Föreläsning 8

- Client/Server
- TCP/IP

JNP: kapitel 8

Klient/Server

Kommunikation bygger ofta på modellen *klient-server*. En klient kopplar upp sig mot en server för att erhålla någon form av tjänst.

Tjänsten kan vara t.ex. vara att få ta del av en websida (Webserver) eller en fil (FTP-server).

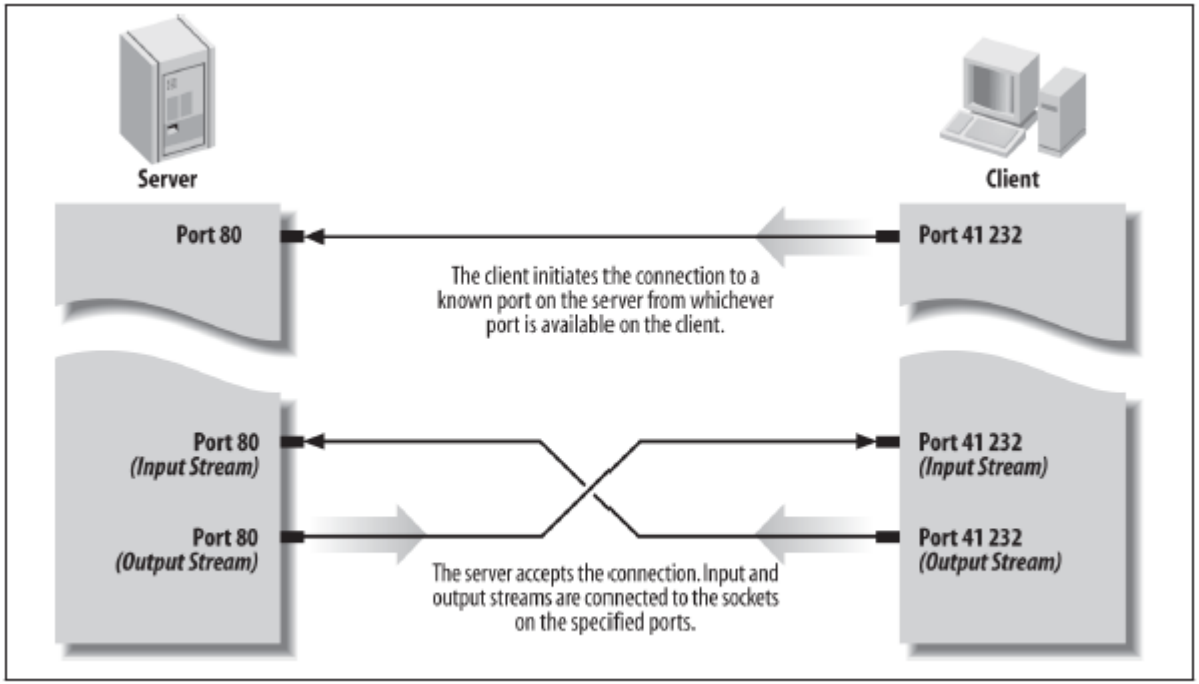


Figure 1-5. A client/server connection

TCP

TCP ger ökad tillförlitlighet vid kommunikation mellan datorer. Paket som inte kommit fram eller vars innehåll förändrats kan skickas på nytt. Och protokollet ser till att paketen har korrekt ordning.

Kommunikationen sker via en ***InputStream*** och en ***OutputStream*** i vardera enhet

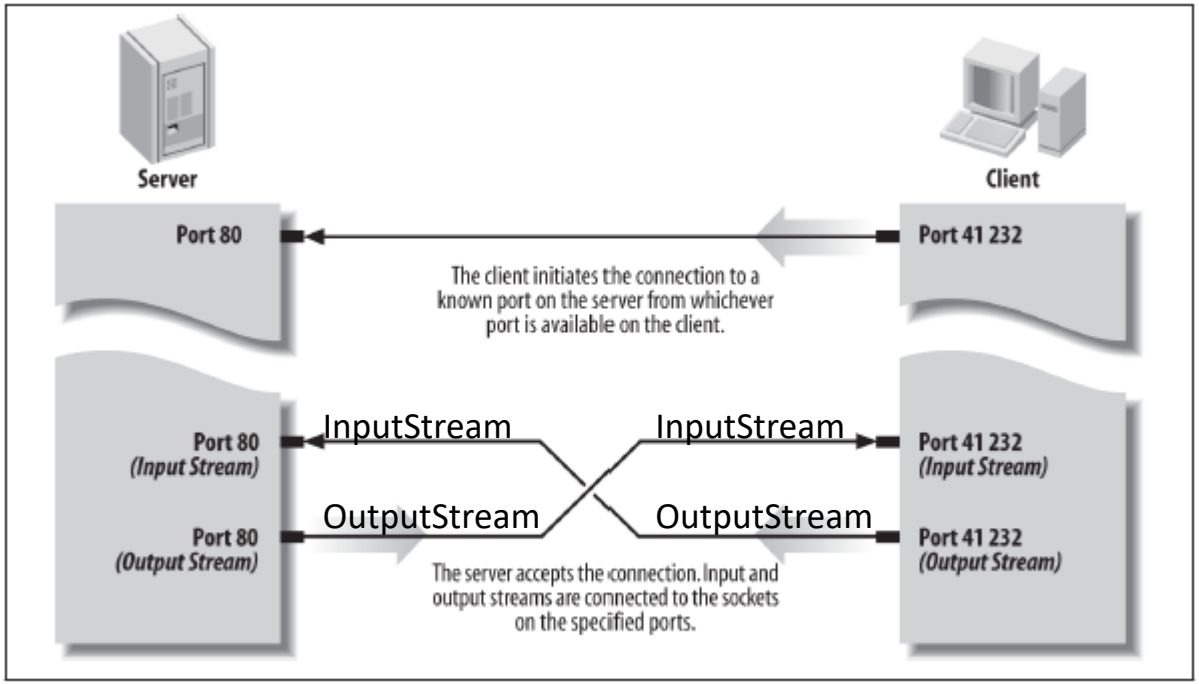


Figure 1-5. A client/server connection

TCP

Vid kommunikation mellan enheterna används klasserna ***ServerSocket*** och ***Socket***. Och naturligtvis lämpliga *strömmar*.

Servern använder sig av både *ServerSocket*- och *Socket*-objekt medan **klienten** använder sig av **Socket**-objekt .

Via *Socket*-objektet erhåller man ***InputStream***-objekt respektive ***OutputStream***-objekt.

- Om man ska överföra enkla datatyper + strängar så kan man koppla strömmarna till objekt av typen ***DataInputStream*** respektive ***DataOutputStream***.
- Om man ska överföra objekt så kan man koppla strömmarna till objekt av typen ***ObjectInputStream*** respektive ***ObjectOutputStream***.
- Om man ska överföra text så kan ***Reader***- respektive ***Writer***-objekt vara lämpliga

Socket, konstruktorer

Constructors

Modifier	Constructor and Description
	`Socket()` Creates an unconnected socket, with the system-default type of SocketImpl.
	`Socket(InetAddress address, int port)` Creates a stream socket and connects it to the specified port number at the specified IP address.
	`Socket(InetAddress address, int port, InetAddress localAddr, int localPort)` Creates a socket and connects it to the specified remote address on the specified remote port.
	`Socket(Proxy proxy)` Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.
protected	`Socket(SocketImpl impl)` Creates an unconnected Socket with a user-specified SocketImpl.
	`Socket(String host, int port)` Creates a stream socket and connects it to the specified port number on the named host.
	`Socket(String host, int port, InetAddress localAddr, int localPort)` Creates a socket and connects it to the specified remote host on the specified remote port.

Flera av konstruktorerna kan kasta ***IOException***.

Socket, metoder bl.a.

Methods

Modifier and Type	Method and Description
`void`	**`close`**`()` Closes this socket.
`InetAddress`	**`getInetAddress`**`()` Returns the address to which the socket is connected.
`InputStream`	**`getInputStream`**`()` Returns an input stream for this socket.
`OutputStream`	**`getOutputStream`**`()` Returns an output stream for this socket.
`int`	**`getPort`**`()` Returns the remote port number to which this socket is connected.
`void`	**`setSoTimeout`**`(int timeout)` Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
`String`	**`toString`**`()` Converts this socket to a `String`.

Flera av metoderna kan kasta ***IOException***.

Koppla upp till tidserver

time.nist.gov ger Greenwich Mean Time (GMT) på port 13. Servern returnerar en eller flera rader med text.

Några steg behövs för att erhålla text-raden med tidsinformation:

- Ett *Socket*-objekt måste skapas:
```
Socket socket = new Socket( "time.nist.gov", 13 );
```

- En *InputStream* behövs:
```
InputStream is = socket.getInputStream();
```

- Strömmen kan läsas på flera sätt. Ett är att använda en *BufferedReader*:
```
BufferedReader br = new BufferedReader( new InputStreamReader( is ) );
```

- Dags att läsa strängen och göra något med den:
```
String time = br.readLine();
while(time!=null) {
   System.out.println( time );  // Den lästa strängen skrivs ut
   time = br.readLine();
}
```

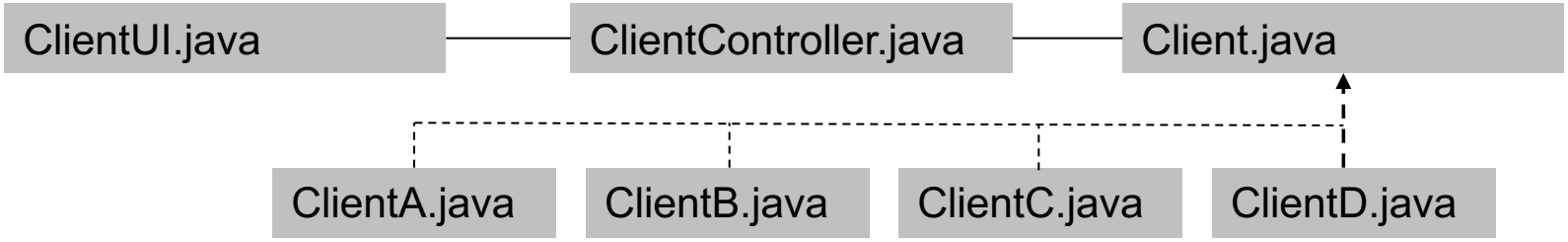
- Slutligen ska strömmen stängas:
```
br.close();
```

Time.java

Några liknande klienter

Client A – ***ClientD*** kommunicerar samma information till en server men på lite olika sätt. Följande operationer kan en klient göra:

- En klient kan lagrat information i servern. Vid förfrågan ska operationen ***PUT*** tillsammans med ***namn*** och ***ålder*** föras över till servern.
  Servern svara med en sträng på formen
- ”AAA,BBB lagrades” där AAA är namnet och BBB är åldern. Detta svar ges om AAA inte finns i servern.
- ”AAA ,CCC” ersattes av ” AAA;BBB” om AAA redan finns i servern.
- En klient kan fråga om information om vis person. Vid förfrågan ska operationen ***GET*** tillsammans med ***namn*** föras över till servern. Servern svara med en sträng på formen
- ”AAA,BBB” där AAA är namnet och BBB är åldern.
- ”AAA är okänd” om AAA ej lagras i servern
- Operationen ***LIST*** ger en lista på de personer som lagras i servern
- Operationen ***REMOVE namn*** tar bort *namn* ur servern.



ClientA – tillfälligt uppkopplad

Client A startar en ny uppkoppling vid varje förfrågan till servern. En förfrågan ser typiskt ut så här:

```
connect();  // uppkoppling mot servern
dos.writeUTF("PUT " + name + " " + age); // skriver sträng till servern
dos.flush(); //  ser till att strängen skickas
String response = dis.readUTF(); //  får resultat från servern, blockerar händelsetråden!
controller.newResponse(response); // meddelar controller-objektet svaret från servern
disconnect();  // avslutar uppkopplingen mot servern
```

Det är ett par nackdelar med klienten:

- Om uppkopplingen tar tid så blir programmet långsamt
- Händelsetråden används för att vänta på serverns respons. Om denna dröjer så blir händelsehanteringen låst under en lång stund.

ClientA.java

ClientB – uppkopplad

Client B startar en uppkoppling då en instans av klassen skapas. När användaren klickar på Quit så kopplar klienter ner.

Klassen använder en separat tråd för att lyssna efter svar från servern.

```
private class Listener extends Thread {
    public void run() {
        String response;
        try {
            while(true) {
                response = dis.readUTF();
                controller.newResponse(response);
            }
        } catch(IOException e) {}
        try {
            exit();
        } catch(IOException e) {}
        controller.newResponse("Klient kopplar ner");
    }
}
```

Kommunikationen sker på samma sätt som i Client A.

```
dos.writeUTF("PUT " + name + " " + age); // skriver sträng till servern
dos.flush(); //  ser till att strängen skickas
------------
String response = dis.readUTF(); //  får resultat från servern via separat tråd!
```

ClientB.java

ClientC – uppkopplad

Client C skiljer sig från Client B på ett sätt, nämligen sättet informationen överförs från klienten till servern.

Att placera information på servern ser ut så här:

```
public void put(String name, String ageStr) throws IOException {
    int age = -1;
    try {
        age = Integer.parseInt(ageStr);
    } catch(NumberFormatException e) {}
    dos.writeUTF("PUT");  // operation skrivs i utströmmen
    dos.writeUTF(name);  // namn skrivs i utströmmen
    dos.writeInt(age);  // ålder skrivs i utströmmen som en int
    dos.flush();  // ser till så att informationenförs över till servern
}
```

ClientC.java

ClientD – uppkopplad

Client D skiljer sig från Client C på ett sätt, nämligen sättet informationen överförs från klienten till servern.

I ClientD för all information över i ett Request-objekt

```
public class Request implements Serializable {
    private Type type;
    private String name;
    private int age;

    public Request(Type type, String name, int age) {
        this.type = type;
        this.name = name;
        this.age = age;
    }
    // getmetoder och toString-metod
}
```

Att placera information på servern ser ut så här:

```
public void put(String name, String ageStr) throws IOException {
    int age = -1;
    try {
        age = Integer.parseInt(ageStr);
    } catch(NumberFormatException e) {}
    oos.writeObject(new Request(Type.PUT,name,age));
    oos.flush();
}
```

ClientD.java