintuitivo)

isConnected()

Restuisce se la socket è mai stata connessa ad un host remoto anche se ora la socket è chiusa

Per verificare se aperta ora: s.isConnected() && !s.isClosed()



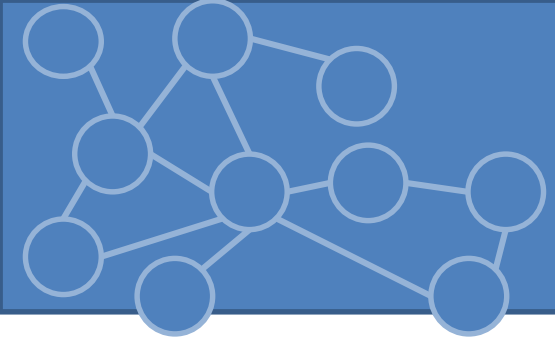
Se si vuole chiudere solo metà connessione posso usare metodi:

shutdownInput() e shutdownOutput()

Che modificano lo stream connesso alla socket in modo tale che sembri che lo stream abbia raggiunto la fine (-1 per read e IOException per write)

Devo comunque chiudere la socket quando finito perché non rilasciano le risorse associate alla socket ma solo modificano gli stati degli streams

isInputShutDown() e isOutputShutdown()



SocketAddress e InetSocketAddress

SocketAddress: endpoint di una connessione

Astratta, senza metodi e costruttore vuoto

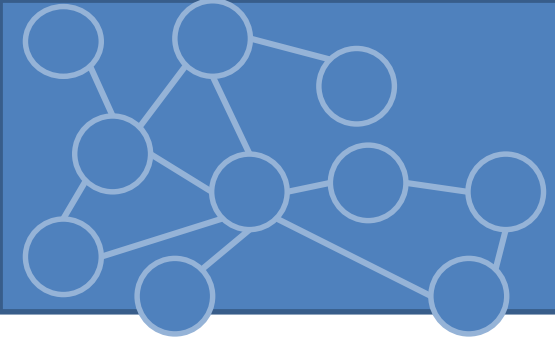
Per memorizzare IP e porta di una socket che voglio riusare.

getRemoteSocketAddres() e getLocalSocketAddress

Necessaria per connettere una socket non connessa creata con costruttore Socket(). Per connettere una socket non connessa uso il metodo **connect(SocketAddress)**

InetSocketAddress estende *SocketAddress* per memorizzare indirizzi a livello trasporto

Constructor Summary
`InetSocketAddress(InetAddress addr, int port)` Creates a socket address from an IP address and a port number.
`InetSocketAddress(int port)` Creates a socket address where the IP address is the wildcard address and the port number a specified value.
`InetSocketAddress(String hostname, int port)` Creates a socket address from a hostname and a port number.



ServerSocket

ServerSocket per creare server che accettano connessioni. Ascolta su una particolare porta del server (macchina) in attesa di richieste di connessione da parte di un client

- Quando client remoto tenta una connessione alla porta, server negozia la connessione e restituisce una Socket connessa tra i due host.

CICLO DI VITA

1. Un oggetto ServerSocket viene creato in ascolto su una porta
2. ServerSocket attende possibili connessioni usando il metodo accept(). Accept() si blocca finchè un client tenta di instaurare una connessione. Crea una Socket
3. Richiede alla Socket gli stream di output e input
4. Server e client interagiscono secondo un particolare protocollo
5. Server o client chiudono la connessione
6. Il server ritorna al passo 2 (se iterativo)

Creazione ServerSocket

ServerSocket(int port) codice:Esempio8.java

Crea una ServerSocket sulla porta specificata dall'argomento

- 0: prima porta disponibile (anonymous port). Solitamente non molto utile

BindException se socket non può essere creata e legata alla porta specificata

- Occupata da altro server o non disponibile per vincoli SO

ServerSocket(int porta, int lunghezzaCoda)

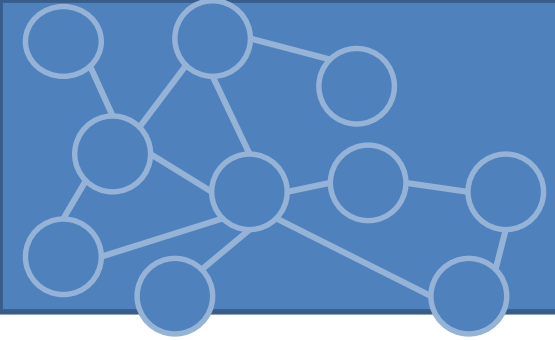
Crea una ServerSocket sulla porta specificata dall'argomento con lunghezza della coda specificata.

- Alcuni SO hanno lunghezza coda massima. La lunghezza coda è data da min(lunghezzaMaxSO, lunghezzaCoda)

ServerSocket(int port, int coda, InetAddress i)

Lega la socket alla porta specificata specificando lunghezza coda. La socket viene legata solo all'indirizzo IP specificato.

Primi due costruttori legavano la socket a tutti gli indirizzi IP locali

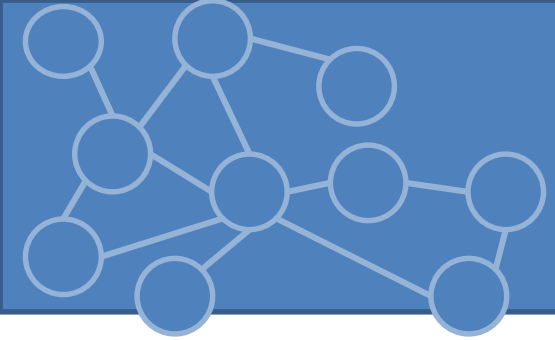


ServerSocket()

Non viene legata ad alcuna porta = non può ricevere connessioni. Viene legata usando il metodo bind()

- Permette di settare alcune proprietà della ServerSocket prima che venga legata ad una porta

```
ServerSocket ssock = new ServerSocket();
InetSocketAddress isa = new InetSocketAddress(80);
//settaggio dei parametri della ServerSocket
ssock.setSoTimeout(3000);
ssock.bind(isa);
```



Accept

ServerSocket agisce in un loop che ripetutamente accetta connessioni. Ad ogni iterazione viene invocato il metodo accept().

Accept() restituisce una Socket = connessione ta client remoto e server. Interazioni attraverso oggetto Socket restituito.

Socket accept()

Blocca esecuzione fino a che un client non si connette.

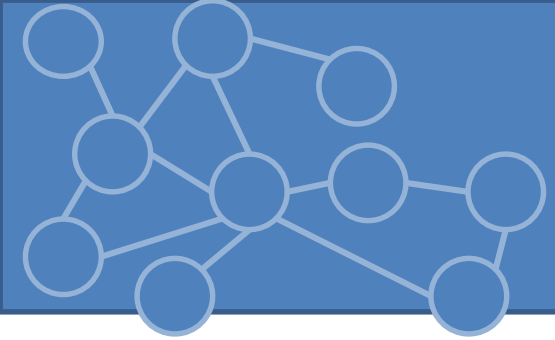
Quando client si connette resituisce un oggetto Socket.

Uso streams della Socket per comunicare col client

Gestione Eccezioni (codice:PrimoServer.java)

Importante distinguere tra eccezioni che possono chiudere il server ed eccezioni che chiudono la connessione attuale

- Eccezioni sollevate da accept() o da input/output streams non dovrebbero far terminare il server



Chiusura

close()

Libera la porta su cui ServerSocket era in ascolto + interrompe tutte le socket aperte che quella ServerSocket aveva accettato

- ServerSocket chiuse immediatamente quando un programma viene chiuso.

isClosed()

True se ServerSocket è stata chiusa. Quelle non legate non sono considerate chiuse

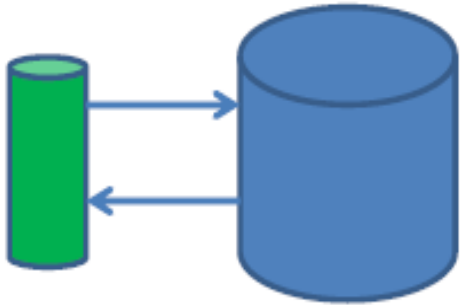
isBound()

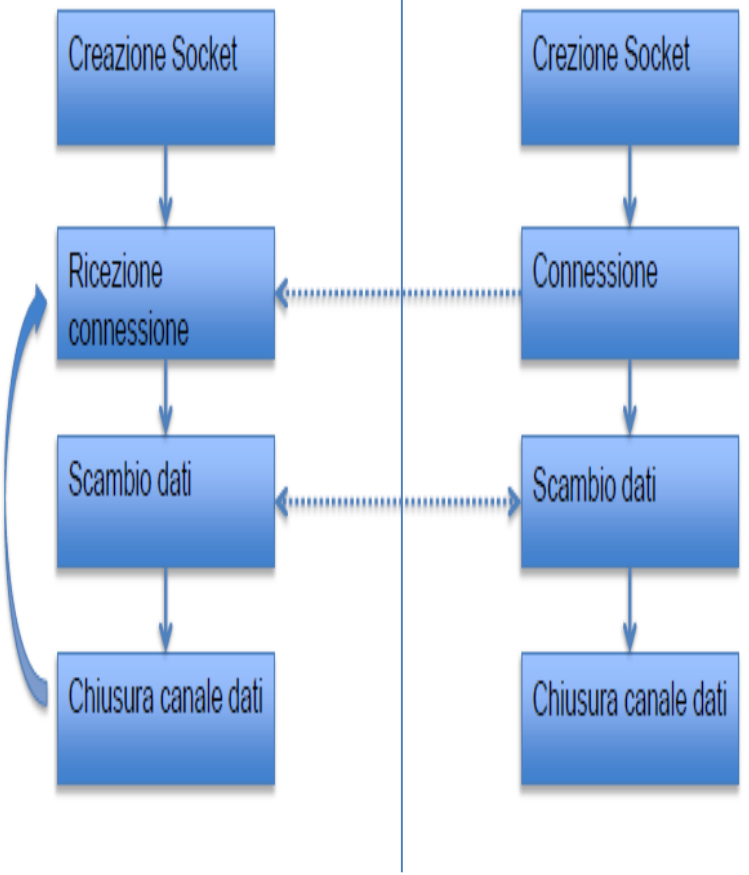
True se ServerSocket è stata legata ad una porta anche se ora è chiusa.

- Per testare se ServerSocket è aperta:

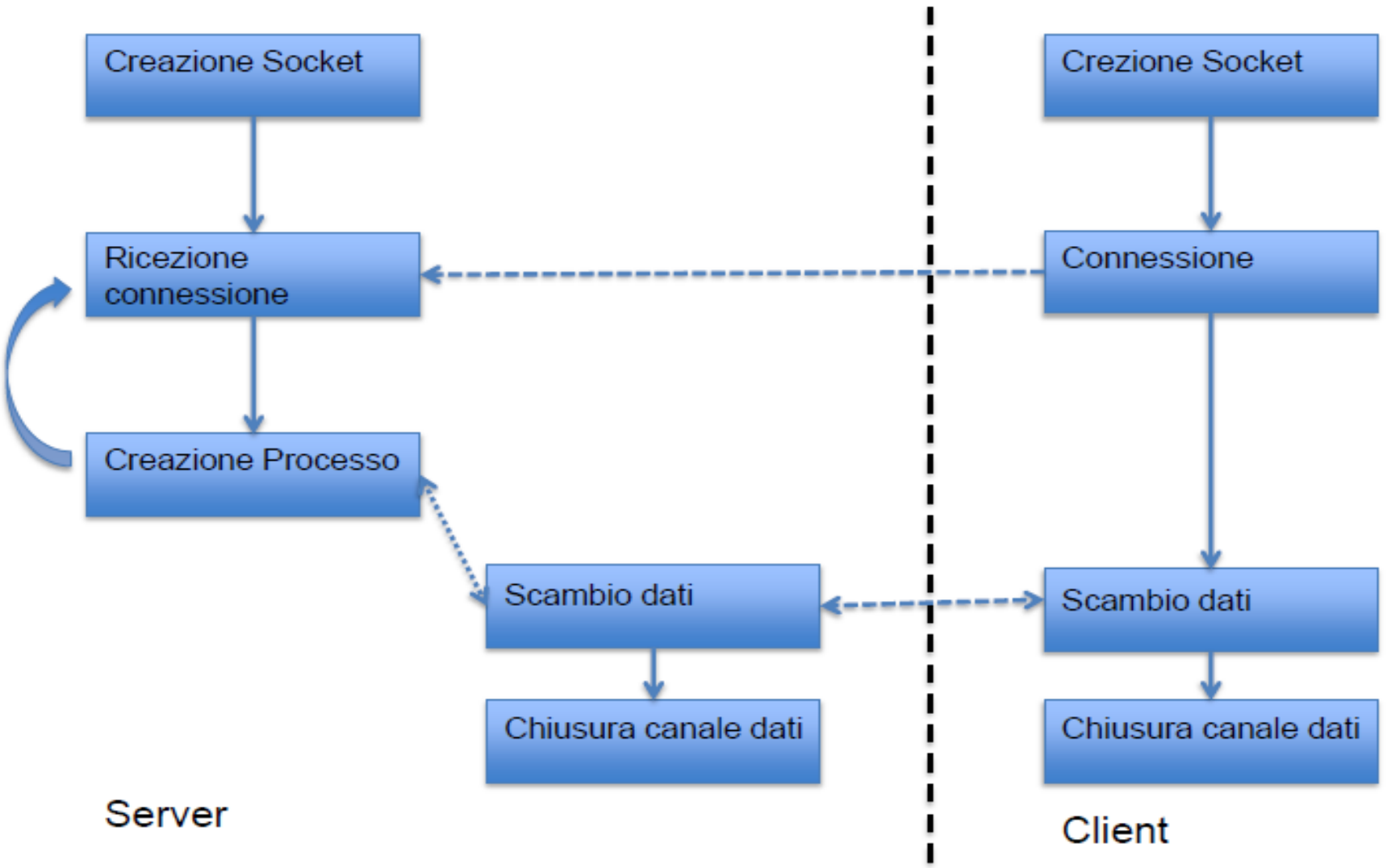
```
s.isBound && ! S.isClosed()
```

Server Iterativo

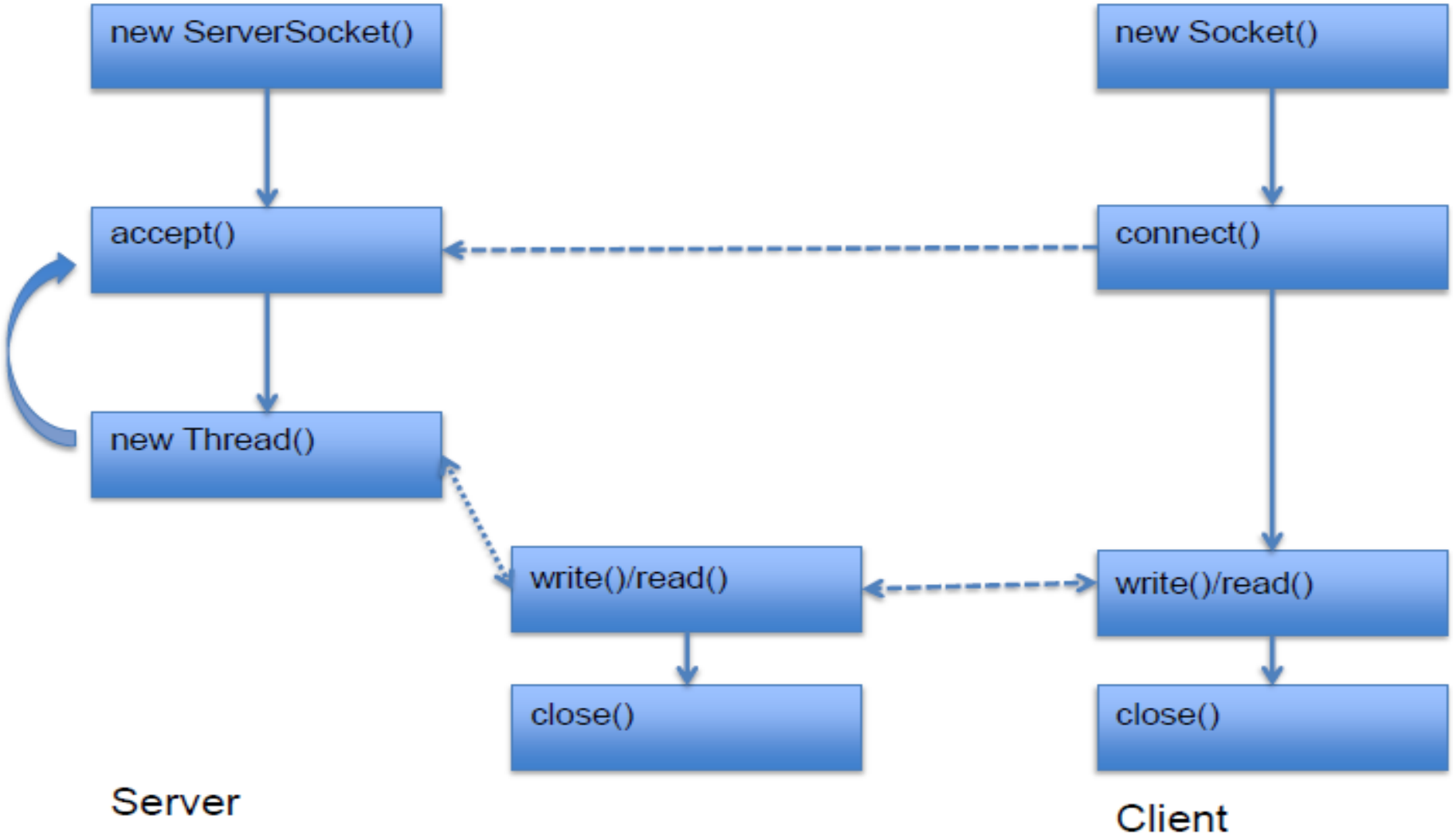




Server Multithread



In Java


new ServerSocket()
new Socket()
accept()
connect()
new Thread()
write()/read()
write()/read()
close()
close()
Server
Client