UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI **MATEMATICA E INFORMATICA**

Interfacce Grafiche e Programmazione ad Eventi

Carmine Dodaro

Anno Accademico 2019/2020

Client/Server e Peer-to-peer

Modalità di comunicazione

- Client/Server
  - Un server che si occupa di gestire le richieste dei client
  - Tanti client che comunicano con un server
  - Funzionamento tipico:
    1) Il server resta in attesa di una richiesta da parte del client
    2) Il client effettua la richiesta
    3) Il server risponde alla richiesta e si torna al punto 1)
- Peer-to-peer (P2P)
  - I vari host si comportano sia come client che come server e comunicano tra di loro

Indirizzo IP e porta

Comunicazione

Per riuscire a far comunicare un client con un server, abbiamo bisogno dell'indirizzo IP del server e della porta in cui avverrà la comunicazione, dove:

- l'indirizzo IP identifica, all'interno della rete, il server a cui vogliamo collegarci
- Il numero della porta identifica il servizio a cui vogliamo connetterci, ad esempio la porta 80 è di solito usata per i servizi http
- un indirizzo particolare è rappresentato da 127.0.0.1 (detto anche localhost), è l'indirizzo attraverso cui ogni computer, all'interno di una rete, può contattare sé stesso usando un client.

Le reti in Java: Socket e ServerSocket

package java.net

Un socket è un endpoint per la comunicazione tra due macchine. Quindi, è una rappresentazione astratta che ci permette di far comunicare due terminali.

Per la comunicazione in rete, in Java si usa il concetto di stream (che abbiamo già visto per i file). Un socket ha uno stream di input e uno stream di output. Quando due terminali vogliono comunicare tra di loro attraverso il socket:

- un terminale manda i messaggi all'altro terminale scrivendo sullo stream di output associato ad un socket
- l'altro terminale usa lo stream di input per leggere i dati inviati dall'altro terminale

La classe ServerSocket è un'implementazione che permette di attendere, su una determinata porta, alcune richieste di connessioni che possono avvenire sulla rete.

Le reti in Java: Socket e ServerSocket

Funzionamento

1. Il server usa un ServerSocket per mettersi in attesa di connessioni su una porta
2. Il client prova a collegarsi, attraverso un socket, al ServerSocket del server
3. Se la connessione riesce, viene creato un socket nel server
4. Client e server possono comunicare tra di loro usando i rispettivi socket

Le reti in Java: Socket e ServerSocket

Server

```
ServerSocket server = new ServerSocket(8000);
Socket socket = server.accept();
```

Il server attende per una connessione sulla porta 8000. Non appena un client richiede di connettersi il metodo accept restituisce un Socket che permetterà al server di scrivere al client e di ricevere dal client. ATTENZIONE: la scelta della porta è a vostra discrezione. Alcune porte potrebbero essere già occupate da altri processi e quindi non sarebbe possibile usarle.

Client

```
Socket socket = new Socket("127.0.0.1", 8000);
```

Il client si connette al server che si trova in ascolto sulla porta 8000 all'indirizzo IP 127.0.0.1. Il client può usare socket per scrivere al server e ricevere dal server.

Inviare e ricevere stringhe di testo

Inviare messaggi

```
BufferedOutputStream bOut = new BufferedOutputStream(
    socket.getOutputStream());
PrintWriter out = new PrintWriter(bOut, true); //flush
out.append("My message");
out.flush();
//or
out.println("Another possible way to send a message");
```

Ricevere messaggi

```
BufferedReader in = new BufferedReader(new
    InputStreamReader(socket.getInputStream()));
while (connected) {
  if (in.ready()) {
    System.out.println(in.readLine());
  }
}
```

Inviare e ricevere oggetti

Inviare oggetti

```
ObjectOutputStream out = new ObjectOutputStream(socket.
    getOutputStream());
out.writeObject(new Integer(10));
out.writeObject("Hi");
out.writeObject(new Long(10000));
out.writeObject(new Student("id", "name"));
out.writeObject(new JPanel());
out.flush();
```

Ricevere oggetti

```
ObjectInputStream out = new ObjectInputStream(socket.
    getInputStream());
Integer i = (Integer) out.readObject();
String s = (String) out.readObject();
Long l = (Long) out.readObject();
Student s = (Studente) out.readObject();
JPanel p = (JPanel) out.readObject();
```

Inviare e ricevere oggetti

Quali oggetti si possono inviare?

In Java si possono inviare oggetti tramite rete che implementano l'interfaccia Serializable, che si trova nel package java.io.Serializable. In realtà, gli oggetti che implementano Serializable possono anche essere salvati interamente su file. L'interfaccia Serializable non ha metodi o campi e serve solo per identificare gli oggetti che possono essere serializzati.

Cos'è il serialVersionUID

Durante la serializzazione di un oggetto, ad ogni classe viene associato un numero di versione, appunto il serialVersionUID, che è usato successivamente durante la deserializzazione per verificare che il mittente e il destinatario di un oggetto serializzato abbiano caricato la classe di quell'oggetto.

Rendere una class serializable

classe Student

```
public class Student implements Serializable {
  private static final long serialVersionUID =
      4094302331073094744L;
  private String firstName;
  private String lastName;
  transient private String id;
  public Studente(String id, String f, String l) {
    this.id = id;
    this.firstName = f;
    this.lastName = l;
  }

  public String toString() {
    return id + " " + firstName + " " + lastName;
  }
}
```

Gli oggetti di tipo transient non sono trasferiti, quindi il loro valore sarà null nel destinatario.