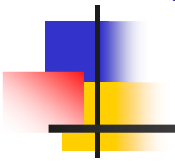
Linguaggi

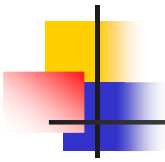
Corso M-Z - Laurea in Ingegneria Informatica
A.A. 2007-2008

Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it



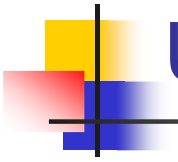
- lezione 15 -

Socket in Java



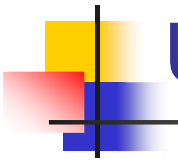
Java in Rete

- Java permette l'accesso alla rete tramite due package:
  - java.net (Socket ServerSocket)
  - java.rmi (uso di oggetti remoti)



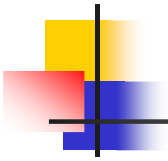
URL e Connessioni

- La classe URL cattura il concetto di indirizzo Internet (URL) nella forma standard:
  - http://localhost/index.html
  - file:///autoexec.bat
- Un oggetto URL si crea a partire dall'indirizzo che rappresenta:
  URL url = new URL("....");
- e si usa per aprire una connessione verso tale indirizzo.
- Per aprire una connessione, si invoca sull'oggetto URL il metodo openConnection():
  URLConnection c = url.openConnection();
- Il risultato è un oggetto URLConnection, che rappresenta una "connessione aperta"
- in pratica, così facendo si è stabilito un canale di comunicazione verso l'indirizzo richiesto



URL e Connessioni

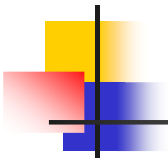
- Il fatto di avere aperto la connessione non significa che ci si è veramente connessi. Per connettersi tramite tale connessione:
  c.connect();
- Per comunicare si recuperano dalla connessione i due stream (di ingresso e di uscita) a essa associati, tramite i metodi:
  public InputStream getInputStream()
- restituisce lo stream di input da cui leggere i dati (byte) che giungono dall'altra parte
  public OutputStream getOutputStream()
- restituisce lo stream di output su cui scrivere i dati (byte) da inviare all'altra parte
- su questi stream si legge / scrive come su qualunque altro stream di byte.



URL e Connessioni

- ESEMPIO: Connettersi all'URL dato e, nell'ipotesi che esso invii testo, visualizzarne il contenuto

```
import java.io.*; import java.net.*;
class EsempioURL {
  public static void main(String args[]){
  URL u = null;
  try
      {u = new URL(args[0]); //indirizzo di URL passato come argomento}
  catch (MalformedURLException e)
          {System.err.println("URL errato: " + u);}
  URLConnection c = null;
  try {   System.out.print("Connecting...");
          c = u.openConnection(); c.connect();
          System.out.println("..OK");
```

URL e Connessioni

- ESEMPIO: Connettersi all'URL dato e, nell'ipotesi che esso invii testo, visualizzarne il contenuto

```
…
  InputStreamReader is = new InputStreamReader(c.getInputStream());
  BufferedReader r = new BufferedReader(is);
          System.out.println("Reading data...");
          String line = null;
          while((line=r.readLine())!=null)
          System.out.println(line);
          } catch (IOException e) {System.err.println(e);
} } }
```

URL e Connessioni

- Esempio invocazione e output programma precedente:

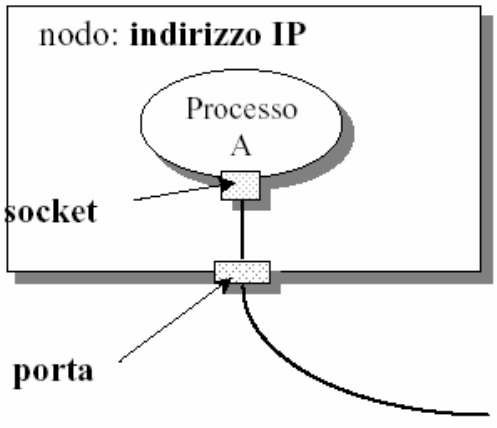
```
D:|esercizi>java EsempioURL file:///P.html
Connecting.....OK
Reading data...

<TITLE> Esempio di form </TITLE>
<H1> Esempio di form </H1>
<FORM                                    METHOD="POST"
   ACTION="http://localhost/cgi/prova.exe">
Inserisci il testo: <INPUT NAME="testo">
e poi premi invio: <INPUT TYPE="submit" VALUE="invio">
</FORM>
```

- NOTA: Se avessimo la possibilitá di trasformare in forma grafica quello che il programma legge dall'URL avremmo praticamente realizzato un Browser Web.

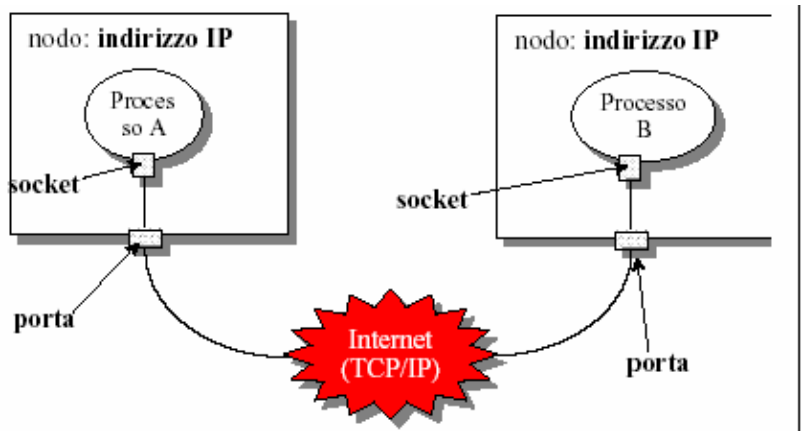
Socket

- Un socket è concettualmente una porta
- Collega un certo processo al mondo esterno
- E' identificata da un numero (port number) unico su una data macchina (nodo o host)
- Ogni nodo e' identificato dal suo indirizzo IP

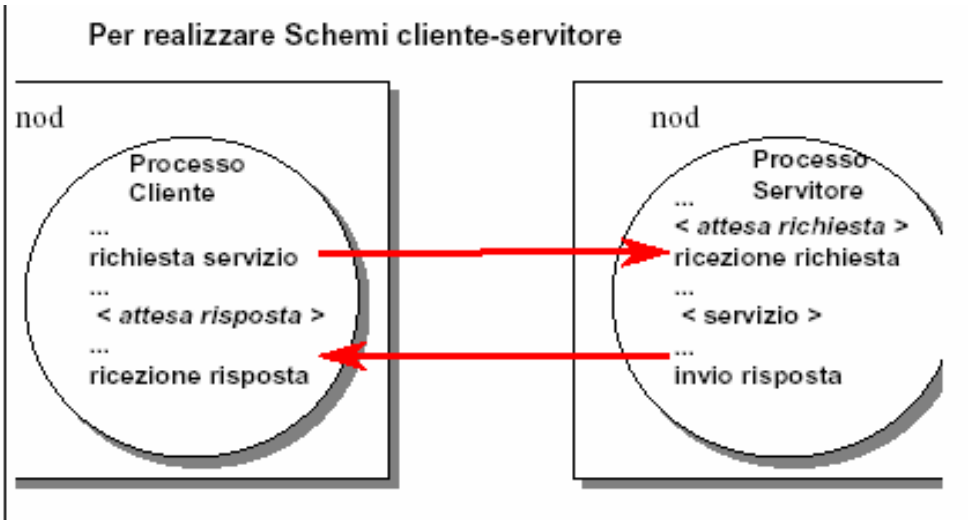


Socket

- Il socket è un canale di comunicazione
- Permette a due processi, residenti sulla stessa macchina o anche molto distanti, di comunicare fra loro nello stesso modo
- Modello cliente / servitore:
  - il servitore deve stare in attesa di possibili comunicazioni in arrivo
  - i clienti (anche più di uno) parlano con il servitore (enti attivi)...



Socket



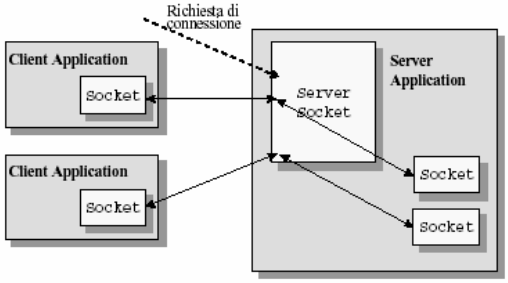
Socket

- Esistono fondamentalmente due tipi di socket: socket stream e socket datagram
- **Le socket stream**
  - sono affidabili, stabiliscono una connessione stabile e bidirezionale con l'altra parte, che dura finché non si decide di chiuderla
  - Usare quando l'ordine dei messaggi è importante e l'affidabilità è cruciale
  - Spesso c'è un limite massimo alle connessioni che si possono aprire
- **Le socket datagram**
  - non sono affidabili, non stabiliscono una connessione stabile: la comunicazione è unidirezionale come un telegramma ma sono meno costose
  - Usare quando le prestazioni sono fondamentali e/o ci vorrebbero troppe connessioni aperte insieme
  - Non devono esserci problemi se i messaggi arrivano in ordine qualunque

Socket stream

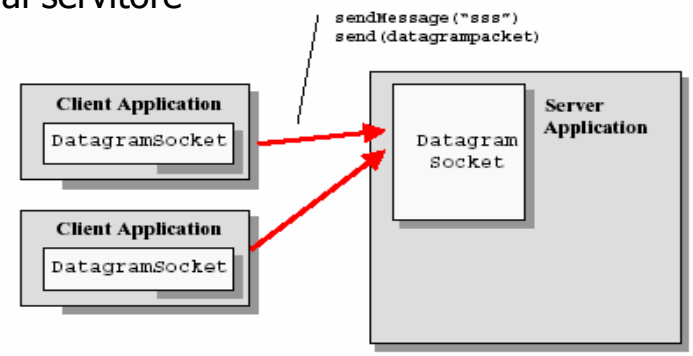
- Il servitore crea un ServerSocket con un numero e si mette in attesa
- Un cliente, quando vuole comunicare col servitore, crea la sua Socket specificando con chi vuole parlare (nome dell'host, numero di porta)
- Il servitore accetta la richiesta del cliente: con ciò si crea una Socket già collegata al cliente, tramite cui i due comunicano.
- Al Socket sono associati due stream, uno dal cliente verso il servitore ed uno dal servitore verso il cliente
- La comunicazione cliente/servitore è bidirezionale: i ruoli "cliente" e "servitore" sono tali solo nella fase iniziale, quando si instaura la connessione; una volta connessi, i due processi si parlano alla pari

- Il ServerSocket serve per stare in attesa di richieste dai clienti: quando ne arriva una, l'effettiva comunicazione avviene tramite un nuovo Socket appositamente creato



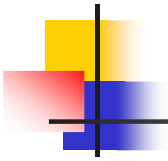
Socket datagram

- Il servitore crea la sua DatagramSocket con un numero noto, e si mette in attesa
- Un cliente crea la sua DatagramSocket
- Quando vuole inviare un messaggio al servitore, il cliente gli manda un "data-gramma" specificando nome dell'host e numero di porta
- non si crea alcuno stream stabile
- comunicazione solo dal cliente al servitore



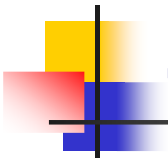
La classe Socket

- Costruire una Socket significa aprire la comunicazione verso l'altra parte
  public Socket(InetAddress remoteAddr, int remotePort)
- Crea una socket stream e la collega alla porta specificata della macchina remota corrispondente all'indirizzo IP dato
  public Socket(String remoteHost, int remotePort)
- Crea una socket stream e la collega alla porta specificata della macchina remota corrispondente al nome dato
- Esempio
  Socket s = new Socket("mypc.unibo.it",13);
- Alcuni metodi utili:
  public InetAddress getInetAddress()
- restituisce l'indirizzo della macchina remota a cui la socket è connessa
  public InetAddress getLocalAddress()
- restituisce l'indirizzo della macchina locale
  public int getPort()
- restituisce il numero di porta sulla macchina remota a cui la socket è connessa
  public int getLocalPort()
- restituisce il numero di porta su localhost a cui la socket è legata



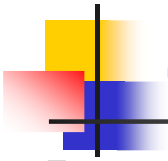
La classe Socket

- Per comunicare, si recuperano dalla socket i due stream (di ingresso e di uscita) a essa associati, tramite i metodi:
  public InputStream getInputStream()
- restituisce lo stream di input da cui leggere i dati (byte) che giungono dall'altra parte
  public OutputStream getOutputStream()
- restituisce lo stream di output su cui scrivere i dati (byte) da inviare all'altra parte
- Poi, su questi stream si legge / scrive come su qualunque altro stream di byte. Al termine, per chiudere la comunicazione si chiude il Socket:
  public synchronized void close()
- chiude la connessione e libera la risorsa



Servizi Standard

- Moltissimi servizi di uso comune sono standardizzati e disponibili su macchine sia Unix sia (non sempre) Windows
- echo (porta 7): rimanda indietro tutto quello che gli si invia, come un'eco
- daytime (porta 13): restituisce data e ora
- telnet (porta 23): consente il collegamento remoto, da altro terminale (solo Unix)
- smtp (porta 25): consente la spedizione della posta (se abilitata)
- http (porta 80): protocollo http per comunicazione tra browser e www server

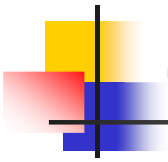


Servizi Standard

Esempio1: connessione alla porta 13 e stampa del risultato ottenuto

```
import java.net.*;
import java.io.*;
public class EsempioNet1 {
   public static void main(String args[]){
   Socket s = null;
   try {  s = new Socket(…,13);
    InputStream is = s.getInputStream();
    InputStreamReader ir =new InputStreamReader(is);
    BufferedReader r = new BufferedReader(ir);
    String line = r.readLine();
    System.out.println(line);
    s.close();
     } catch (UnknownHostException e){
    System.err.println("Host unknown");
     } catch (Exception e){System.err.println(e);
  }}}
```

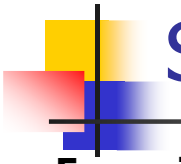
RISULTATO: thu dec 09 16:44:46 2007



Servizi Standard

Esempio 2: un cliente che si connette a una macchina remota sulla porta di echo (porta 7), le invia un messaggio, e stampa ciò che riceve.

```
import java.net.*;
import java.io.*;
public class EsempioNet2 {
   public static void main(String args[]){
    Socket s = null;
    try {     s = new Socket(…,7);
              InputStream is = s.getInputStream();
              InputStreamReader ir =
                        New InputStreamReader(is);
              BufferedReader r =
                        new BufferedReader(ir);
              OutputStream os = s.getOutputStream();
              OutputStreamWriter or =
                        new OutputStreamWriter(os);
              BufferedWriter outr =
                        new BufferedWriter(or);
```

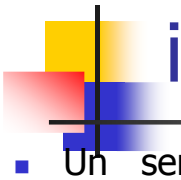


Servizi Standard

Esempio 2 continua:

```
                    String msg = "Ciao, io sono Pippo!";
                    outr.write(message, 0, message.length());
                    outr.newLine();//invia fine linea
                    outr.flush(); //svuota buffer
                //istruzioni importanti
        String line = inr.readLine();
                System.out.println("Ricevuto: ");
                System.out.println(line);
                s.close();
        } catch (UnknownHostException e){
                System.err.println("Host unknown");
        } catch (Exception e){
                System.err.println(e);
        }
        }
}
```

Risultato Esempio 2:
Ciao, io sono Pippo!



Socket stream: implementazione

- Un servitore deve creare la propria ServerSocket su una porta predeterminata
  - il numero di porta del server deve essere noto ai clienti perché ne avranno bisogno per connettersi al servitore
  - per i servizi standard, i numeri di porta sono standardizzati
  - per i nostri servizi, possiamo scegliere un numero qualunque, purché libero
- Costruire una ServerSocket significa predisporsi a ricevere richieste di connessione da clienti remoti.
  public ServerSocket(int localPort)
- Crea una ServerSocket e la collega alla porta locale specificata
  public ServerSocket(int localPort, int count)
- Crea una ServerSocket che accetta al massimo count richieste pendenti, e la collega alla porta locale specificata
  public Socket accept()
- mette il servitore in attesa di nuove richieste di connessione: se non ce ne sono, il servitore si blocca in attesa, altrimenti restituisce un oggetto Socket, tramite cui avviene l'effettiva comunicazione tra cliente e servitore. Da quel momento in poi, anche il servitore comunica col cliente tramite una normale Socket. Valgono quindi gli stessi mezzi visti poc'anzi.

Socket stream: implementazione

- Altri metodi utili
  public InetAddress getInetAddress()
- restituisce l'indirizzo della macchina locale
  public int getLocalPort()
- restituisce il numero di porta sulla macchina locale a cui la socket è legata
- Esempio di implementazione:

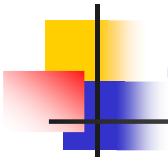
```
...
ServerSocket ss = new ServerSocket(porta);
    try {
        while (true) {
            /*una volta attivato il server non termina mai; svolge per
    definizione un ciclo infinito */
            Socket clientSock = ss.accept();
            /* clientSock è la socket tramite cui si parla col cliente*/
            ... fase di comunicazione col cliente...
            clientSock.close();
    }
    } catch (Exception e) { ... }
```


Socket stream: esempio

Un servitore daytime sulla porta 7777 (rimanda indietro la data)

SCHEMA DEL SERVER:

```
import …;
public class EsempioServer {
public static void main(String args[]){
    // creazione ServerSocket su porta 7777
    //ciclo senza fine:
    // attesa connessione,
    // recupero socket cliente,
    // recupero stream e comunicazione
    // chiusura socket cliente
    }
}
```

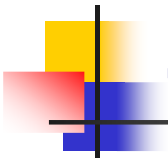


Socket stream: esempio

```
import java.net.*; import java.io.*;
public class EsempioServer {
    public static void main(String args[]){
    ServerSocket ss = null;
    try { ss = new ServerSocket(7777);
            while (true) {
                    Socket clientSock = ss.accept();
                    OutputStream os =clientSock.getOutputStream();
                    PrintStream outp = new PrintStream(os);
                    outp.println(new java.util.Date());
                    clientSock.close();}       // END WHILE
    } catch (UnknownHostException e){System.err.println("Host unknown");}
    catch (Exception e){System.err.println(e);}}
```

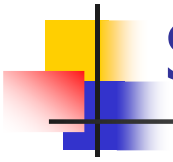
CLIENT:

```
    public class EsempioNet2 {
      public static void main(String args[]){
                Socket s = null;
                try { s = new Socket("localhost",7777);
        ...
```

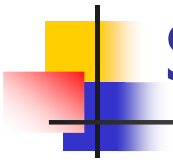
Socket stream: esempio

- Un altro esempio:
- Il server risponde con un numero progressivo, a partire da 1, a ogni cliente che si connette.
- Il cliente si limita a visualizzare tutto quello che gli arriva dal server, fino a quando non riceve il messaggio Stop: a quel punto, il cliente termina.



Socket stream: esempio

```
import java.net.*; import java.io.*;
public class Cliente1 {
    public static void main(String args[]){
    Socket s = null;
    try { s = new Socket("localhost",11111);
          BufferedReader r =
          new BufferedReader( new
                              InputStreamReader(s.getInputStream()));
          String line;
          while((line=r.readLine())!=null ){
                    System.out.println(line);
                    if (line.equals("Stop")) break;
          }
    r.close(); s.close();
    } catch (UnknownHostException e)
          {System.err.println("Host unknown");
    } catch (Exception e){System.err.println(e);
} } }
```

Socket stream: esempio

```
import java.net.*; import java.io.*;
    public class Server1 {
          public static void main(String args[]){
                    Socket cs = null; ServerSocket ss = null;
                    int numero = 1;
                    try {
                              ss = new ServerSocket(11111);
                              while (true) { // ciclo infinito del server
                              cs = ss.accept();
                              PrintWriter pw = new
                              PrintWriter(cs.getOutputStream(),true);
                              // il parametro true significa autoflush attivo
                              pw.println("Nuovo numero: " + numero);
                              numero++; pw.println("Stop");
                              pw.close(); cs.close();   } // end while
                    }
    catch (UnknownHostException e){System.err.println("Host unknown"); }
    catch (Exception e){System.err.println(e); }
} }
```

Socket stream: esempio

...

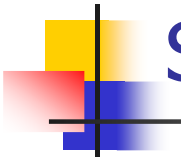
Il risultato invocando più clienti:

D:|esercizi>java Cliente1
Nuovo numero: 1
Stop
D:|esercizi>java Cliente1
Nuovo numero: 2
Stop
D:|esercizi>java Cliente1
Nuovo numero: 3
Stop

Socket stream: esempio

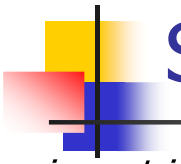
- Realizzare una mini calcolatrice client-server
- Il cliente invia al server una serie di valori double: lo 0 indica la fine della sequenza **(ipotesi: i valori sono sulla riga di comando).** Poi si mette in attesa del risultato dal server, e lo stampa a video.
- Il server riceve dal cliente una serie di valori double, e *li somma uno all'altro* fino a che riceve uno 0; a quel punto, invia il risultato al cliente.

```
import java.net.*; import java.io.*;
    public class Cliente2 {
          public static void main(String args[]){
          Socket s = null;
          try {     s = new Socket("localhost",11111);
                    DataInputStream is = new
    DataInputStream(s.getInputStream()));
                    DataOutputStream os = New
                     DataOutputStream(s.getOutputStream()));
```



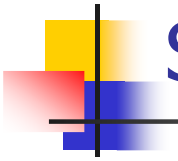
Socket stream: esempio

```
        for (int i=0; i<args.length; i++) {
              System.out.println("Sending " + args[i]);
              os.writeDouble(Double.parseDouble(args[i])); }
        os.writeDouble(0);
        // lettura risultato dal server
        double res = is.readDouble();
        System.out.println("Risultato = " + res);
        is.close(); os.close(); s.close();
        } catch (Exception e){System.err.println(e);}
    }
    }
```

Socket stream: esempio

```
import java.net.*; import java.io.*;
public class Server2 {
    public static void main(String args[]){
          Socket cs = null; ServerSocket ss = null;
          try {
                ss = new ServerSocket(11111);
                while (true) { // ciclo infinito
                cs = ss.accept();
                DataOutputStream os = new
                   DataOutputStream(cs.getOutputStream());
                DataInputStream os = new
                  DataInputStream(cs.getInputStream());
                double res = 0;
                while(is.available()>0){
                // finche ci sono dati disponibili
                      res += is.readDouble();
                      os.writeDouble(res);
                      os.close(); is.close(); cs.close();
                } // end while   }
```



Socket stream: esempio

```
} catch (UnknownHostException e){
            System.err.println("Host unknown"); }
     catch (Exception e){
            System.err.println(e); }
} }
```

ESEMPIO client/server: Il risultato in alcune situazioni:

```
D:|esercizi>java Cliente2 3 4 5
Sending 3
Sending 4
Sending 5
Risultato = 12.0
D:|esercizi>java Cliente2 34 5
Sending 34
Sending 5
Risultato = 39.0
```