PROGRAMMAZIONE DI RETE

Socket in Java

Programmi e applicazioni per le reti

- Obiettivo della programmazione di rete:
  - fare comunicare due o più programmi presenti su elaboratori diversi
  - La rete può essere LAN, WAN o Internet

Programmi e applicazioni per le reti

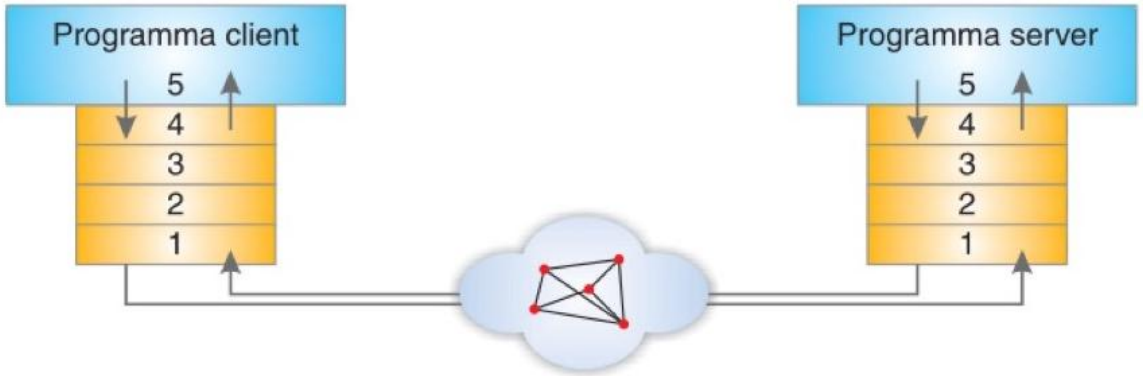
- Java è un linguaggio nato per realizzare applicazioni in rete
- Java supporta tutte le funzionalità necessarie per gestire la programmazione distribuita tra i computer di una rete (network programming).
- Si possono creare programmi che utilizzano il modello client/server e che si basano sui protocolli di rete TCP/IP

Client/Server

- Per costruire un'applicazione in rete si devono realizzare almeno un programma **server** e un programma **client**.
- La comunicazione tra i due programmi viene gestita tramite un insieme ben definito di regole, chiamate **protocolli**

ArchitetturaTCP/IP

- I programmi **server** e **client** si posizionano sul livello 5
- Comunicano utilizzando i servizi offerti dagli altri livelli dell'architettura.
- I programmi si interfacciano direttamente solo con il livello 4



ArchitetturaTCP/IP (2)

- le informazioni vengono spedite indicando il computer di destinazione con un indirizzo **IP**.
- L'applicazione destinataria all'interno del PC è individuata dalla **porta** (livello 4)
- Le porte sono identificate con numeri tra 0 e 65.535
- le porte da 0 a 1023, dette *well known ports*, sono riservate per protocolli predefiniti

Applicazioni client/server in Java

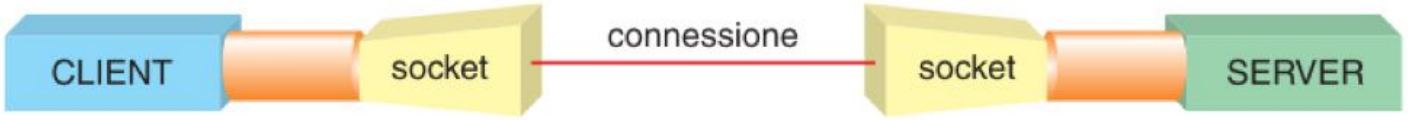
- La realizzazione di applicazioni client/server in Java si basa sulle classi contenute nel package j**ava.net**
- In quasi tutte le applicazioni di rete viene anche utilizzato il package **java.io** per gestire i flussi di comunicazione in input e output.

```
import java.net.*;
import java.io.*;
```

Applicazioni client/server in Java (2)

- I programmi client si differenziano dai programmi server per la loro struttura;
- la loro implementazione in Java è leggermente diversa.
- La comunicazione avviene tramite le entità chiamate **socket** (letteralmente, presa di corrente).

La **socket** è un punto da cui il programma può inviare i dati in rete e può ricevere i dati dalla rete.



Programmi Client

- Le operazioni eseguite da un programma client possono essere schematizzate con il diagramma di flusso a lato.

- L'apertura di una connessione corrisponde a una richiesta inviata al programma server tramite la

```
connessione = new Socket(server, porta);
```

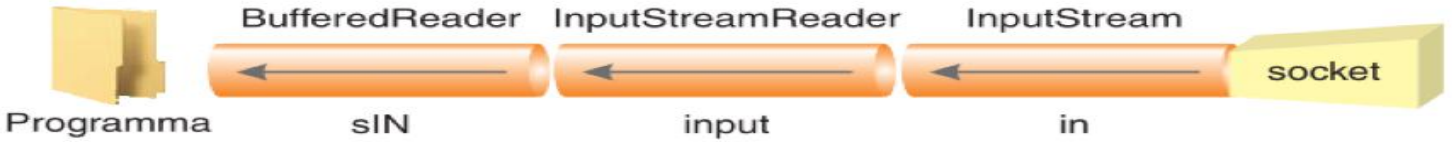
- L'invio e la ricezione dei dati sono gestite in maniera simile alla scrittura e alla lettura dei file di dati,tramite uno **stream**.
- Al **socket** è associato:
  - un flusso di ***input*** (ricezione dati dalla rete)
  - un flusso di ***output*** (trasmissione dati in rete)

Programmi Client (2)

- Le seguenti istruzioni mostrano come viene impostato il flusso di input nel caso in cui la comunicazione avvenga con sequenze di

```
InputStream in = connessione.getInputStream();
InputStreamReader input = new InputStreamReader(in);
BufferedReader sIN = new BufferedReader(input);
```

- Il metodo ***getInputStream*** restituisce il flusso in input associato alla socket.
- Il programma utilizza il metodo ***readLine*** della classe **BufferedReader** per ricevere i dati.

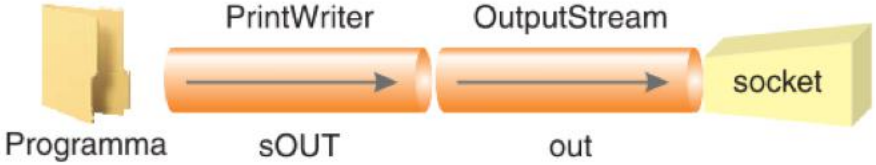


Programmi Client (3)

- Le seguenti istruzioni mostrano come viene impostato il flusso di output nel caso in cui la comunicazione avvenga con sequenze di

```
OutputStream out = connessione.getOutputStream();
PrintWriter sOUT = new PrintWriter(out);
```

- Il metodo ***getOutputStream*** restituisce il flusso in output associato alla socket.
- Il programma utilizza il metodo ***println*** della classe **PrintWriter** per inviare i dati.



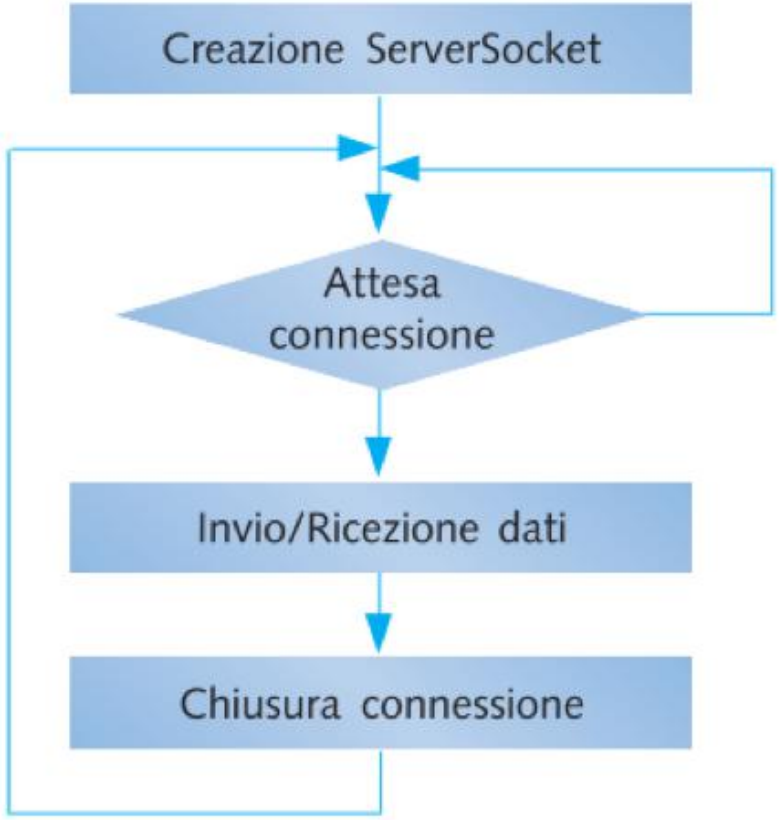
Programmi Client (4)

- La chiusura della connessione viene eseguita quando si vuole interrompere la comunicazione di rete.
- Si deve richiamare il metodo ***close*** nel seguente

```
connessione.close();
```

Programma Server

- Le operazioni eseguite da un programma server possono essere schematizzate con il seguente diagramma di flusso



Programma Server (2)

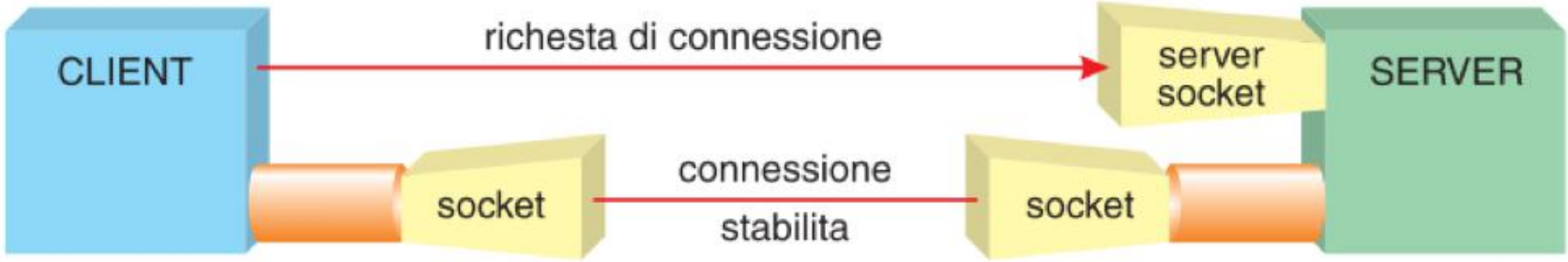
- I programmi server hanno la caratteristica di **restare attivi** in attesa di ricevere una richiesta di connessione da parte dei client
- Questo comportamento viene realizzato con una socket particolare implementata con la classe **ServerSocket**.
- La creazione di una socket di tipo server viene eseguita con la seguente istruzione:

```
ServerSocket sSocket = new ServerSocket(porta);
```

- Il parametro indica la porta che viene riservata all'applicazione per ricevere le richieste dagli altri computer.
- Il numero di porta deve essere superiore a 1023

Programma Server (3)

- Dopo aver creato la ServerSocket, il programma server si mette in attesa che un client si colleghi.
- Il metodo ***accept*** della classe **ServerSocket** è utilizzato per bloccare il programma server finché viene intercettata una richiesta di connessione.
- Quando giunge la richiesta, il metodo accept restituisce una socket che gestirà la connessione con il client.



Programma Server (4)

- Il codice che riassume il comportamento del server è il seguente:

```
while (true)
{
   Socket connessione = sSocket.accept();
   // invio-ricezione dati tramite la connessione
   connessione.close();
}
```

- Tutte le operazioni eseguite con le socket possono generare delle eccezioni, principalmente eccezioni di tipo ***IOException***
- È quindi necessario racchiudere le istruzioni nei blocchi ***try…catch*** per evitare che il programma venga interrotto bruscamente.

Esercizio 1

- Realizzare un'applicazione client/server utile ai client per sincronizzare la data e l'ora con quella di un server.
- Il programma client si collega al server per ottenere le informazioni sulla data e sull'ora correnti.
- Il server, ogni volta che riceve una richiesta di connessione, invia al programma client l'informazione richiesta.

Esercizio 2

- Realizzare la classe server e la classe client in modo che, a connessione avvenuta, il server invii al client il messaggio di benvenuto: "Hello, World!".
- Il client, ricevuto il messaggio termina la connessione, mentre il server resta in attesa.

Esercizio 3

- Modificare il programma client dell'esercizio 1 per inserire da linea di comando il nome del server e della porta a cui connettersi
- Il programma client deve essere poi eseguito usando il comando java ClientClock localhost 3333
- Si ricordi che i valori passati da linea di comando al programma Java sono contenuti nell'array args[ ]