Socket e programmazione client-server in JAVA

Advanced Computer Programming

Prof. Luigi De Simone

Sommario

- Socket TCP/UDP in Java
- Server Multithread
- InetAddress
- URL e URLConnection

Riferimenti:

- http://docs.oracle.com/javase/tutorial/networking/
- Harold, “Java Network Programming”, 4th Ed, O’Reilly
  - Capitoli 8, 9, 12
- Hughes, Shoffner et. Al. “Java Networking Programming”, 2nd Ed, Manning, 2000
  - Capitoli 14, 16, 20

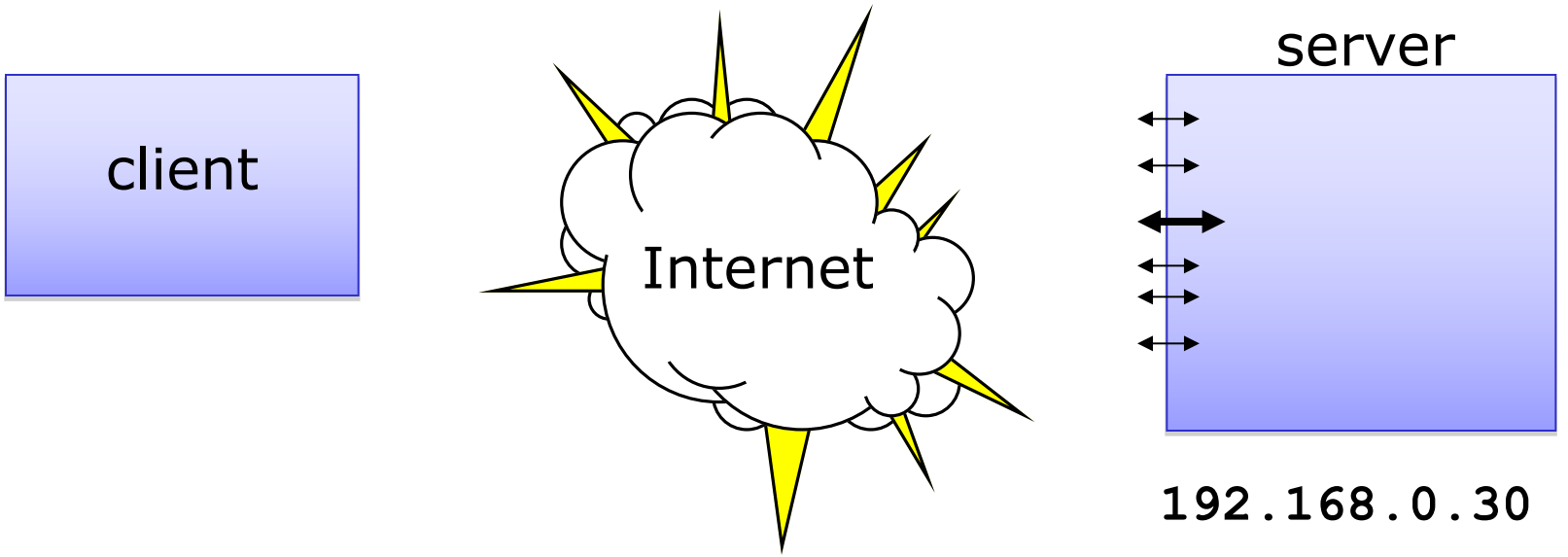
Socket TCP/IP in JAVA

Per comunicare su rete tra applicazioni distribuite, JAVA mette a disposizione le seguenti classi:

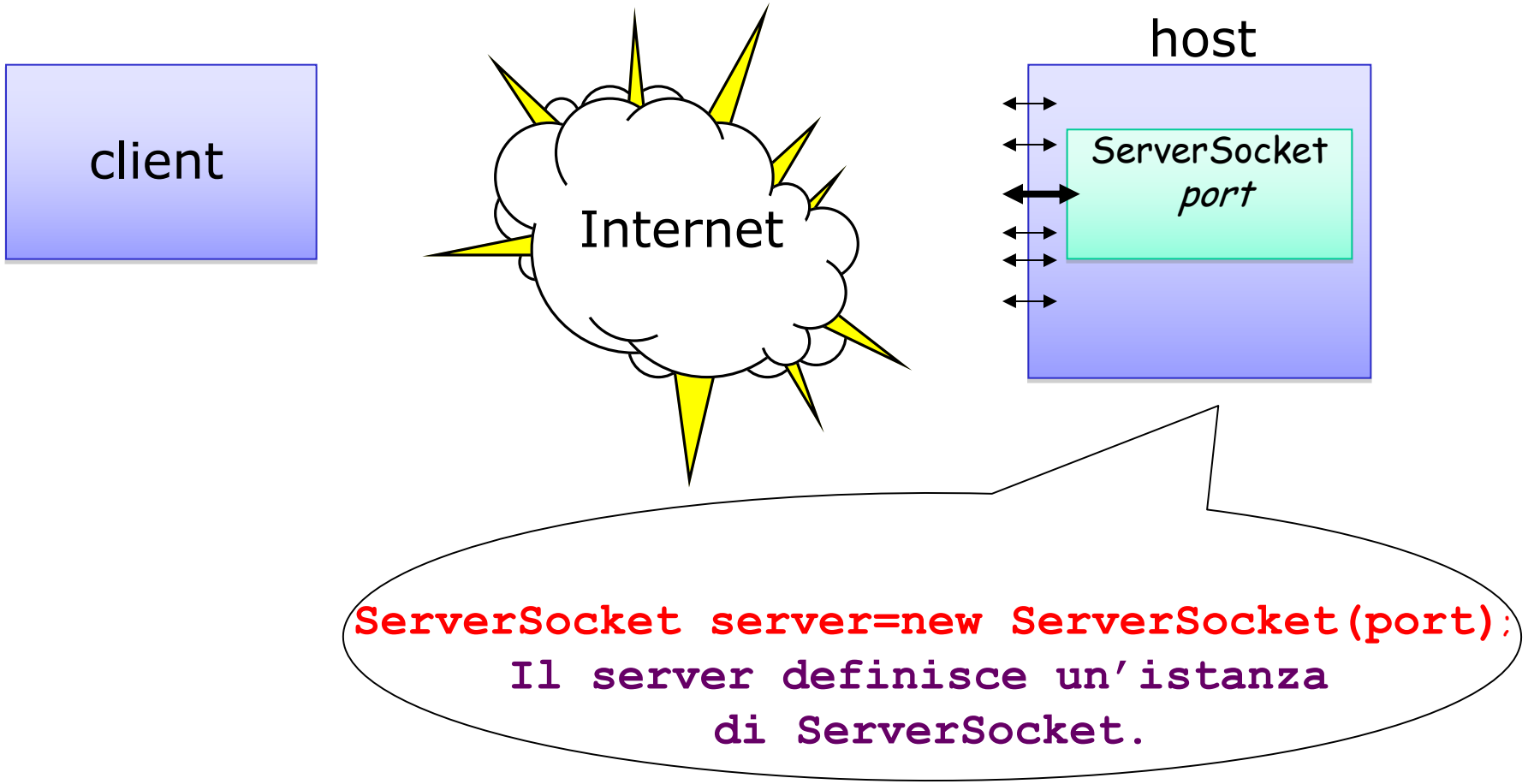
- la classe Socket, per **creare una connessione di rete**
- La classe ServerSocket utilizzata da un **server**, per **accettare una richiesta di connessione**

Programmazione client-server TCP/IP

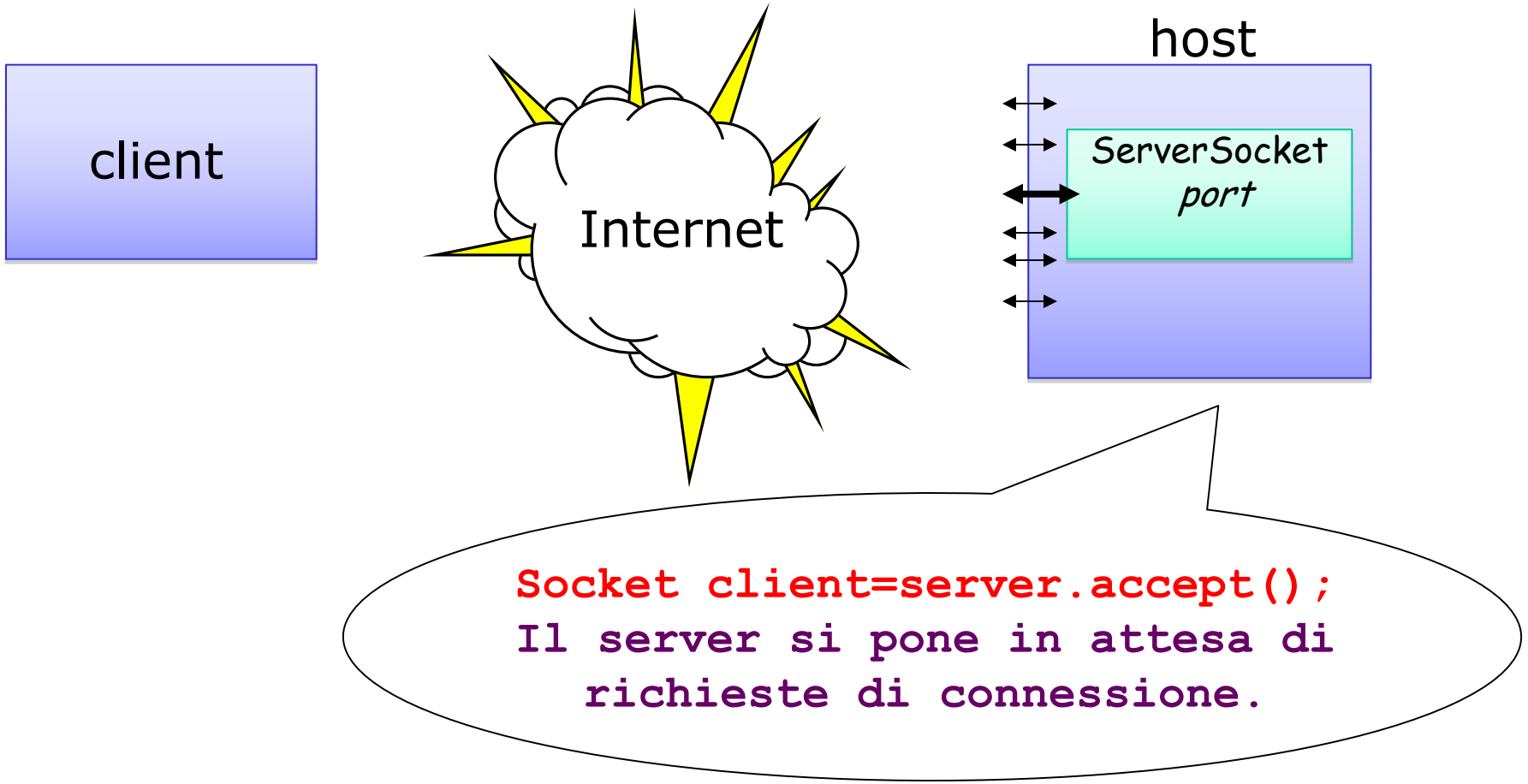
Il processo server che gira sul nodo X, si pone in attesa di richieste di connessioni sul porto Y. Utilizzo della classe **ServerSocket**



Programmazione client-server TCP/IP



Programmazione client-server TCP/IP



Classe ServerSocket

- La classe **ServerSocket** ha due costruttori:
  - `ServerSocket(int port) throws IOException;`
  - `ServerSocket(int port, int backlog) throws IOException.`

- Il parametro port indica il "port number" sull'host locale (può assumenre valori 1-65535, anche se i valori da 1 a 1023 sono riservati);

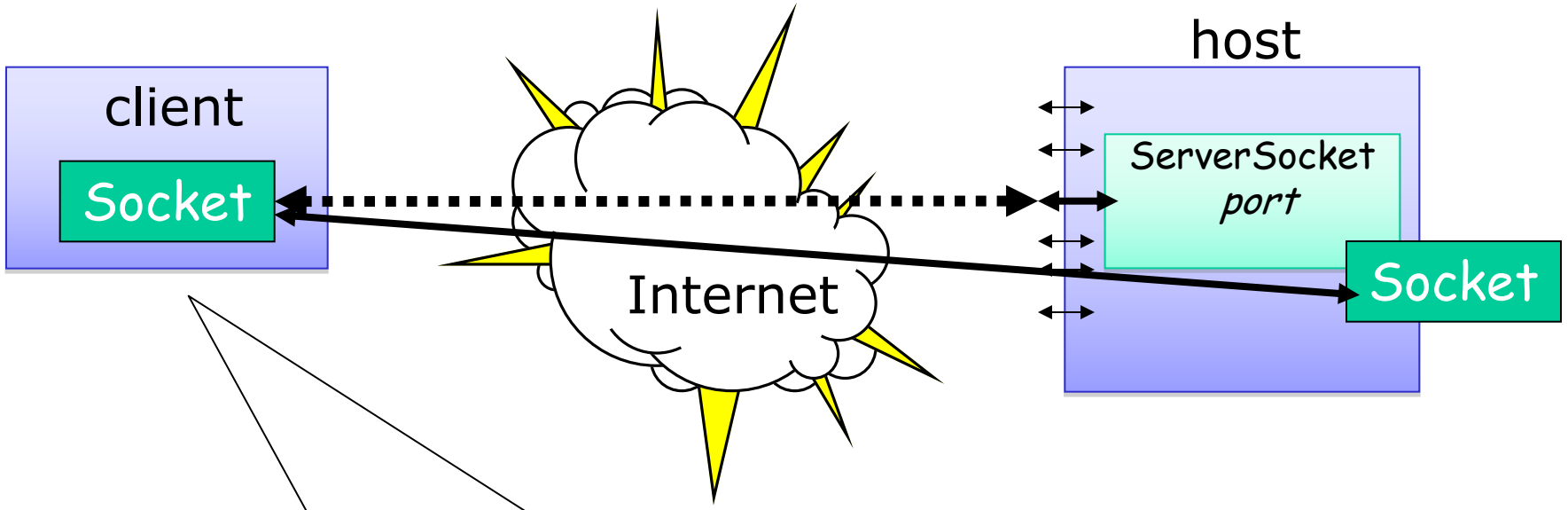
- Il parametro **backlog** indica il **numero di massimo di richieste di connessione che possono essere accodate** dal sistema operativo. Utilizzando il primo costruttore tale parametro assume il valore di **default di 50**.

Classe ServerSocket

Il metodo più importante è il seguente:

- `Socket accept() throws IOException`
  - Comporta l'attesa (sul port number del server) di una richiesta di connessione (listening).
  - Alla ricezione di una connessione, l'oggetto ServerSocket ***crea una socket che rappresenta la connessione TCP con il client*.**
  - Il riferimento a tale socket verrà restituito al chiamante

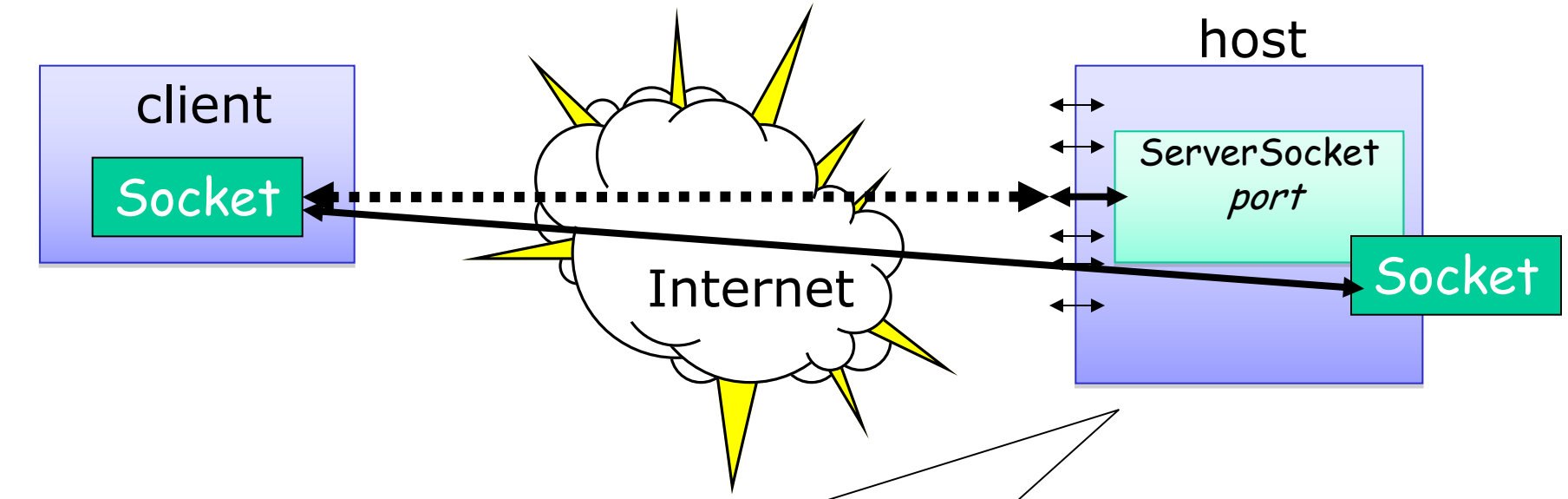
Programmazione lato client




```
socket = new Socket(address, PORTNUM);
```

Il client istanzia un oggetto di tipo Socket, e richiede una connessione.

Effetto lato server



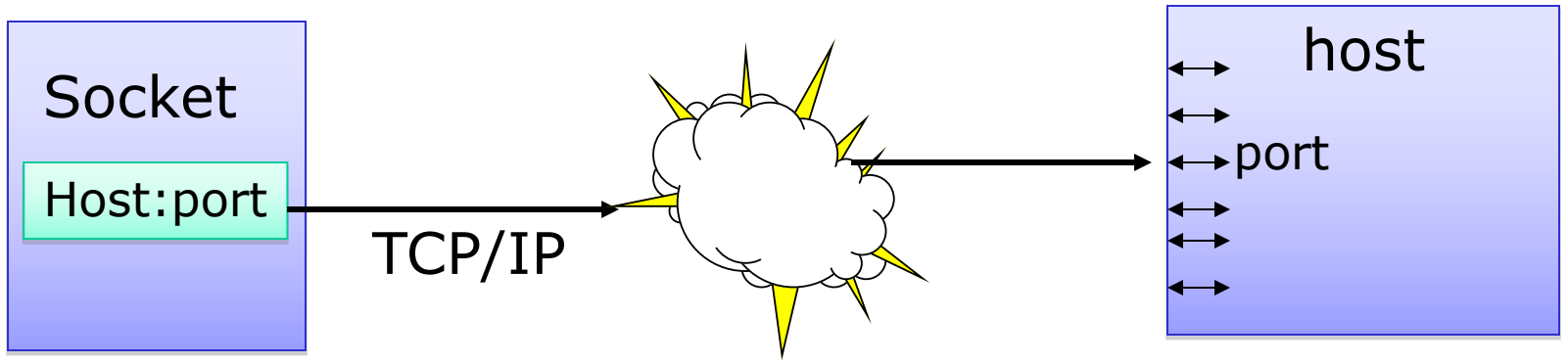
Il server esce dall'accept() che restituisce la socket per la comunicazione con il client

La classe Socket

La classe Socket fornisce un' **interfaccia socket** per i client TCP.

Per aprire una connessione con il server bisogna creare un' istanza di tale classe

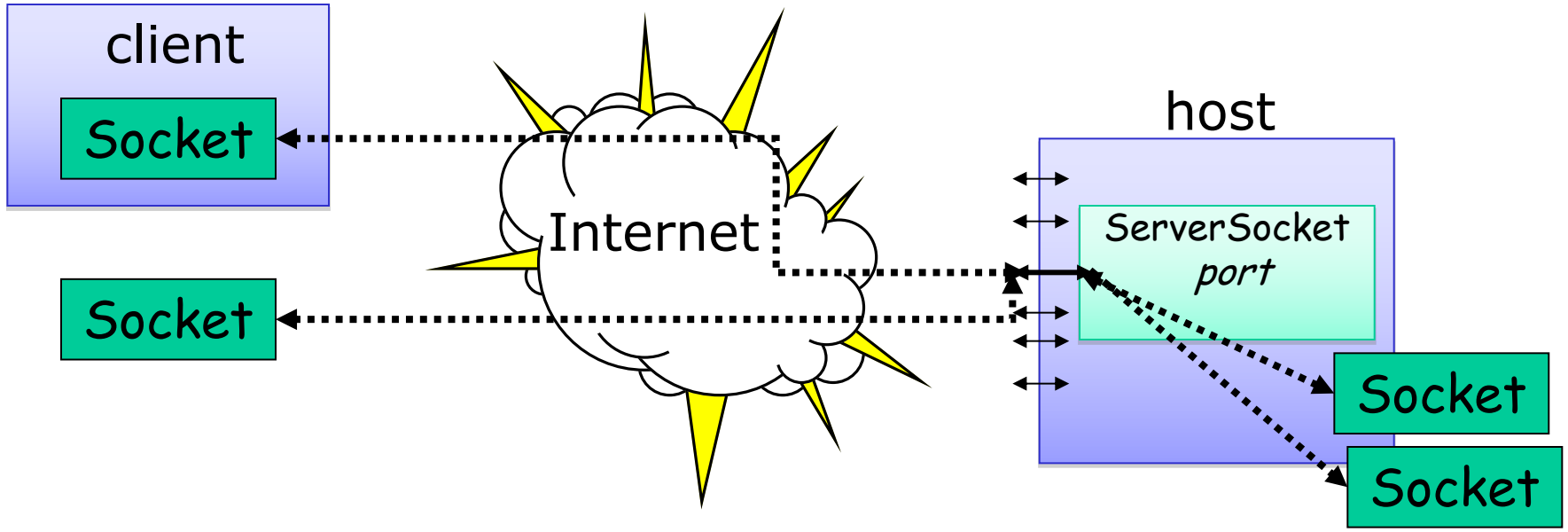
```
Socket conn= new Socket(nomeHost, numport)
```



Classe ServerSocket

La classe ServerSocket:

- come detto in precedenza, ServerSocket incapsula le strutture dati per la gestione delle connessioni TCP/IP server-side.
- invocando nuovamente l'accept il server può essere messo in attesa di una nuova connessione client.



Comunicazione client/server

Invocando i metodi **getInputStream()**, **getOutputStream()** di Socket, è possibile instaurare il canale di comunicazione tra client e server

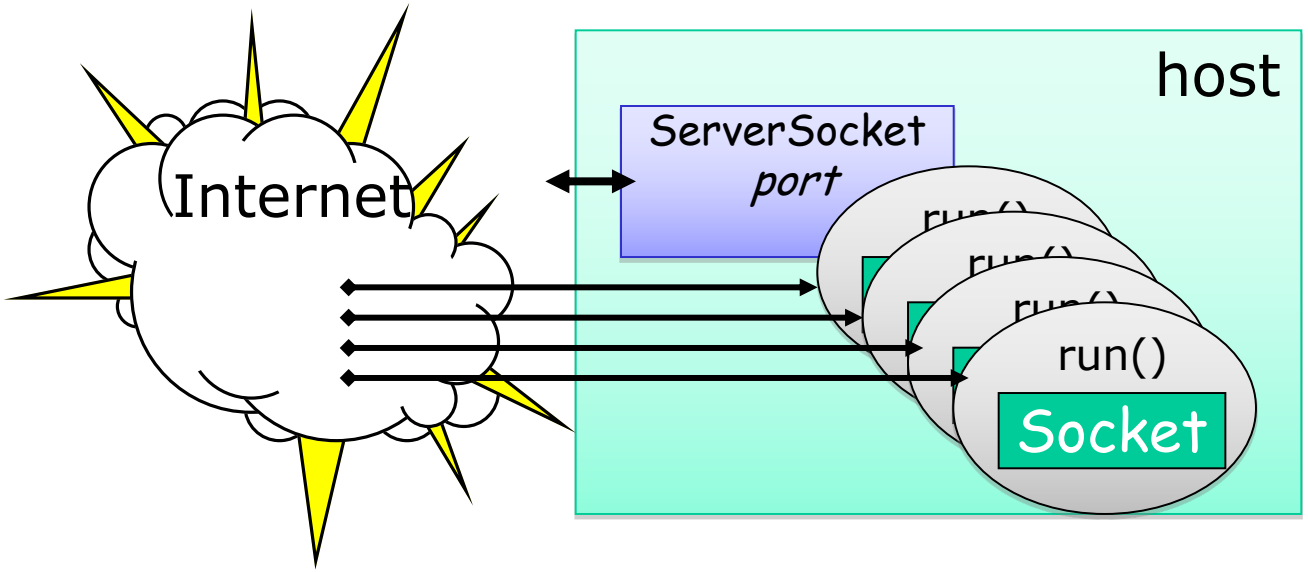
```
InputStream   in  = conn.getInputStream();
OutputStream out= conn.getOutputStream();
```

...o utilizzare le classi Reader/Writer

```
writer  = new OutputStreamWriter(out);
 reader =  new InputStreamReader(in);
```

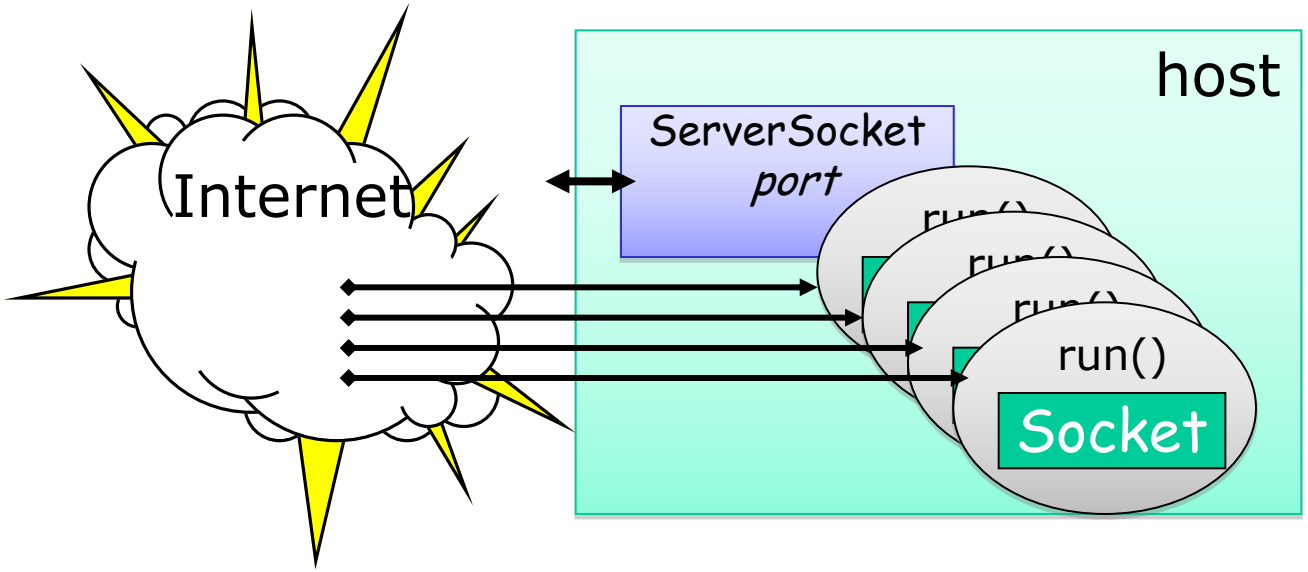
Server Multithread

Permettono di migliorare l' efficienza creando uno o più thread per connessione.



Server Multithread

Permettono di migliorare l' efficienza creando uno o più thread per connessione.



- Un Thread Worker che preveda un costruttore che accetti come parametro una socket;
- Nel run() del worker è contenuto tutto il codice per il soddisfacimento della richiesta.

Worker Thread

```
// Schema generale WORKER


   public class MyWorker extends Thread{
     private Socket socket;…

     MyWorker (Socket skt){socket=skt;…}

     public void run(){
       InputStream in=socket.getInputStream();
      OutputStream ot= socket.getOutputStream();
       …//elabora richiesta
     }
   }
```

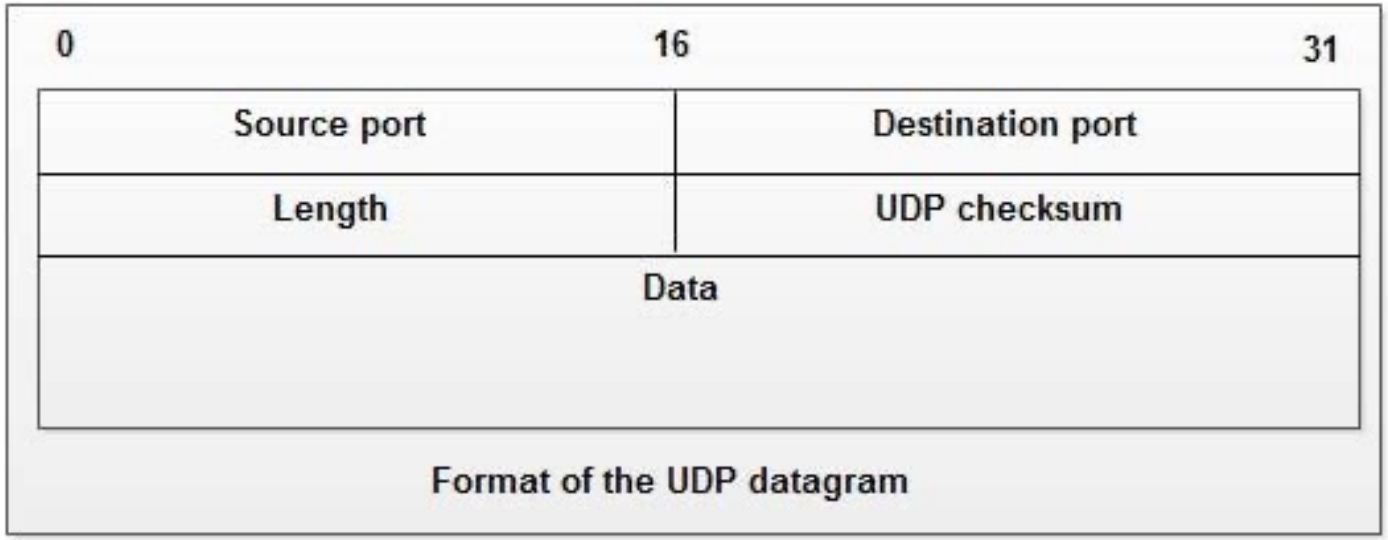
Server

```
// Schema generale Server
   …
   while (…) {
     Socket client= server.accept();
     MyWorker myWorker=new MyWorker(client);
     myWorker.start();
   }
   …
```

Socket UDP/IP (1/3)

- Nel package **java.net** sono fornite le classi per la comunicazione basata su **UDP**
- Le classi principali sono:
  - **DatagramPacket:** astrazione del **pacchetto UDP** (datagramma);
  - **DatagramSocket:** astrazione della **socket UDP**

- La struttura dati incapsulata in un' istanza di DatagramPacket consiste di due parti:
  - **address**;
  - **data**.



Socket UDP/IP (2/3)

Gli oggetti di tipo DatagramSocket:

- **Tale classe è utilizzata per inviare e ricevere pacchetti**
  - UDP è ***connectionless*** quindi si può utilizzare la stessa socket per inviare pacchetti a destinazioni diverse

    DatagramSocket () throws SocketException

    DatagramSocket (int port) throws SocketException

- **La trasmissione e ricezione avvengono mediante i seguenti metodi:**
  - `void send(DatagramPacket packet) throws IOException`
  - `void receive(DatagramPacket packet) throws IOException`

Socket UDP/IP (3/3)

Ricezione di un datagramma

```
DatagramSocket socket= new DatagramSocket(port);
byte buffer[]= new byte[65508];
//65508 e' la max dimensione di un pacchetto UDP
DatagramPacket pkt=new DatagramPacket(buffer, buffer.length);
//Se forniamo una dimensione del buffer troppo piccola
//la dimensione eccedente del pacchetto andrà persa
socket.receive(pkt); //chiamata bloccante
InetAddress fromAddr=pkt.getAddress();
int fromPort= pkt.getPort();
int length=pkt.getLength();
byte[] data= pkt.getData();
```

- **Buffer è un array di byte che conterrà il contenuto del pacchetto all' arrivo. L' array è pre-allocato**
- **Length massimo numero di byte che sono letti nel buffer (dati extra nel pacchetto UDP saranno scartati)**

Socket UDP/IP (3/3)

Ricezione di un datagramma

```
DatagramSocket socket= new DatagramSocket(port);
byte buffer[]= new byte[65508];
//65508 e' la max dimensione di un pacchetto UDP
DatagramPacket pkt=new DatagramPacket(buffer, buffer.length);
//Se forniamo una dimensione del buffer troppo piccola
//la dimensione eccedente del pacchetto andrà persa
socket.receive(pkt); //chiamata bloccante
InetAddress fromAddr=pkt.getAddress();
int fromPort= pkt.getPort();
int length=pkt.getLength();
byte[] data= pkt.getData();
```

Invio di un datagramma

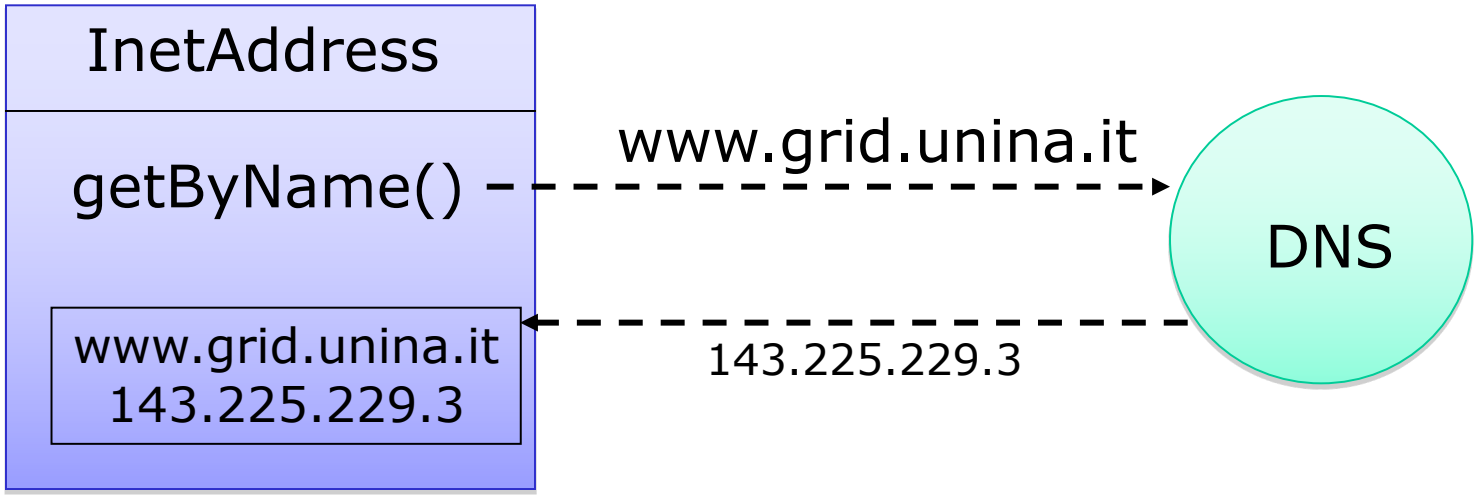
```
DatagramSocket socket= new DatagramSocket();
..
//Si costruisce il pacchetto UDP da inviare
DatagramPacket pkt=new DatagramPacket(data, data.length,
   InetAddress.getByName("pippo.pluto.it"), portNum);

//Si invia il pacchetto
socket.send(pkt);
…
```

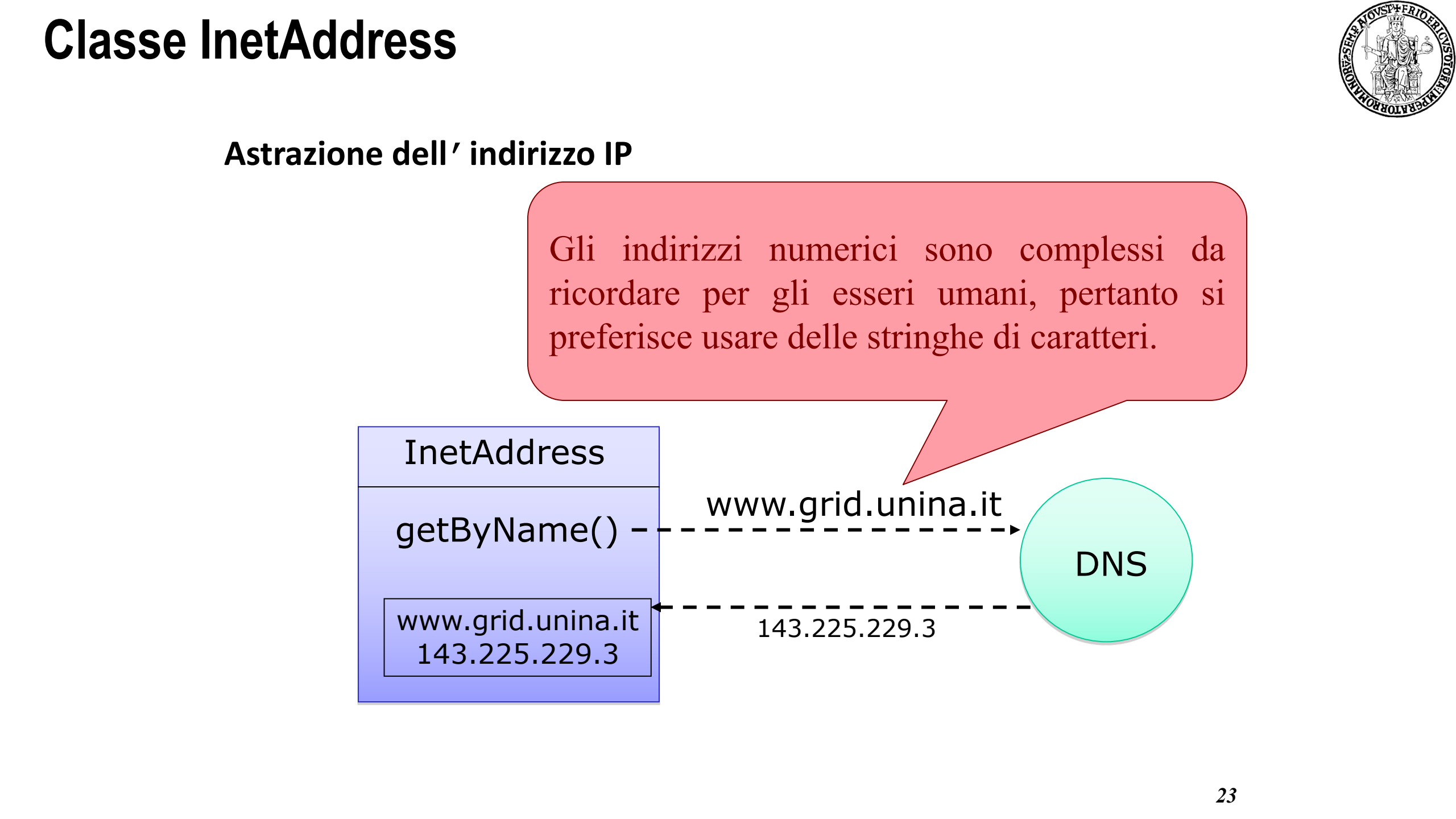
Classe InetAddress in JAVA

Astrazione dell' indirizzo IP

Java offre una classe per l' astrazione degli indirizzi



Classe InetAddress



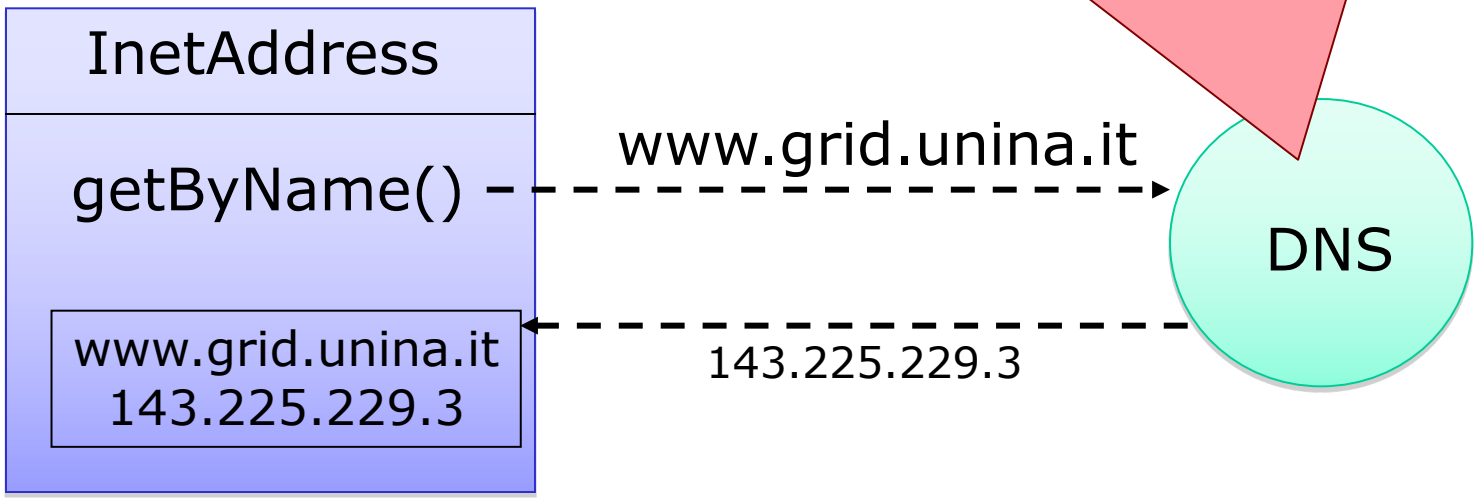
Astrazione dell' indirizzo IP

Gli indirizzi numerici sono complessi da ricordare per gli esseri umani, pertanto si preferisce usare delle stringhe di caratteri.

Classe InetAddress

Astrazione dell' indirizzo IP

Tali stringhe sono convertite nell'indirizzo IP associato per mezzo dell'interrogazione a un Domain Name System (spesso indicato con DNS).



Classe InetAddress

Alcuni metodi di utilità

- **InetAddress getLocalHost() throws UnknownHostException :** restituisce l'indirizzo IP della macchina locale;
- **InetAddress getByName(String host) throws UnknownHostException :** restituisce l'indirizzo IP associato a un dato host name;
- **Inetaddress[] getAllByName(String host) throws UnknownHostException :** un array di indirizzi IP per un dato host name;
- **byte[] getAddress() :** restituisce l'indirizzo, contenuto nell'oggetto, come seguenza di byte;
- **String getHostName() :** restituisce il nome della macchina su cui sta girando il programma
- **boolean isMulticastAddress() :** restituisce vero se l'oggetto incapsula un indirizzo multicast.

Classe InetAddress

Le eccezioni

- **`UnknownHostException`**

  l'host non può essere correttamente identificato;

- **`SecurityException`**

  viene sollevato se il SecurityManager non permette l'esecuzione di una specifica operazione.