

Il Linguaggio Java: Socket

Il Linguaggio Java: Socket.

Walter Cazzola

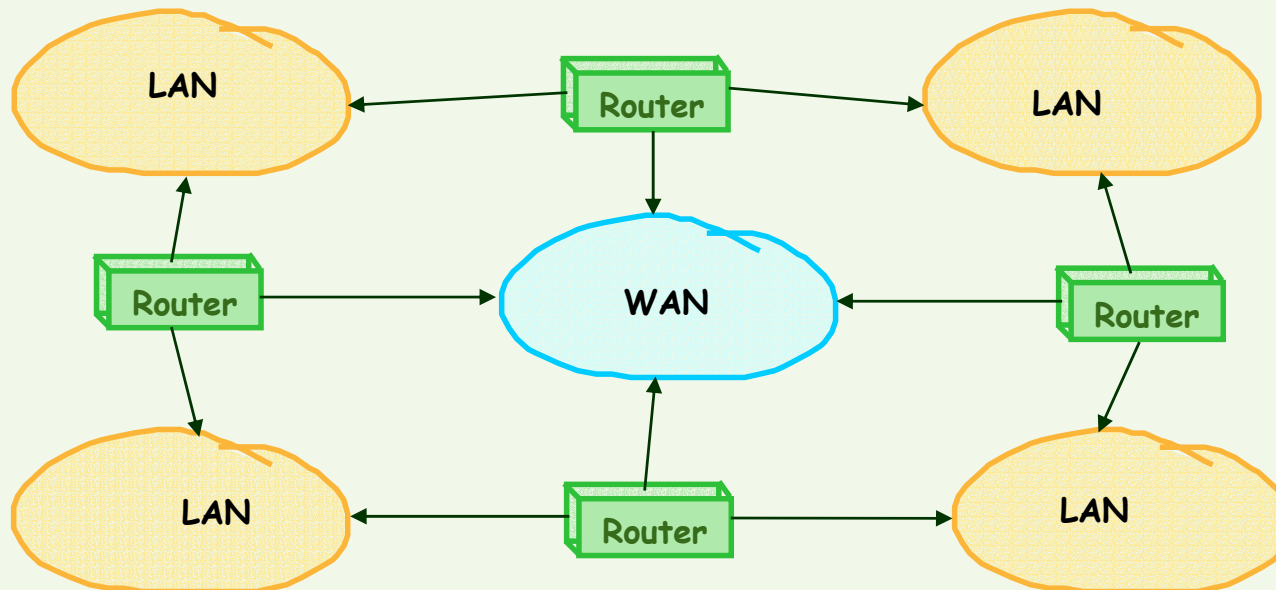
Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano.

e-mail: cazzola@disi.unige.it

Internet

Con il concetto di internet si indica una collezione di una o più reti connesse da router.

La rete che conosciamo come Internet non è altro che un esempio del concetto stesso di internet.



Protocolli di Rete

Un protocollo è un insieme di regole che governano le comunicazioni.

Esempi:

- **SMTP**: Simple Mail Transfer Protocol
- **HTTP**: HyperText Transfer Protocol
- **FTP**: File Transfer Protocol

Il protocollo HTTP utilizza un "Uniform Resource Locator (URL)" per specificare un indirizzo Internet:

```
METODO://NOME-HOST/PERCORSO  
HTTP://www.disi.unige.it/person/CazzolaW/index.html
```

Protocolli di Rete: Client/Server

Un'applicazione client/server spezza un task fra due processi (spesso su computer diversi): il client ed il server.



Protocollo Client/Server:

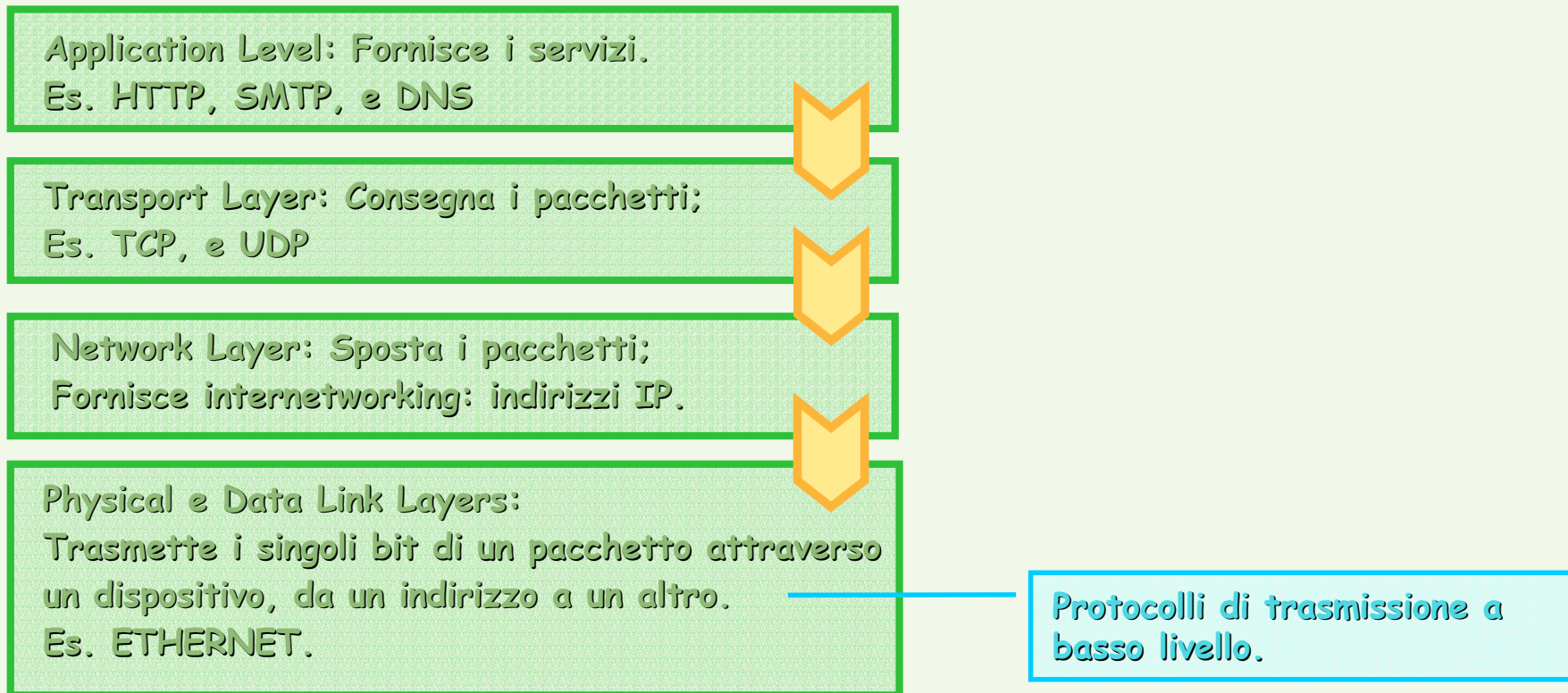
Server: mette a disposizione un servizio su un particolare host.

Client: contatta il server e richiede il servizio.

Server: accetta e soddisfa una richiesta da parte di un client.

Protocolli di Rete: ISO/OSI

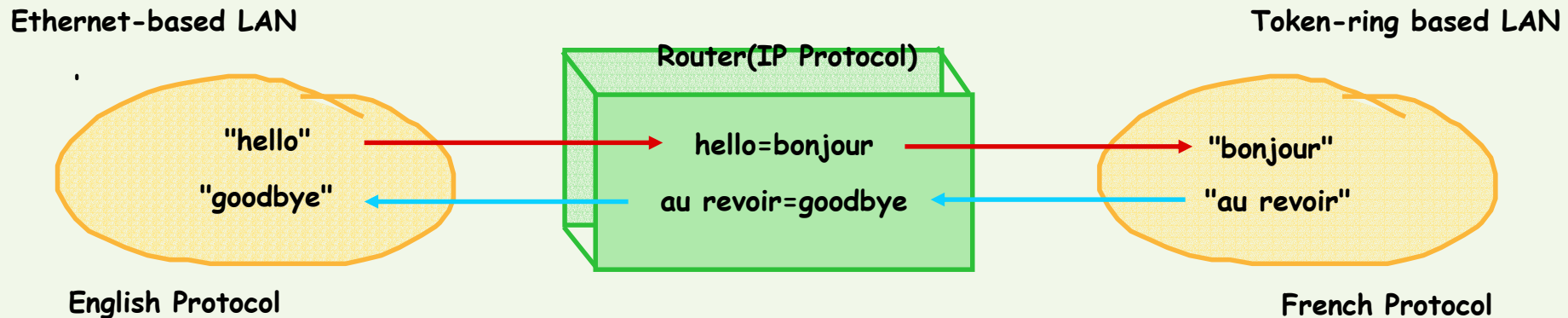
Una rete è divisa in diversi strati, ognuno dei quali gestisce uno specifico protocollo. I messaggi scambiati sulla rete attraversano ogni singolo strato:



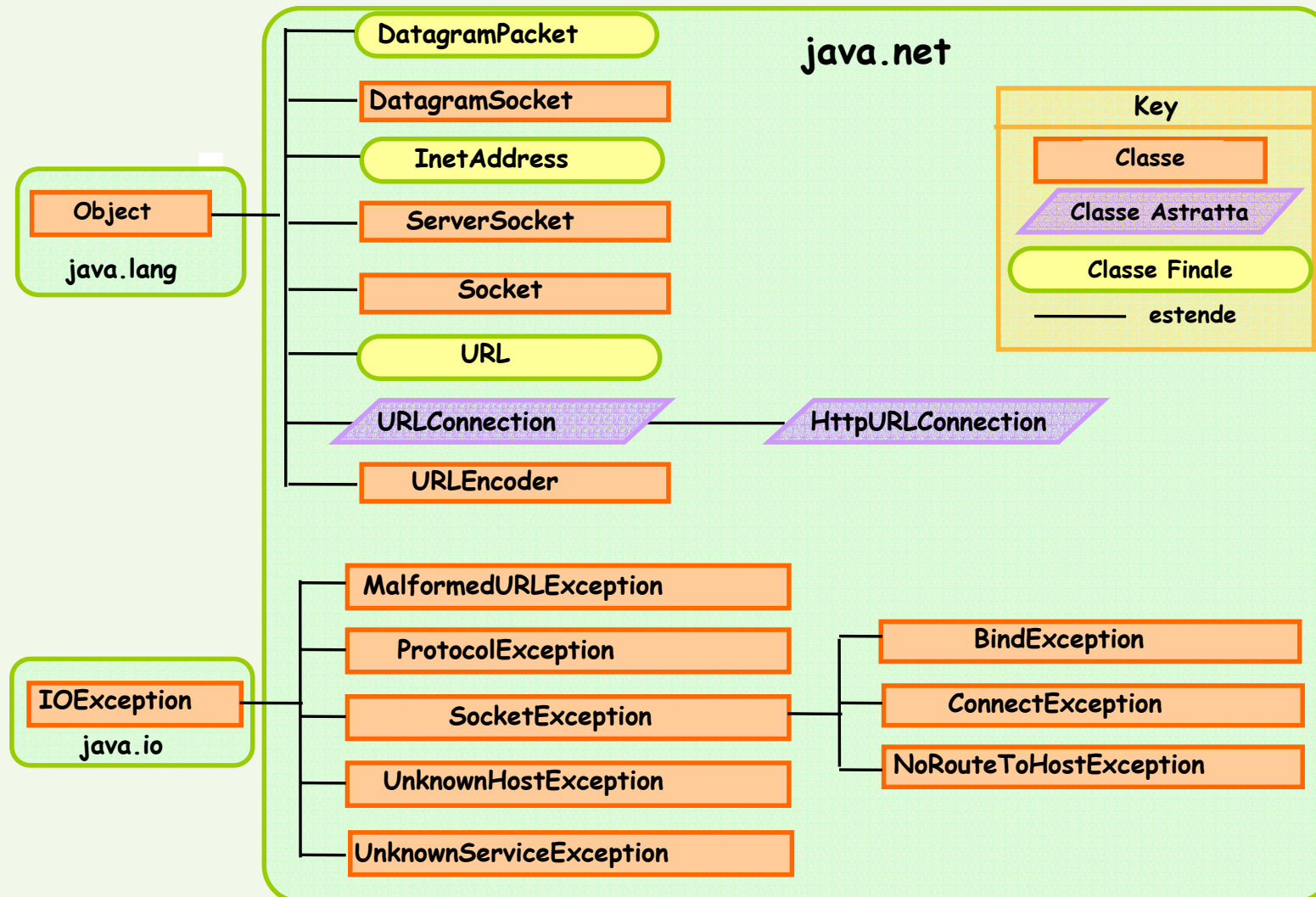
Il Protocollo di Internetworking (IP)

Il protocollo di internetworking (IP) traduce un protocollo di rete in un altro.

IP rende possibile la comunicazione tra reti diverse:



Il Package java.net




Java Library: java.net.URL

Metodi per collegarsi ad un sito web:

```
public final class URL extends Object {  
    // Costruttore  
    public URL( String urlSpec ) throws MalformedURLException;  
    // metodi di istanza pubblici  
    public URLConnection openConnection() throws IOException;  
    public final InputStream openStream() throws IOException;  
}
```

Esempio:

```
URL url;  
try {  
    url = new URL("http://www.disi.unige.it/person/CazzolaW/index.html");  
} catch (MalformedURLException e) {  
    System.out.println("Malformed URL: " + url.toString());  
}
```



URL della risorsa.

Applet: Scaricare dalla Rete

Metodi per scaricare risorse (audio/video) dalla rete:

```
public class Applet extends Panel {  
    public AppletContext getAppletContext();  
    public AudioClip getAudioClip(URL url);  
    public Image getImage(URL url);  
    public void play(URL url);  
    public void showStatus(String msg);  
}
```

```
URL url;  
try {  
    url = new URL("http://www.disi.unige.it/person/CazzolaW/sound.au");  
    play(url);  
    url = new URL("http://www.disi.unige.it/person/CazzolaW/demo.gif") ;  
    imgRef = getImage(url);  
} catch (MalformedURLException e) {  
    System.out.println("Malformed URL: "+url.toString());  
}
```


Scarica e suona un file audio.

Scarica e memorizza un immagine.

Es.: Scaricare e Visualizzare un File

Leggere il contenuto di un file remoto è, grosso modo, come leggere un file locale.

```
public class Download {  
    private URL url; // l'URL del file che si vuole scaricare.  
    private BufferedReader data; // il Reader che useremo per accedere al file.  
    public Download(String _url) throws MalformedURLException {  
        url = new URL(_url);  
    }  
    public void readText() throws IOException {  
        data = new BufferedReader(new InputStreamReader(url.openStream()));  
        String line = data.readLine();  
        while (line != null) { // legge ogni singola riga del file  
            System.out.println(line); // e la visualizza sul terminale  
            line = data.readLine();  
        }  
        data.close();  
    } // readTextIntoDisplay()  
}
```



BufferedReader.readLine()
ritorna null alla fine del file.

Es.: Scaricare e Visualizzare un File

Metodo main():

```
public static void main(String argv[]) {  
    try {  
        Download d = new Download("http://www.disi.unige.it/person/CazzolaW/index.html");  
        d.readText();    // scarica e visualizza la pagina HTML  
    } catch (MalformedURLException e) {    // potrebbe essere lanciato da URL()  
        System.out.println("URL: "+e.getMessage());  
    } catch( IOException e ) {    // potrebbe essere lanciato da read() o close()  
        System.out.println("IO: "+e.getMessage());  
    }  
}
```

Comunicazioni Client/Server Via Sockets

Un socket è un canale di comunicazione bidirezionale.

Sono state introdotte in Unix nel 1981.

La gestione (creazione, uso e rilascio) delle socket è interamente a carico dell'applicazione.

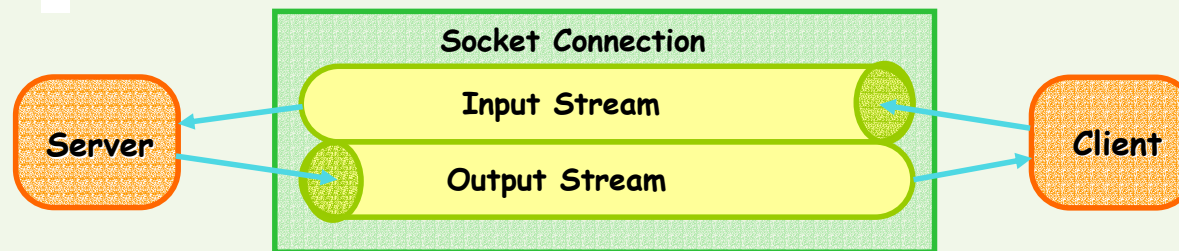
Paradigma Client/Server.

Il server crea un socket su una porta specifica ed attende che il client richieda una connessione.

Due tipi di trasporto via socket: UDP (veloce ma inaffidabile) e TCP.

Socket

Ogni socket ha due flussi, uno per l'input e l'altro per l'output.



Analogia: chiamata telefonica.

- **Server:** attende che il telefono suoni allora inizia il servizio.
- **Client:** chiama il numero telefonico (URL), si connette e richiede un servizio.

Socket Programming con TCP

Affinché il cliente possa collegarsi al server:

- il server deve essere già in esecuzione; e
- DEVE aver creato il socket che dà il benvenuto al client.

Connessione del cliente:

- si crea un TCP socket locale al cliente;
- specificando l'IP del server e la porta su cui sta attendendo la connessione.
- il client TCP crea una connessione al server TCP.

Lato server:

- Il server TCP crea il socket per permettere la comunicazione;

Nota. TCP offre un trasferimento ordinato ed affidabile dei byte in un canale fra client e server.

Socket: Protocollo Base dei Server

Pseudocodice:

1. Creare un `SocketServer` e scegliere una porta.
2. Mettersi in ascolto sulla porta e accettare una connessione da un client.
3. Conversare col client.
4. Chiudere il socket.

Java:

```
Socket connectionSocket;           // riferimento al socket
ServerSocket welcomeSocket;        // la porta sulla quale il server sarà in attesa
try {
    welcomeSocket = new ServerSocket(10001); // creare una porta
    connectionSocket = welcomeSocket.accept(); // attende richiesta connessione

    // comunicazione con il client

    connectionSocket.close();           // chiudere il socket
} catch (IOException e) {
    e.printStackTrace();
}
```

Su.thor.disi.unige.it

Socket: Protocollo Base dei Client

Pseudocodice:

1. Apre una connessione verso il server specificato.
2. Conversa con il server.
3. Chiude la connessione.

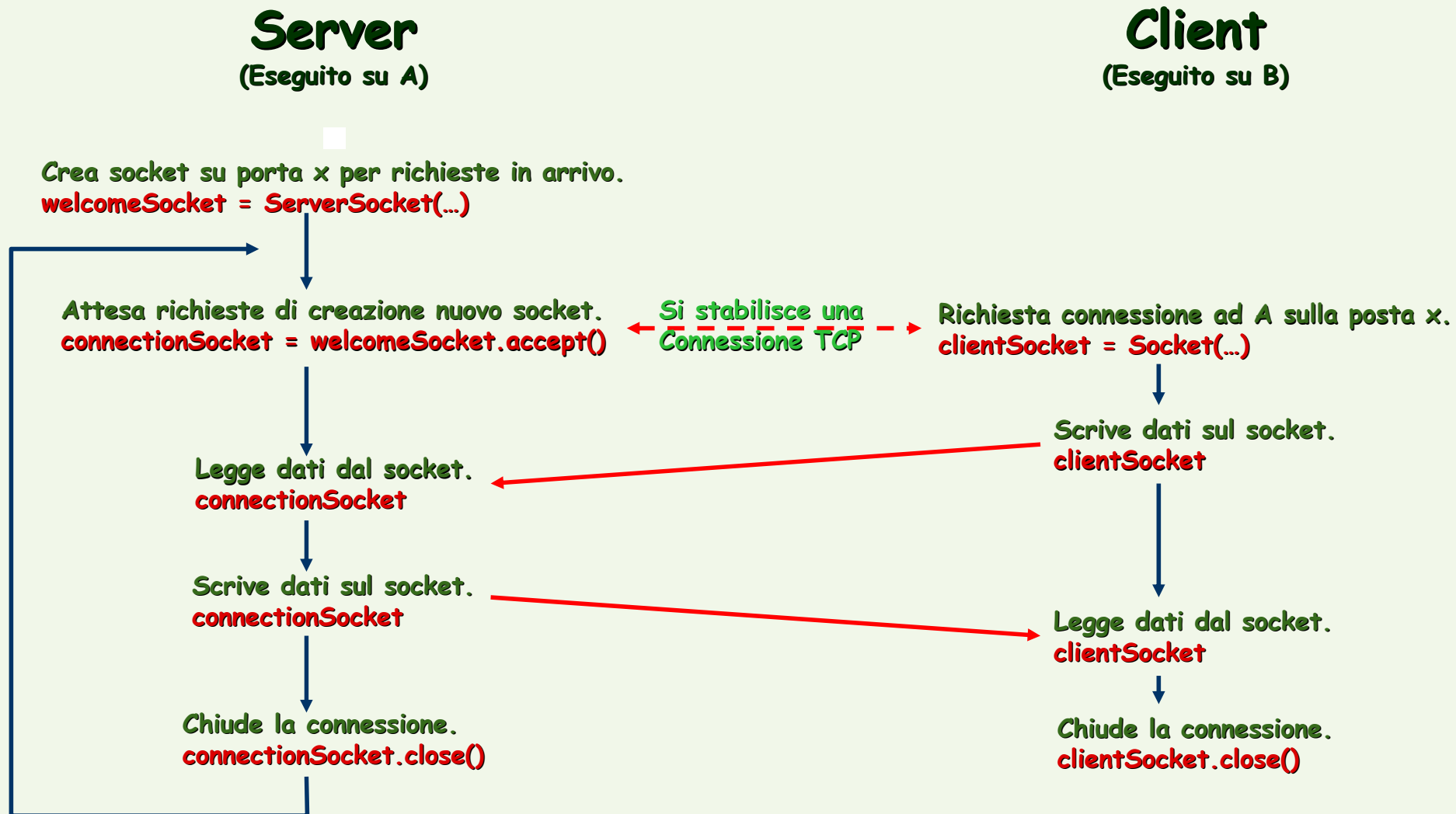
Java:

```
Socket clientSocket;      // riferimento al socket
try {
    clientSocket = new Socket("thor.disi.unige.it", 10001); // connessione

    // effettua la comunicazione con il server

    clientSocket.close();      // chiude il socket
} catch (IOException e ) {
    e.printStackTrace();
}
```


Interazione Client/Server (TCP)



Socket: Scrivere in un Socket

Un metodo per scrivere su un socket (usabile sia da client che da server):

```
protected void writeToSocket(Socket sock, String str)
    throws IOException {
    OutputStream oStream = sock.getOutputStream();
    for (int k = 0; k < str.length() ; k++)
        oStream.write(str.charAt(k));
} // writeToSocket()
```

Ottiene lo Stream di Output.

Scrive sullo Stream di Output.

Un socket crea automaticamente i propri stream.

Nota: non chiudere lo stream del socket dopo un'operazione di I/O a meno che il socket non sia più necessario.

Socket: Leggere da un Socket

Analogamente a quanto fatto per scrivere su un socket:

```
protected String readFromSocket(Socket sock)
    throws IOException {
    InputStream iStream = sock.getInputStream();
    String str="";
    char c;
    while ( ( c = (char) iStream.read() ) != '\n')
        str = str + c + "";
    return str;
}
```

Ottiene lo Stream di Input.

legge un byte e lo
converte in char.

Protocollo: se il client scrive dei byte, il server DEVE leggere tali byte e viceversa.

Esempio: Echo Client/Server

Scrivere un'applicazione client/server il cui comportamento è:

input da tastiera.

Client Side:

```
CLIENT: connected to 'thor.disi.unige.it'  
SERVER: Hello, how may I help you?  
CLIENT: type a line or 'goodbye' to quit  
INPUT: hello  
SERVER: You said 'hello'  
INPUT: this is fun  
SERVER: You said 'this is fun'  
INPUT: goodbye  
SERVER: Goodbye  
CLIENT: connection closed
```

Server Side:

```
Echo server at thor.disi.unige.it waiting for connections  
Accepted a connection from thor.disi.unige.it  
Closed the connection
```

```
Accepted a connection from thor.disi.unige.it  
Closed the connection
```


Esempio: Echo Client/Server (cont.)

Si usa una classe comune che fornisca le operazioni di lettura e scrittura sul file e nasconda la natura duale dei socket.

Classe Comune

```
public class ClientServer extends Thread {  
    protected InputStream iStream;           // i due flussi legati al socket  
    protected OutputStream oStream;  
  
    protected String readFromSocket(Socket sock) throws IOException {  
        iStream = sock.getInputStream();  
        String str=""; char c;  
        while ( ( c = (char) iStream.read() ) != '\n') str = str + c + "";  
        return str;  
    } // readFromSocket()  
  
    protected void writeToSocket(Socket sock, String str) throws IOException {  
        oStream = sock.getOutputStream();  
        if (str.charAt( str.length() - 1 ) != '\n') str = str + '\n';  
        for (int k = 0; k < str.length() ; k++) oStream.write(str.charAt(k));  
    } // writeToSocket()  
} // ClientServer
```


Esempio: Echo Client/Server (cont.)

Classe Server

```
public class EchoServer extends ClientServer {
    private ServerSocket port;
    private Socket socket;

    public EchoServer(int portNum, int nBacklog) {
        port = new ServerSocket(portNum, nBacklog);
    }

    public void run() {
        while(true) {
            socket = port.accept();
            System.out.println("Accepted a connection from " + socket.getInetAddress());
            provideAService(socket);
            socket.close();
            System.out.println("Closed the connection\n");
        }
    } // run()

    protected void provideAService(Socket socket) {
        writeToSocket(socket, "Hello, how may I help you?\n");
        do {
            str = readFromSocket(socket);
            if (str.toLowerCase().equals("goodbye")) writeToSocket(socket, "Goodbye\n");
            else writeToSocket(socket, "You said '" + str + "'\n");
        } while (!str.toLowerCase().equals("goodbye"));
    } // provideAService()
```


Esempio: Echo Client/Server (cont.)

```
public class EchoClient extends ClientServer {
    protected Socket socket;
    private BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

    public EchoClient(String url, int port) {
        socket = new Socket(url, port);
        System.out.println("CLIENT: connected to " + url + ":" + port);
    } // EchoClient()

    public void run() {
        requestAService(socket);
        socket.close();
    } // run()

    protected void requestAService(Socket socket) {
        String servStr = readFromSocket(socket);
        if (servStr.substring(0,5).equals("Hello")) {
            String userStr = "";
            do {
                userStr = in.readLine();
                writeToSocket(socket, userStr + "\n");
                servStr = readFromSocket(socket);
                System.out.println("SERVER: " + servStr);
            } while (!userStr.toLowerCase().equals("goodbye"));
        }
    } // requestAService()
}
```

Classe Client

// effettua l'handshaking

// legge l'input
// lo scrive sul socket
// legge la risposta del server
// e la visualizza
// finché non riceve un 'goodbye'

Socket Programming con UDP

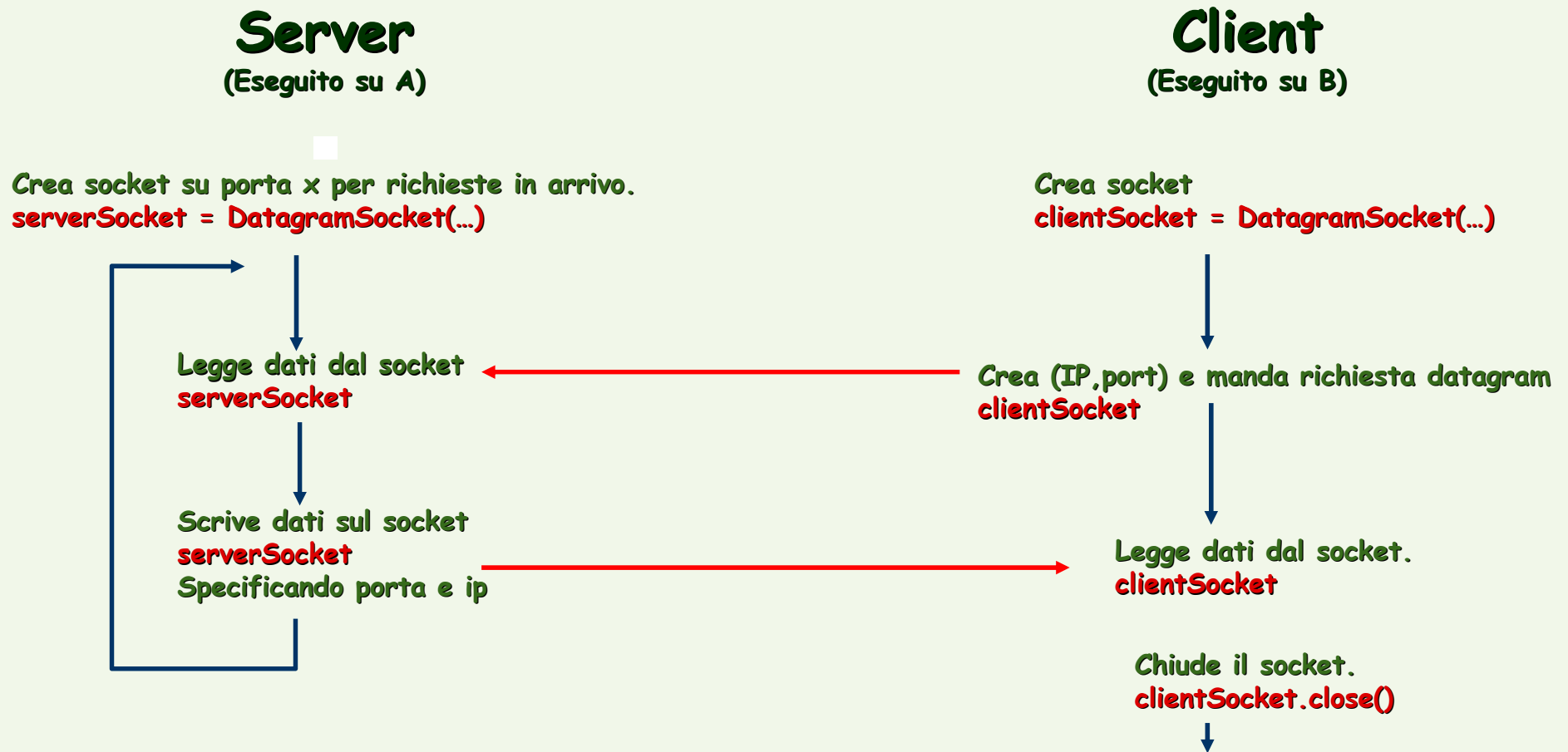
UDP offre un trasferimento inaffidabile di gruppi di byte (datagram) fra client e server.

UDP: nessuna "connessione" fra client e server:

- nessun handshaking;
- il client attacca esplicitamente ad datagram l'indirizzo IP e la porta di destinazione;
- il server deve estrarre l'indirizzo IP e la porta del client dal datagram ricevuto;

UDP: i dati trasmessi possono essere persi o ricevuti nell'ordine sbagliato.

Interazione Client/Server (UDP)



InetAddress

Classe che incapsula gli indirizzi IP (supporta sia nomi simbolici che indirizzi numerici).

- **getLocalHost()**: rende un oggetto InetAddress rappresentante l'IP del localhost;
- **getByName()**: rende un oggetto InetAddress relativo all'IP dell'host specificato come argomento.

DatagramSocket

Classe che implementa i socket, sia client che server, con protocollo UDP;

- **send():** spedisce datagram (oggetti della classe **DatagramPacket**);
- **receive():** riceve datagram;
- **getLocalPort():** ritorna la porta su cui è aperto il socket.
- **getLocalAddress():** ritorna l'indirizzo del localhost.

DatagramPacket

Classe che implementa i datagram UDP spediti.

- costruttori:
 - per datagram spediti; e
 - per datagram ricevuti.
- metodi:
 - **getData()**: ritorna i dati contenuti nel datagram;
 - **getPort()**: ritorna la porta di destinazione del datagram;
 - **getSocketAddress()**: Ritorna l'indirizzo di destinazione del datagram.

Esempio: Java Client (UDP)

```
class UDPClient {  
    public static void main(String args[]) throws Exception {  
        ...  
        DatagramSocket clientSocket = new DatagramSocket();           // crea client socket  
        InetAddress IPAddress = InetAddress.getByName("hostname");     // recupera IP  
        byte[] sendData = new byte[1024];  
        byte[] receiveData = new byte[1024];  
        ...  
        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);  
        clientSocket.send(sendPacket);                                  // spedisce datagram  
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);  
        clientSocket.receive(receivePacket);                           // legge il datagram dal server  
        String modifiedSentence = new String(receivePacket.getData());  
        ...  
        clientSocket.close();  
    }  
}
```


Esempio Java Server (UDP)

```
class UDPServer {  
    public static void main(String args[]) throws Exception {  
        DatagramSocket serverSocket = new DatagramSocket(9876); // crea datagram socket alla porta 9876  
        byte[] receiveData = new byte[1024];  
        byte[] sendData = new byte[1024];  
  
        while(true) {  
            // crea spazio per il datagram che dovrà ricevere  
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);  
            serverSocket.receive(receivePacket); // riceve datagram  
  
            InetAddress IPAddress = receivePacket.getAddress(); // recupera IP e porta del mittente  
            int port = receivePacket.getPort();  
  
            ...  
            sendData = elaboratedSentence.getBytes(); // costruisce il datagram da ritornare  
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, port);  
            serverSocket.send(sendPacket); // scrive datagram nel socket.  
        }  
    }  
}
```