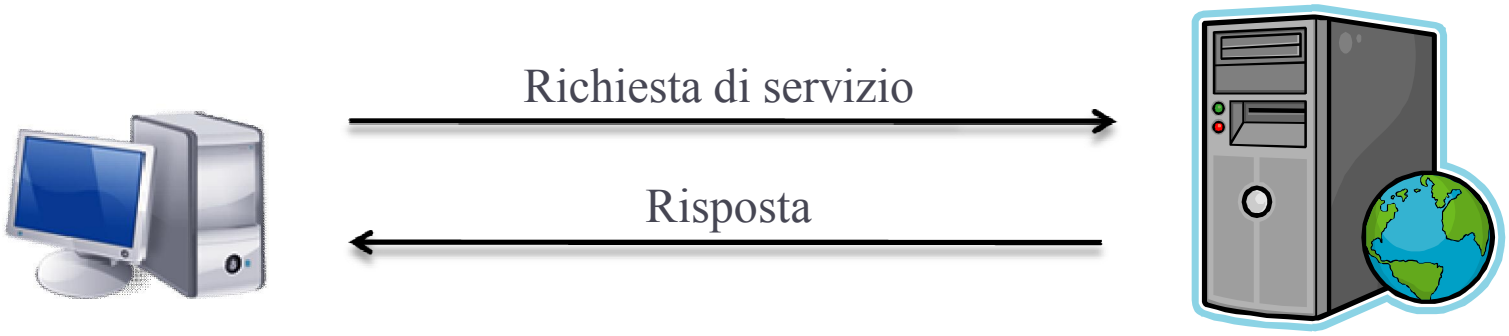
Java Socket

LSO 2008

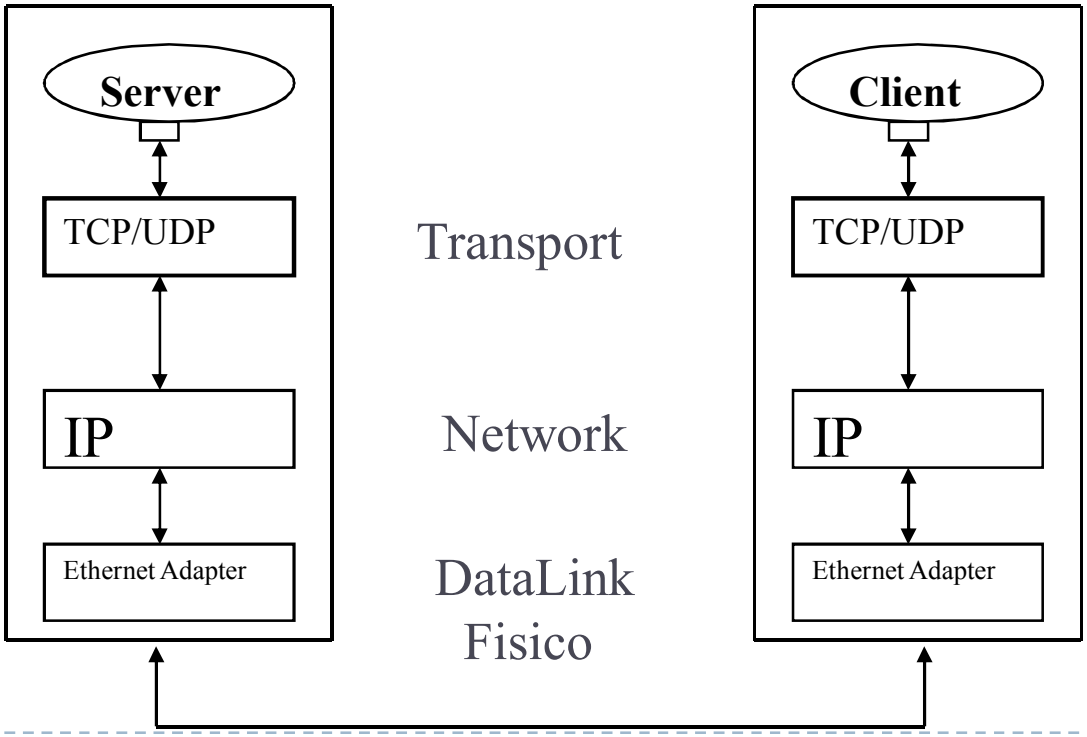
Modello Client/Server



- Il cosiddetto **lato client**, effettua la richiesta di esecuzione di un servizio. La sua controparte, il **lato server**, effettua l'esecuzione del sevizio richiesto.
- Chiaramente, è necessario che il lato client e quello server si "intendano" esattamente circa il significato della richiesta e della relativa replica. Si introduce allora il concetto di **Protocollo**.

Architettura di protocolli

I protocolli implementano quelle funzionalità che consentono lo scambio di messaggio sul mezzo fisico disposto fra due macchine, (ad esempio la suddivisione in quadri, il rilevamento di errori, il controllo di flusso ecc.)



Identificazione delle parti in gioco

- Ogni computer della rete è identificato da un indirizzo ***IP*** unico a 32 bit (Es. 130.136.32.1).
- Gli IP sono difficili da ricordare, quindi si usano dei nomi(**hostname**). Dei particolari server, chiamati DNS(Domain Name System) si preoccuperanno di dirci l'IP corrispondente al nome specificato.

  Es. www.google.it -> 209.85.135.147
- Per verificare l'associazione fra nomi e indirizzi si puo' utilizzare il comando nslookup che offre la lista di tutti i server alias per un indirizzo.

  Es. >nslookup www.google.it

Servizi forniti dai server

- I servizi che un server fornisce, sono dei processi attivi sulla macchina fisica dove esegue il server.
- Ad un servizio viene associato un numero di porta, questo permette di avere più servizi sullo stesso server.
- I numeri di porta possono essere

  - assegnati in genere a servizi di sistema (porte da 0-1023)

  - dinamici o privati (1024 - 65535)
- Servizi più famosi:
  - HTTP : porta 80
  - Telnet : porta 23,
  - FTP : porta 21
  - SSH : porta 22

La comunicazione

- Per attivare una connessione fra due processi si utilizza dunque lo schema Host:Port
  - il server è eseguito su un host e aspetta richieste su una determinata porta
  - il client viene eseguito su un host e richiede l'uso di una porta per effettuare richieste ad un server
- Specificando ora anche il protocollo per la comunicazione riusciamo a descrivere univocamente la connessione con una quintupla detta Association:

(protocollo,ind locale, proc locale, ind remoto, proc remoto)

es: (TCP,123.23.4.221,1500,234.151.124.2,4000)

Concetto di Socket

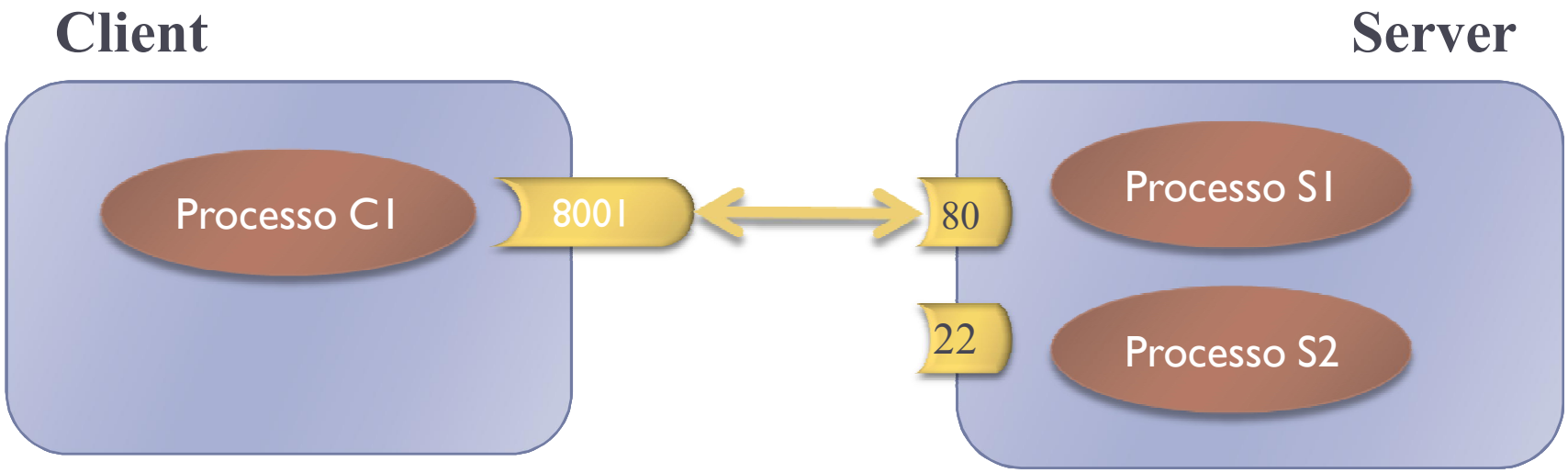
- Il socket è l'astrazione di un canale di comunicazione tra processi distribuiti in rete
- La quintupla definita viene in realtà costituita a partire da due elementi simmetrici, un'associazione locale ed una remota:
  - Protocollo, Ind. locale, porta locale (TCP,123.23.4.221,1500)
  - Protocollo, Ind. remoto, porta remota(TCP ,234.151.124.2,4000)
- Ciascuna si queste parti è detta **socket**

Nota:

protocolli differente possono usare la stessa porta;

Es. 1040 per TCP ≠ 1040 UDP

Concetto di Socket



- Un socket è un descrittore di risorsa che consente ad una applicazione di effettuare operazioni di lettura / scrittura verso un particolare dispositivo di I/O.
- Interfaccia per la suite di protocolli TCP/IP
- Punto di arrivo nella comunicazione in rete

Creazione di un socket

- Lato server
  1. Creazione del socket
  2. Bind ad una porta
  3. Listen, predisposizione a ricevere sulla porta
  4. Accept, blocca il server in attesa di una connesione
  5. Lettura - scrittura dei dati
  6. Chiusura
- Lato client
  1. Creazione del socket
  2. Richiesta di connessione
  3. Lettura - scrittura dei dati
  4. Chiusura

Il protocollo TCP

Il servizio offerto da TCP è il trasporto di un **flusso di byte** bidirezionale tra due applicazioni in esecuzione su host differenti. Il protocollo permette alle due applicazioni di trasmettere contemporaneamente nelle due direzioni, quindi il servizio può essere considerato "**Full Duplex**".

- TCP è un protocollo **orientato alla connessione**, ovvero prima di poter trasmettere dati deve stabilire la comunicazione, negoziando una connessione tra mittente e destinatario, che viene esplicitamente chiusa quando non più necessaria. Esso quindi ha le funzionalità per creare, mantenere e chiudere una connessione.

Protocollo TCP

- TCP garantisce che i dati trasmessi, se giungono a destinazione, lo facciano in ordine e una volta sola ("**at most once**"). Questo è realizzato attraverso vari meccanismi di acknowledgement e di ritrasmissione su timeout.
- TCP possiede funzionalità di controllo di flusso e di controllo della congestione sulla connessione, attraverso il meccanismo della finestra scorrevole. Questo permette di ottimizzare l'utilizzo della rete anche in caso di congestione, e di condividere equamente la capacità disponibile tra diverse sessioni TCP attive su un collegamento.
- TCP fornisce un servizio di multiplazione delle connessioni su un host, attraverso il meccanismo delle porte.

Il protocollo UDP

- A differenza del TCP, non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi.
- L'UDP ha come caratteristica di essere un protocollo di rete inaffidabile, protocollo connectionless, ma in compenso molto rapido ed efficiente per le applicazioni "leggere" o time-sensitive. Infatti, è usato spesso per la trasmissione di informazioni audio o video. Dato che le applicazioni in tempo reale spesso richiedono un ritmo minimo di spedizione, non vogliono ritardare eccessivamente la trasmissione dei pacchetti e possono tollerare qualche perdita di dati.

Protocollo UDP

- L'UDP fornisce soltanto i servizi basilari del livello di trasporto, ovvero:
  - multiplazione delle connessioni, ottenuta attraverso il meccanismo delle porte
  - verifica degli errori mediante una checksum, inserita in un campo dell'intestazione del pacchetto.
- mentre TCP garantisce anche il trasferimento affidabile dei dati, il controllo di flusso e il controllo della congestione.
- UDP è un protocollo stateless ovvero privo di stato: non mantiene lo stato della connessione dunque rispetto a TCP ha informazioni in meno da memorizzare. Un server dedicato ad una particolare applicazione che sceglie UDP come protocollo di trasporto può supportare molti più client attivi.

Introduzione ai socket in Java

- L'interfaccia deriva da quella standard di Berkeley, introdotta nel 1981
- E' organizzata in maniera da soddisfare i principi dell'Object Orienting Programming
- Contiene classi per la manipolazione degli indirizzi e la creazione di socket lato client o server
- Implementazione nel package **java.net**

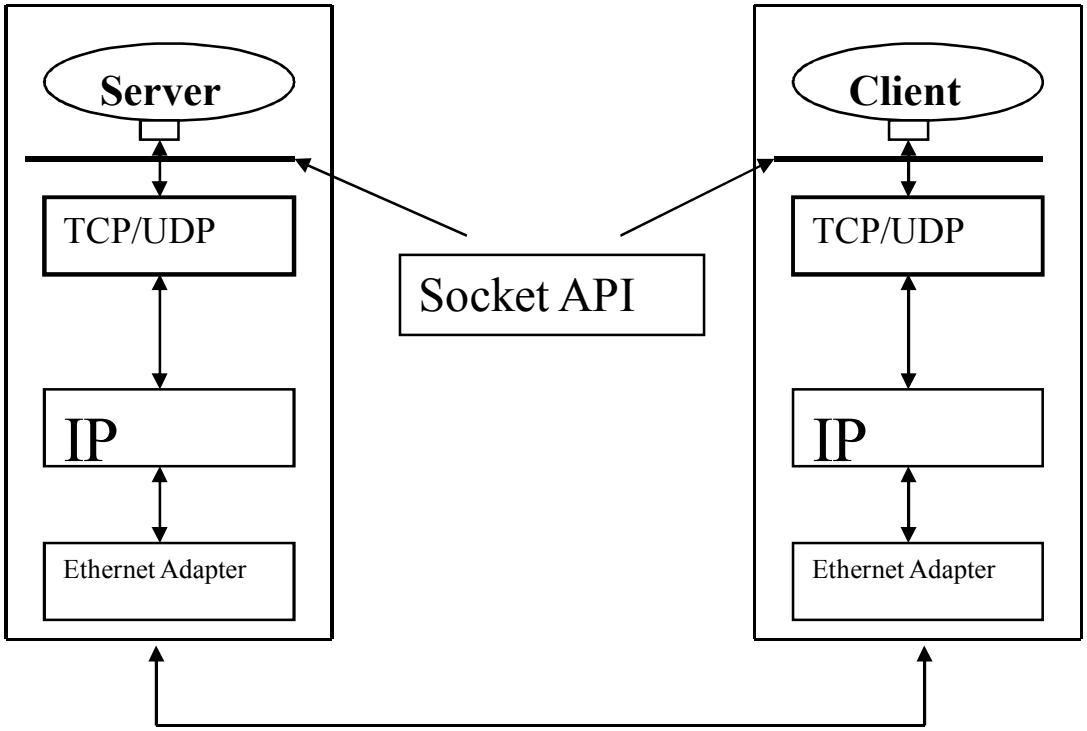
Il package Java.net

Il Java Networking API (java.net) fornisce le classi e interfacce per le seguenti funzioni

- **Indirizzamento**
  - **InetAddress**
- **Creazione di connessioni TCP**
  - **ServerSocket,Socket**
- **Creazione di connessioni UDP**
  - **DatagramPacket,DatagramSocket**
- **Localizzare risorse di rete**
  - **URL, URLConnection, HttpURLConnection**
- **Sicurezza (autenticazione, permessi)**

Interfaccia socket in Java

I server e i client si scambiano messaggi attraverso le Socket API che si collocano immediatamente sopra al livello transport della comunicazione



La classe InetAddress

- Rappresenta un indirizzo IP e fornisce i metodi per manipolarlo. Non offre costruttori pubblici ma dei metodi statici per creare istanze di questa classe:
  - **InetAddress InetAddress.getByName(String hostname)**
  - **InetAddress[] InetAddress.getAllByName(String hostname)**
  - **InetAddress InetAddress.getLocalHost()**

- Una volta istanziato un oggetto di questa classe si possono utilizzare i metodi
  - **String getHostAddress()**
  - **String getHostName()**

La classe InetAddress

- Il metodo **getByName()** può impiegare il DNS per recuperare l'indirizzo IP associato all'hostname fornito come argomento
  - Si può specificare nel suo argomento sia un indirizzo IP nella forma decimale che il nome di un host
  - In entrambi i casi, l'oggetto InetAddress restituito conterrà l'indirizzo IP a 32 bit
- Nel caso che ad un hostname siano associati più indirizzi IP, li si può ottenere chiamando la **getAllByName()**
- La **getLocalHost** restituisce l'indirizzo IP dell'host su cui viene eseguita
- **getHostName** e **getHostAddress** restituiscono rispettivamente il nome e l'indirizzo IP che rappresentano

Vedi InetAddressTest.java

Classi per il protocollo TCP

- Java fornisce due diverse classi per la comunicazione con il protocollo TCP che rispecchiano la struttura client/ server:
  - creazione socket per il server : classe **ServerSocket**
  - creazione socket per il client : classe **Socket**

- La differenziazione del socket Client e Server è dovuto alle diverse operazioni che vengono svolte al momento di stabilire una connessione
  - Un server ottiene l'oggetto socket da una chiamata al metodo accept,
  - Il client deve provvedere a creare un'istanza del socket.

Classe ServerSocket

- Implementa un server socket : viene utilizzata da server che accettano connessioni dai client. Costruttori:
  - **ServerSocket()**
  - **ServerSocket(int port)**
  - **ServerSocket(int port,int backlog)**
  - **ServerSocket(int port, int backlog, InetAddress bindAdd)**

- Il parametro port specifica la porta su cui rimanere in attesa, il parametro backlog il numero massimo di richieste di connessione (default 50) mentre il parametro bindAdd viene utilizzato dai server multihomed(con più interfacce di rete) per specificare un determinato indirizzo

Classe ServerSocket

- Non è necessario specificare la famiglia dei protocolli; si opera sempre con IP
- L'impiego di questa classe implica l'uso del protocollo TCP
- Un eventuale errore genera una eccezione IOException sollevata dai costruttori
- Il costruttore realizza tutte le operazioni di socket(), bind() e listen()

Classe ServerSocket

- Il metodo più importante è **accept**, che blocca il server in attesa di una connessione. E' questo il metodo che restituisce un oggetto di tipo socket (con le stesse proprietà di quello originale) completamente istanziato nei parametri (dati locali e remoti) che viene poi utilizzato per gestire la comunicazione con il client.
  - Socket newconnection = myServerSocket.accept();
- Genera in caso di errore un'eccezione IOException
- Solo dopo che il metodo accept() ritorna un socket valido è possibile eseguire operazioni di I/O su quel socket

Classe ServerSocket

- Sono disponibili anche altri metodi che permettono di gestire altri aspetti legati ai socket, come ad esempio timeout o buffer
- **setSoTimeout(int timeout)**:fissa il tempo massima di attesa per una accept
- **setReceiveBufferSize(int size):** permette di settare la dimensione del buffer di ricezione associato al socket

Classe Socket

- Rappresenta una socket client ed è quindi un estremo della comunicazione fra due macchine. Costruttori:
  - **Public Socket(String host, int Port)**
  - **Public Socket(InetAddress, int Port)**

- Questi costruttori specificano nel primo argomento l'host remoto e nel secondo la porta del server con cui effettuare la connessione. Altri costruttori permettono di creare il socket associandolo ad una determinata porta e indirizzo locale

Classe Socket

- Non è necessario specificare la famiglia dei protocolli;si opera sempre con IP
- L'impiego di questa classe implica l'uso del protocollo TCP
- Puo' generare le eccezioni

  - IOException - errore creato alla creazione del socket

  - UnknownHostException - l'host non è stato trovato
- realizza le operazioni creazione socket , connect

Classe Socket

Questa classe fornisce alcuni metodi per poter estrarre e manipolare informazioni sul socket creato:

- **getLocalAddress, getInetAddress, getPort, getLocalPort** forniscono gli indirizzi e le porte di accesso della connessione
- **setTcpNoDelay** forza la spedizione dei dati senza che il pacchetto TCP sia completo
- **sendUrgentData** spedisce un byte di dati urgenti sul socket scavalcando altre operazioni di write in sospeso
- **setSoTimeout** specifica il tempo di attesa in lettura sul socket
- **setSendBufferSize, setReceiveBufferSize** specificano le dimensioni dei buffer di lettura e scrittura

Classe Socket

Metodi per Input/Output dei dati

- **InputStream getInputStream()** restituisce uno stream in lettura per il socket
- **OutputStream getOutputStream()** restituisce uno stream in scrittura per il socket

ServerSocket e Socket:gestione degli errori

- E' importante che i socket siano propriamente chiusi al termine da una applicazione mediante il metodo close() delle classi (sia Socket che ServerSocket) poiché questi sono una risorsa di rete del sistema
- La chiusura dei socket deve essere eseguita indipendentemente dal flusso di esecuzione del programma, utilizzando per questo le sezioni try e finally

Vedi package socketexample

1. *ServerIterativo.java*
2. *ServerConcorrente.java*
3. *ClientExample.java*

Classi per il protocollo UDP

- Questo protocollo basa la comunicazione sullo scambio di messaggi, che vengono inviati singolarmente da un computer all'altro.
- I messaggi consistono in array di byte che vengono inviati al mittente
- I messaggi contengono tutte le informazioni necessarie per essere recapitati
- I messaggi sono inviati uno indipendentemente dall'altro
- Le classi che gestiscono questo tipo di comunicazione iniziano per **Datagram**

Classe DatagramPacket

- Questa classe serve per preparare il nuovo pacchetto da spedire attraverso la rete con la classe DatagramSocket. Costruttori:

  - public DatagramPacket(byte[] buf, int length);
  - public DatagramPacket(byte[] buf, int offset,int length);
  - public DatagramPacket(byte[] buf, int offset,InetAddress address,int port);

Classe DatagramPacket

- Non è necessario specificare la famiglia dei protocolli;si opera sempre con IP
- L'impiego di questa classe implica l'uso del protocollo UDP
- Puo' generare le eccezioni

  - IOException - errore creato alla creazione del socket
  - UnknownHostException - l'host non è stato trovato

Classe DatagramPacket

- Sono forniti anche altri metodi per gestire i pacchetti creati:
  - **setAddress,setPort,setData,**
  - **setSocketAddress**

- Permettono di specificare la porta e l'indirizzo a cui spedire il pacchetto, oltre che specificare i dati da spedire

Classe DatagramSocket

- Serve per aprire un socket con il quale inviare e ricevere un DatagramPacket utilizzando il protocollo UDP. Se il socket è usato solo per inviare non è necessario specificare la porta, metre se si vuole anche ricevere si deve specificare la porta a cui il Datagram è associato.Costruttori:
  - **public DatagramSocket();**
  - **public DatagramSocket(int port);**
  - **public DatagramSocket(int port, InetAddress addr);**

Si puo' specificare la porta e l'indirizzo locali su cui si vuol ricevere

Classe DatagramSocket

- Non è necessario specificare la famiglia dei protocolli;si opera sempre con IP
- L'impiego di questa classe implica l'uso del protocollo UDP
  - non stabilisce nessuna connessione fra client e server
  - non garantisce né l'ordinamento né l'arrivo dei pacchetti
- L'overhead basso permette una trasmissione molto veloce
- Puo' generare le eccezioni
  - IOException - errore creato alla creazione del socket
  - UnknownHostException - l'host non è stato trovato

Classe DatagramSocket

- **send(DatagramPacket p)** spedisce un pacchetto sul socket, includendo tutte le informazioni necessarie per l'invio
- **receive(DatagramPacket p)** riceve un pacchetto sul socket. Il pacchetto ricevuto contiene indirizzo e porta della macchina che lo ha inviato. Blocca il thread fino alla ricezione di un pacchetto. Se il messaggio è più lungo della lunghezza del pacchetto, il messaggio viene troncato
- **getLocalAddress(), getLocalPort()….**
- **setSoTimeout(),setSendBufferSize, setReceiveBufferSize**()...

Classe DatagramSocket

- Poiché questo tipo di connessione è a scambio di pacchetti, non si hanno metodi per creazione di stream, ma solo per impostare o estrarre i contenuti dei pacchetti
- uso i costruttori per inserire i dati nei pacchetti, uso il metodo **getData()** per estrarre i dati dai pacchetti
- i dati sono sempre contenuti in array di byte, si effettuano delle operazioni di casting per ottenere dati tipo Stringa, interi ecc. a partire dagli array.

```
String messaggio;
byte[] msg = p.getData();
messaggio = new String(msg);
```

Vedi package socketexample

1. *ServerDatagram.java*
2. *ClientDatagra.java*

Classe URL

- La classe URL rappresenta una Uniform Resource Locator, un puntatore ad una risorsa nel WWW. Una risorsa puo' essere un file, una directory, o un oggetto di rete generico.

  Http://java.sun.com/index.html
- I protocolli possono essere vari, ma i più utilizzati sono http e ftp. La specifica del protocollo è necessaria
- Costruttori
  - **URL(String URL)**
  - **URL(String protocollo, String host, String port, String file)**

Classe URL

L'accesso ai dati riferiti dall'URL è possibile in3 modi:

- **URLConnection openConnection()** ritorna un oggetto URLConnection che rappresenta la connessione con l'oggetto remoto rappresentato dall'URL. Nel caso esistano delle sottoclassi specializzate di URLConnection, la connessione ritornata sara' di questo tipo (es. HttpURLConnection)
- **InputStream openStream()** apre una connessione e ritorna un InputStream per la lettura da questa connessione. In pratica esegue: (openConnection()).getInputStream
- **getContent()** restituisce il contenuto dell'URL
- altri metodi permettono si regolare i parametri di questa classe:**set(...), getHost(), getPort()...**

Classe HttpURLConnection

- Rappresenta una connessione ad un URL con supporto del protocollo HTTP
- Varie istanze di questa classe sullo stesso server possono condividere una stessa connessione di rete. In questo caso la chiusura di un InputStream o OutputStream rilascia le risorse relative all'istanza associata, ma non ha effetto su altre connessioni persistenti.
- Definisce una serie di constanti per rappresentare gli status code ritornati dal server, es.

  HTTP_OK=200;HTTP_NON_FOUND=404;

  HTTP_INTERNAL_ERROR=500

Classe HttpURLConnection

- Si puo' gestire la connessione Http tramite alcuni metodi
  - **setRequestMethod()** specifica il tipo di richiesta da eseguire
  - il contenuto della richiesta è inviato attraverso un OutputStream
  - **getResponseCode()** resituisce il codice della risposta del server
  - **getResponseMessage()** restituisce il messaggio di risposta del server
  - è possibilie gestire automaticamente la redirezione delle richieste con **setFollowRedirects()…**

Vedi package socketexample

1. *HttpUrlConnectionExample.java*

Link Utili

- Documentazione ufficiale sul package java.net
  - http://java.sun.com/j2se/1.4.2/docs/api/java/net/package-summary.html

- Esempi su codice Java
  - http://www.google.it