Verkkoliikennettä Java[ssa|lla]

Jouni Smed

9.2.2001

Perusteita 1 (2)

- tarvittavat luokat paketissa `java.net`
- IP-osoitteita käsitellään `InetAddress`-olioina
- luonti (huom. ei konstruktoria):

```
InetAddress addr =
   InetAddress.getByName(address);
```

- parametri
  - DNS-muodossa (`"staff.cs.utu.fi"`)
  - IP-numerona (`"139.232.75.8"`)
  - `null` (= `"localhost"` = `"127.0.0.1"`)

Perusteita 2 (2)

- portti ohjaa osoitteeseen tulevan liikenteen oikealle pistokkeelle (*socket*)
- porttinumerot 1–1024 varattuja
- pistoketyypit:
  - `ServerSocket`: kuuntee tiettyyn porttiin tulevia yhteyspyyntöjä
  - `Socket`: varsinainen pistoke, joka hoitaa viestin välityksen

Palvelin-asiakas -esimerkki

- palvelin

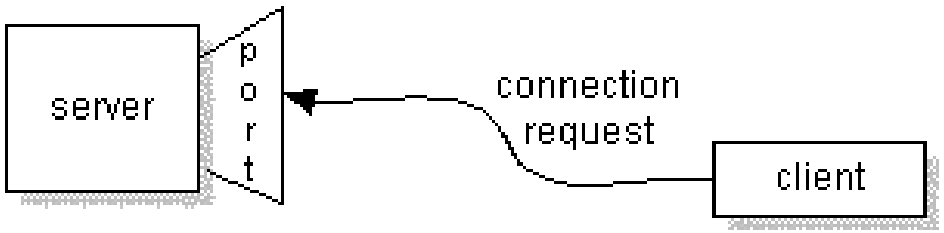
```
ServerSocket s = new
    ServerSocket(PORT);
try {
  Socket socket =
      s.accept();
  try {
      // käytetään
      // pistoketta
  } finally {
      socket.close();
} finally {
    s.close();
}
```

- asiakas

```
Socket socket = new
    Socket(addr, PORT);
try {
    // käytetään
    // pistoketta
} finally {
    socket.close();
}
```

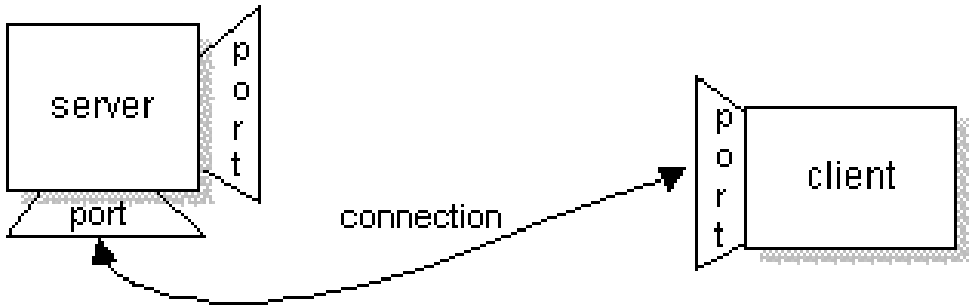
Mitä oikein tapahtuu? 1 (2)

- palvelin luo palvelupistokkeen, joka jää kuuntelemaan annettua porttia (ts. suoritus pysähtyy `accept()`-metodiin)
- asiakas luo pistokkeen antaen sille palvelimen osoitteen ja palvelupistokkeen porttinumeron; luotu pistoke lähettää yhteyskutsun



Mitä oikein tapahtuu? 2 (2)

- palvelupistoke vastaa kutsuun luomalla pistokkeen *johonkin* porttiin ja välittämällä tämän porttinumeron asiakkaalle
- asiakas yhdistää pistokkeensa uuteen porttiin ja pistoke palautetaan konstruktorin kutsujalle
- `accept()` palauttaa palvelimeen luodun pistokkeen



Tietovirtojen käyttö 1 (2)

- syötevirta:

```
BufferedReader in = new
  BufferedReader(new InputStreamReader(
  socket.getInputStream()));
```

- tulosvirta:

```
PrintWriter out = new PrintWriter(new
  Buffered Writer(new
  OutputStreamWriter(
  socket.getOutputStream())), true);
```

- virtoihin luku ja kirjoitus normaalia:

```
out.println("foo");
String s = in.readLine();
```

Tietovirtojen käyttö 2 (2)

- huom. eo. palvelin voi olla yhteydessä vain yhteen asiakkaaseen kerrallaan
- useampi samanaikainen asiakas
  $\Rightarrow$ palvelin luo vastauspistokkeen uuteen säikeeseen
- virrat käyttävät TCP:a
  $\Rightarrow$ luotettavaa mutta hidasta

Palvelinesimerkki 1 (2)

```
class Server extends Thread {
  public Server() { setDaemon(true); }

  public void run() {
    try {
      ServerSocket s = new ServerSocket(PORT);
      while (!terminated) {
        Socket socket = s.accept();
        new Thread(new Handler(socket)).start();
      }
      s.close();
    } catch (IOException e) {
      // portti ei auennut
    }
  }
}
```

Palvelinesimerkki 2 (2)

```
class Handler extends Thread {
  private Socket socket;
  public Handler(Socket s) { socket = s; }

  public void run() {
    try {
      BufferedReader in = new BufferedReader(new
        InputStreamReader(socket.getInputStream()));
      // luetaan jotain
      in.close();
    } catch (IOException e) {
      // lukijan luonti tai luku epäonnistui
    } finally {
      try {
        socket.close();
      } catch (IOException e) {
        // pistoke ei sulkeutunut
} } } }
```

TCP versus UDP

- TCP = *transmission control protocol*
- *reliable, stream-based, point-to-point, lost data retransmission, rerouted if necessary, bytes delivered in the order they are sent;* ***but*** *has a high overhead*
- TCP-portti ≠ UDP-portti

- UDP = *user datagram protocol*
- *independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed;* ***but*** *has a low overhead (28 bytes) and allows multicasting*

UDP ja datagrammit

- ei tarvita palvelinpistoketta: `DatagramSocket`-pistokkeet sekä lähettävät että vastaanottavat paketteja
- `DatagramPacket`-olio sisältää lähetettävän/vastaanotetun tiedon (maksimikoko 64 kB)
- vastaanottava paketti:

```
DatagramPacket dp1 = new
  DatagramPacket(buf, buf.length);
```

- lähetettävä paketti:

```
DatagramPacket dp2 = new
  DatagramPacket(buf, len, addr, port);
```

Datagrammi-esimerkki

```
try {
  socket = new DatagramSocket(PORT);
  socket.receive(dp1);
  socket.send(dp2);
} catch (SocketException e) {
  // pistoke ei auennut
} catch (IOException e) {
  // vikaa viestinnässä
} finally {
  socket.close();
}
```

Datagrammin sisältö

- lähettäjän osoite: `InetAddress addr = dp.getAddress();`
- lähettäjän portti: `int port = dp.getPort();`
- paketin koko: `int l = dp.getLength();`
- paketin sisältö: `byte[] buf = dp.getData();`
- paketin lähettäjän ei tarvitse (välttämättä) sisällyttää yhteystietojaan dataan, vaan vastaanottaja voi selvittää sen yo. metodeilla
- $\Rightarrow$ mahdollisuus väärinkäyttöön: DoS (*denial of service*), jossa paketin lähettäjä on väärentänyt yhteystiedot

Multicast 1 (2)

- UDP:tä käyttävää liikennettä, jossa datagrammia ei lähetetä yksittäiselle koneelle vaan D-luokan osoitteen ilmaisemalle ryhmälle
- D-luokan osoitteet:
  224.0.0.0–239.255.255.255
  paikallisille sovelluksille:
  239.0.0.0–239.255.255.255
- lähettäminen kuten *unicast*-paketeilla

Multicast 2 (2)

- vastaanottajan on liityttävä kuuntelemaan annettua ryhmää:

```
MulticastSocket socket =
  new MulticastSocket(PORT);
InetAddress group =
  InetAddress.getByName(ADDR);
socket.joinGroup(group);
```

- vastaanotto kuten *unicast*-paketeille:

```
socket.receive(dp);
```

- lopuksi lähdetään ryhmästä:

```
socket.leaveGroup(group);
socket.close();
```

Multicast: palvelin

```
class MulticastServer {
  private Socket socket;

  public MulticastServer() {
    try { socket = new DatagramSocket(PORT); }
    catch (SocketException e) { /* pistoke ei auennut */
  } }

  public void send(byte[] data) {
    try {
      Datagram packet = new DatagramPacket(data,
        data.length, GROUP_ADDRESS, PORT);
      socket.send(packet);
    } catch (IOException e) { /* lähetys epäonnistui */
  } }

  public void finalize() {
    socket.close();
    super.finalize();
} }
```

Multicast: asiakas

```
class MulticastClient {
  private MulticastSocket socket;

  public MulticastClient() {
    try {
      socket = new MulticastSocket(PORT);
      socket.joinGroup(GROUP_ADDRESS);
    } catch (IOException e) { /* ei onnistunut */ }
  }

  public byte[] receive() {
    byte[] buf = new byte[BUFFER_SIZE];
    DatagramPacket packet = new DatagramPacket(buf, buf.length);
    try {
      socket.receive(packet);
      return packet.getData();
    } catch (IOException e) { /* vastaanotto epäonnistui */ }
    return null;
} }
```