Hajautetut ohjelmistojärjestelmät ja pilvipalvelut

Soketit 8.11. ja 10.11.2017

Yleistä

- *Hajautettu sovellus*: joukko erillisiä ohjelmia, jotka “keskustelevat” toistensa kanssa.
- Ohjelmat toimivat yleisesti ottaen eri koneissa, mutta näin ei välttämättä ole.
- Huomattava, että laitteet voivat olla myös poissa verkosta tai rikki.
- Kommunikointi: verkon yli, jollakin “keskustelukielellä”.
- Mahdollisesti erikielisiä ohjelmia.
- Yleensä kommunikoivilla ohjelmilla hyvin erilainen rooli.

Yleistä

- *Asiakas*- ja *palvelin*sovelluksia.
- *Palvelin*: sovellus, joka kuuntelee yhteydenottoja.
- *Asiakas*: aloittaa kommunikaation; kommunikaation suunnan painopiste vaihtelee tapauskohtaisesti.
- Roolijako ei täysin selkeä, sama ohjelma voi olla sekä asiakas että palvelin (tavallisesti eri ohjelmajoukkojen suuntiin)
- Yleensä *välitetään "passiivista" tietoa*; ei niinkään pyritä vaikuttamaan toisen ohjelman suoritukseen.
- Verkkosovellukset ja monisäieohjelmat voidaan tietyssä mielessä rinnastaa

Esimerkkejä

- “Pelipöytä”: asiakassovellukset tarjoavat GUI-liittymän pelaajalle; pelin luonti (pelitilanne palvelimella); pelaajan liittyminen peliin; pelaaminen = palvelin välittää siirtoja (& luo tilanteita).
- “Tietokanta”: palvelin ylläpitää tietoa; asiakassovellukset hakevat ja päivittävät. Esim. demosuoritusten ylläpito; kurssisuoritusten ylläpito; harjoitustöiden ohjauksen seuranta; tuotevalikoiman ylläpito; . . .
- “Tiedotuspalvelu”: asiakas vain hakee tietoa palvelimelta. Esim. kellon aika; noppien arpominen; päivän tiedotukset. . .
- “Palvelupyynnöt”: sähköposti lähettäminen; tilauksen tekeminen

Ongelmia

- Miten välittää tietoa?
- Miten asiakas tietää, “missä palvelin on”?
- Miten palvelin osaa vastata?
- Yhteyden muodostaminen & lopetus?
- Useita asiakkaita?
- Asiakas käyttää useita palvelimia?
- Ongelmia samanaikaisuudesta?
- Palvelin: miten palvella kaikkia tasapuolisesti?
- Miten hallita hajautetun sovelluksen siirtymistä tilasta toiseen?

Kommunikointi

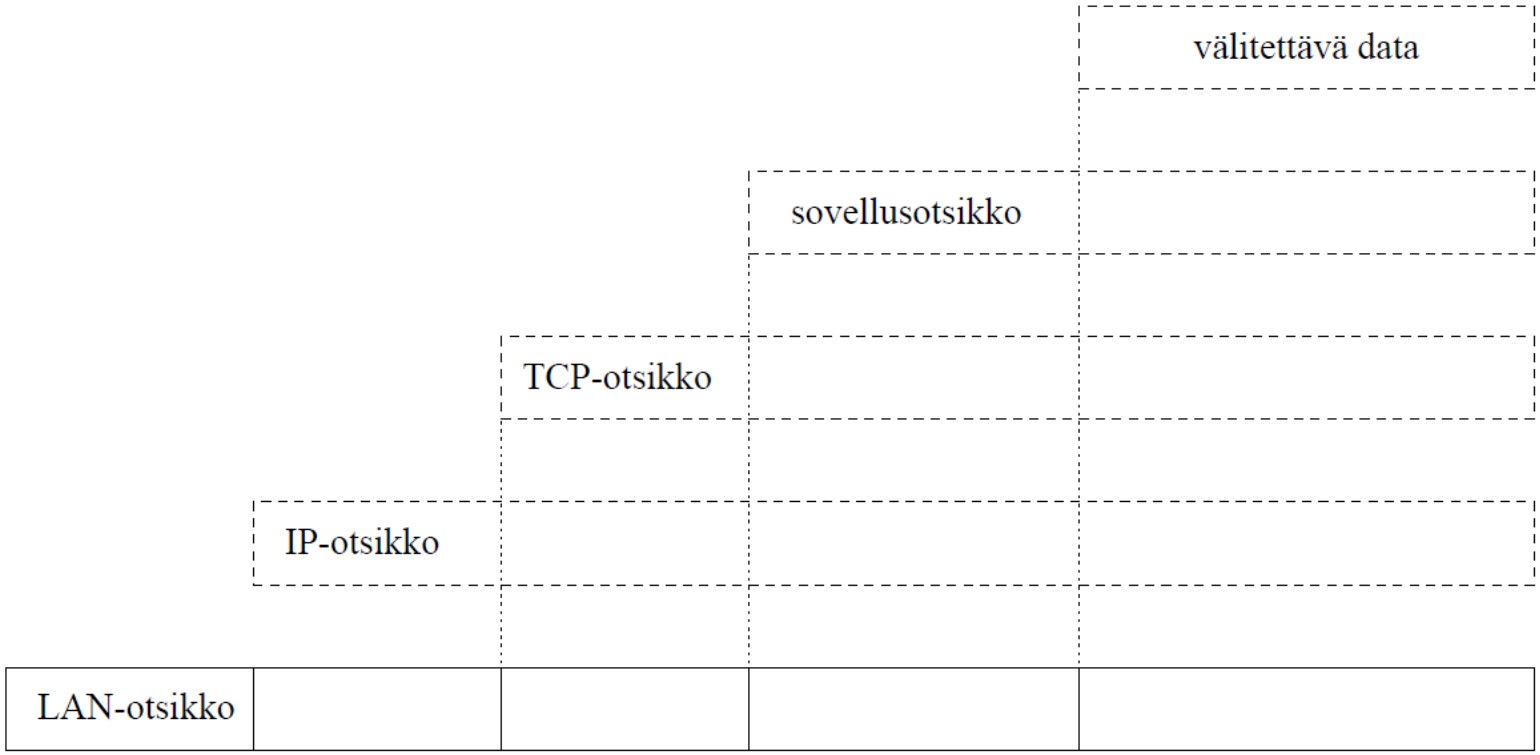
- Käytetään jotakin protokollaa, nykyisin tapahtuu yleensä TCP/IP:n päällä.
- IP: osoitteet koneille + viestien ja keskustelun muoto.
- IP-osoite; esim. 130.232.1.1
- IPv6: 2001:0db8:85a3:08d3:1319:8a2e:0370:7334
- Internet-osoite: esim. linux.utu.fi
- Erikoistapauksena localhost
- TCP (Transmission Control Protocol): IP:n päällä; ideana taata, että paketit tulevat vastaanottavalle sovellukselle kunnossa ja oikeassa järjestyksessä. *Yhteydellinen*.
- UDP (User Datagram Protocol): *yhteydetön*, “best-try”, ei ongelmatilanteiden hallintaa (koodausvirheet, puuttuvat paketit).

Kommunikointi

- *Porttinumerot*: TCP:hen (ja UDP:hen) liittyvä numero; väliltä 0 – 65535; kertoo, “missä koneen sisällä” vastaanottaja on. Lähettäjällä on samoin porttinumero.
- **Soketit**: sovellustason yhteys (IP-osoite & portti), jota Java-ohjelmassa voidaan käsitellä tiedostovirran tapaan. Toisen osapuolen ei tarvitse olla Java-ohjelma.
- **RMI** (Remote Method Invocation): tapa välittää metodin kutsuja asiakkaalta palvelimelle. Kutsu = tietoa asiakkaalta + ohjaus; takaisin tietoa palvelimelta. Molemmat osapuolet ovat Java-ohjelmia.
- RPC, . . .

Verkon kerrokset

- Sovelluskerros (WWW-selain & -palvelin)
- Kuljetuskerros (TCP)
- Internet-kerros (IP)
- Fyysinen verkkokerros (Ethernet, ATM)

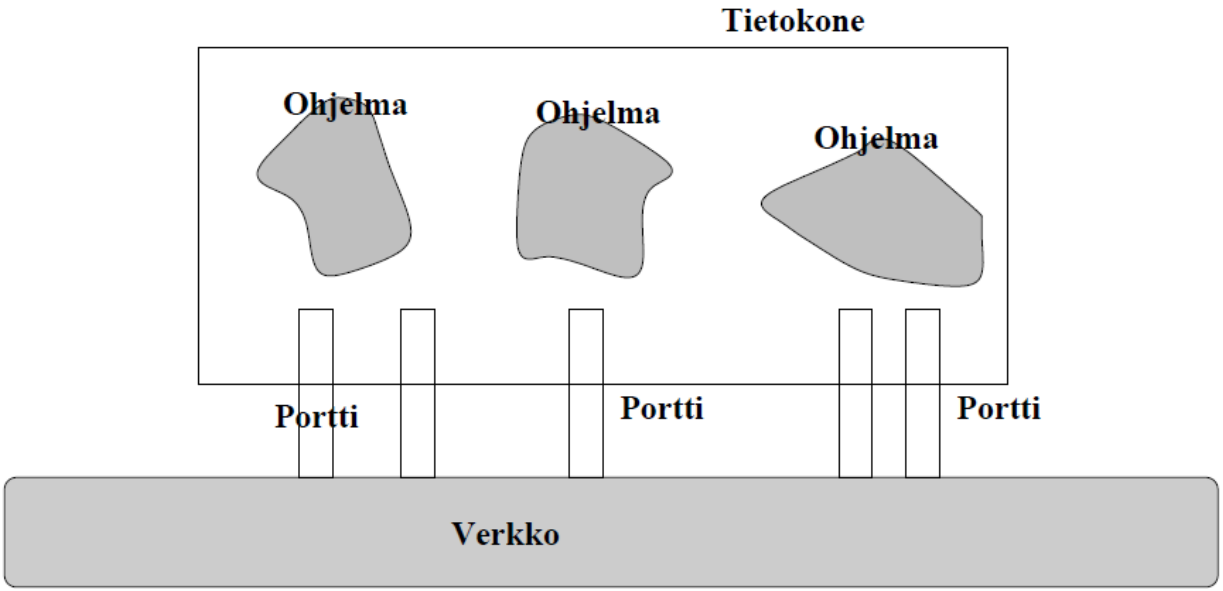


Soketit

- Soketti ≈ kytkös: (kone1, portti1) ↔ (kone2, portti2) niin, että kumpikin kone “tietää” kytköksen.
- IP:n päällä toimiva kanava, jonka kautta sovellus voi lähettää ja vastaanottaa paketteja
- UDP-pohjainen tiedon välitys: DatagramPacket ja DatagramSocket.
- TCP-pohjainen välitys: Socket ja ServerSocket.
- Tietovirrat InputStream, DataInputStream, ObjectInputStream.

- Useita asiakkaita?
- Monilähetys (ryhmälähetys, multicast): useita vastaanottajia.
- Yleislähetys (broadcast).
- Paketissa java.net.

Soketit



- Porttinumerot: 0 . . . 65535.
- Tietokoneella IP-osoite. (IPv4: 32-bit; IPv6: 128-bit.)
- Yksi ohjelma voi käyttää useampaa porttia samanaikaisesti.
- Koneen käyttöjärjestelmä välittää verkkokortin ottaman tiedon porttiin kytkeytyneelle ohjelmalle.
- Lähettäminen vastaavasti.
- 0 . . . 1023: standardoitu merkitys (IANA), esim. 80: HTTP

UDP-soketit

- Idea: asiakas luo paketteja (datagrammeja), joita se yksi kerrallaan lähettää vastaanottajalle.
- Asiakkaan lähetyskanava: DatagramSocket (+ mahdollisesti porttimääritys).
- Vastaanottajan kanava: DatagramSocket + porttimääritys.
- Paketit sisältävät tiedon lisäksi kohdeosoitteen ja portin.
- UDP: pakettien järjestys voi matkalla vaihtua; jotkin paketit voivat hukkua; ei kuittauksia; ei mitään automaattista kontrollia.

Luokka InetAddress 1/2

- IP-osoitteiden muodostamista varten.
- **static InetAddress getByName(String host)** - Muodostaa internet-nimen (tai IP-osoitteen) perusteella IP-osoitteen. Esim. linux.utu.fi → 130.232.203.60. UnknownHostException.
- **static InetAddress getLocalHost()** – Ohjelmaa ajavan koneen IP-osoite. UnknownHostException.
- **String getHostAddress()** - Palauttaa osoitteen numeromuodossa.
- **String getHostName()** - Palauttaa IP-osoitetta vastaavan internet-osoitteen.

Luokka InetAddress 2/2

- **boolean isMulticastAddress()** - Onko multicast-osoite?
- Nimet haetaan nimipalvelusta (DNS): voi tuottaa turvallisuuspoikkeuksen, jos yhteydenottoa ei sallittu.

Luokka DatagramPacket 1/2

- UDP-paketti.
- Tietosisältö: *dataa tavuina*, *IP-osoite*, *portti*. Vastaanottajan tai lähettäjän tietoja.
- **DatagramPacket(byte[] b, int l, InetAddress a, int p)** - Luo paketin, jonka sisältö on *l* tavua *b*:stä, kohteena “koneen” *a* portti *p*.
- **DatagramPacket(byte[] b, int l)** - Luo “paketin” vastaanottamista varten.
- **InetAddress getAddress()** - Palauttaa (vastaanotetun / luodun) paketin (lähettäjän / vastaanottajan) osoitteen.

Luokka DatagramPacket 2/2

- **int getPort()** - Palauttaa pakettiin liittyvän porttinumeron.
- **byte[] getData()** - Paketin data.
- **int getLength()** - Paketin datan pituus.
- Vastaavat *set-metodit*.

Luokka DatagramSocket 1/4

- Soketti datagrammien *lähettämiseksi* **ja** *vastaanottamiseksi*.
- Toteutus käyttää UDP:tä.
- **DatagramSocket(int port)** - Luo soketin paikallisen koneen porttiin *port*.
- Poikkeus **SocketException**: sokettia ei voitu luoda ko. porttiin.
- Poikkeus **SecurityException**: turvallisuusmanageri ei salli
- operaatiota.
- **DatagramSocket()** - Luo soketin johonkin vapaaseen paikallisen koneen porttiin.
- **DatagramSocket(int port, InetAddress laddr)** - Kuten edellä, mutta käyttäen paikalliskoneen osoitetta *laddr* (koneella voi olla useita osoitteita).

Luokka DatagramSocket 2/4

- **int getLocalPort()** - Palauttaa soketin portin.
- **InetAddress getLocalAddress()** - Palauttaa sokettiin liittyvän paikallisen koneen osoitteen.
- **void send(DatagramPacket p)** - Lähettää soketin kautta paketin *p* paketissa kerrottuun kohteeseen.
- **void receive(DatagramPacket p)** - Vastaanottaa soketin kautta paketin *p*:hen.
- Huomaa osoitteiden käyttö paketissa ja soketissa. Vastaanotetulla paketilla on lähettävän soketin tiedot!
- Poikkeukset IOException ja SecurityException.

Luokka DatagramSocket 3/4

- **void setSoTimeout(int timeout)** - Asettaa odotusajan paketin vastaanottamiselle. Jos aikaraja ylittyy: ’receive’ tuottaa poikkeuksen **InterruptedIOException**. Poikkeus **SocketException**, jos aikarajan asettaminen ei onnistu. Positiivinen tai 0 = rajoittamaton odotusaika.
- **int getSoTimeout()** - Palauttaa odotusajan.
- **void close()** - Sulkee soketin.

Luokka DatagramSocket 4/4

- **void connect(InetAddress a, int p)** - Kytkee soketin operoimaan vain kohteen (*a*, *p*) kanssa (tarkistukset puutteellisia). Oletusarvoisesti: voi operoida kaikkien kanssa.
- **void disconnect()** - Purkaa edellisen kytkennän.
- **InetAddress getInetAddress()** - Kohdekoneen osoite, johon soketti kytketty. null: ei kytketty.
- **int getPort()** - Portti johon kytketty. Arvo -1 = ei kytketty.

UDPSender.java

```
import java.io.*;
import java.net.*;
public class UDPSender {
    public static void main(String[] args) throws Exception {
        // Parametrit: kohdekone, portti, viesti
        if (args.length < 3) {
            System.out.println("Parametrit: kone, portti, viesti");
            System.exit(0);
        }
        InetAddress targetAddr = InetAddress.getByName(args[0]);
        int targetPort = Integer.parseInt(args[1]);
        DatagramSocket socket = new DatagramSocket();
        byte[] data = args[2].getBytes();
        DatagramPacket packet =
                new DatagramPacket (data, data.length, targetAddr, targetPort);
        socket.send(packet);
        socket.close();
    } // main
} // UDPLahettaja
```

UDPReceiver.java 1/3

```
public class UDPReceiver{
    public static final int PORT = 2000;
    public static void main(String[] args) throws Exception {
    JFrame f = new JFrame("UDPVastaanottaja");
    Container c = f.getContentPane();
    JTextField targetField = new JTextField(60);
    JButton b = new JButton("Aseta portti");
    final JTextField tf = new JTextField(30);
    final PortListener pListener = new PortListener(PORT,targetField);
    b.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e) {
            int p = PORT;
            try { p = Integer.parseInt(tf.getText().trim()); } catch (Exception e1) { }
                pListener.setNewPort(p); }});
    c.add(targetField, BorderLayout.NORTH); c.add(tf, BorderLayout.SOUTH);
    c.add(b); f.pack(); f.setVisible(true);
    f.addWindowListener(new WindowCloser()); pListener.start();
} // main
```

UDPReceiver.java 2/3

```
static class PortListener extends Thread {
    private int port;
    private JTextField targetField;
    private DatagramSocket socket;
    private boolean valid;
    public PortListener(int p, JTextField t) throws Exception {
        port = p; targetField = t;
        socket = new DatagramSocket(p);
        socket.setSoTimeout(5000); // 5 sekuntia
        valid = true;
        setDaemon(true);
    } // PortListener
    public void setNewPort(int p) {
        valid = (p == port);
        port = p;
    } // setNewPort
```

UDPReceiver.java 3/3

```
public void run() {
    try {
        byte[] data = new byte[256];
        DatagramPacket received = new DatagramPacket(data, data.length);
        while (true) {
            if (!valid) {
                socket.close();
                socket = new DatagramSocket(port);
                socket.setSoTimeout(5000);
            }
            try { socket.receive(received); }
                catch (InterruptedIOException e) { continue; }
            String msg = new String(received.getData(), 0, received.getLength());
            String newMsg = received.getAddress().getHostName()
                + ":" + received.getPort() + ” sanoo " + msg;
            targetField.setText(msg);
        } // while
    } catch (Exception e) { throw new Error(e.toString()); }
} // run
```

TCP-soketit

- TCP: pakettien järjestys ja perille pääsy “taataan”.
- Idea: *yhteydellinen kaksisuuntainen tietovirta*. Käytetään kuten muitakin Javan tietovirtoja.
- Asiakas luo: Socket-olion kohteeseen.
- Palvelin luo: *ServerSocket*-olion, joka kautta muodostetaan Socket-olio.
- Soketteihin liittyvät sisään- ja ulosmenevät tietovirrat.
- TCP:ssä ei ole eksplisiittisiä paketteja.
- *Kirjoitetaan vain tietovirtaan*!

Luokka Socket 1/2

- “Asiakas-soketti”, asiakaspuolen soketti.
- **Socket(InetAddress address, int port)** - Luo soketin kohteeseen *address*, porttiin *port*. Paikallisesta koneesta allokoidaan jokin vapaa portti. IOException: jos kytkeytyminen ei onnistu.
- **Socket(String host, int port)** - Luo soketin kohteeseen *host*, porttiin *port*.
- **Socket(String host, int p, InetAddress lAddr, int lp)** - Nyt paikallisen koneen portti on *p* ja osoite *lAddr*.
- **InputStream getInputStream()** - Palauttaa soketin sisääntulevan tiedostovirran.
- **OutputStream getOutputStream()** - Palauttaa soketin ulosmenevän tiedostovirran.

Luokka Socket 2/2

- **int getPort()** - Kohdekoneen portti.
- **int getLocalPort()** - Paikallisen koneen portti.
- **getInetAddress()** ja **getLocalAddress()** vastaavasti.
- **void setSoTimeout(int timeout)** - Asettaa timeout-arvon kuten UDP:n kohdalla.
- **void close()** - Sulkee soketin.
- **void shutdownInput()** - Sulkee sisääntulevan virran. Otetaan edelleen vastaan, mutta tuhotaan automaattisesti.
- **void shutdownOutput()** - Sulkee ulosmenevän virran. Kirjoitusyritys tuottaa poikkeuksen.

Luokka ServerSocket

- Palvelinsoketti.
- Yhteyden muodostuksen aloittaa asiakas ja palvelinsoketin tarkoitus on muodostaa palvelimelle asiakas-soketti.
- **ServerSocket(int port)** - Luo palvelin soketin porttiin *port*. *port* = 0 → mikä tahansa vapaa portti.
- **getInetAddress**, **getLocalPort**, **close** ja **setSoTimeout**: kuten aiemmin.
- **Socket accept() -** Ottaa vastaan yhteyden muodostuspyynnön ja luo sen perusteella asiakas-soketin.

Tietovirrat ja soketit

- Sokettien tietovirtaan voidaan kirjoittaa koodatusti melkein mitä vain.
- Javassa luokkia primitiivialkioiden ja olioiden koodaamiseksi.
- **DataOutputStream** ja **DataInputStream**: voidaan kirjoittaa primitiivityyppien mukaisia alkioita (’writeFloat’, ’readFloat’, . . .).

  new DataOutputStream(OutputStream)

  new DataInputStream(InputStream).
- ObjectInputStream & ObjectOutputStream: luokkia sarjallisten olioiden lukemista ja kirjoittamista varten! Konstruointi samoin.

Palvelimen toiminnasta

- Socket clientSocket = serverSocket.accept();
- Palvelin palvelee usein monia asiakkaita.
- **Ei voi asiakas-soketin muodostamisen jälkeen keskittyä kokonaan siihen.**
- Ratkaisu: asiakas-soketin käsittely siirretään sitä varten tehtävälle säikeelle.

Palvelimen ”ydin”

```
while (true) {
    Socket clientSocket = serverSocket.accept();
    CommHandler c = new CommHandler(clientSocket);
    c.start();
} // while
```

Finger.java

```
public class Finger {
    public static final int FINGER=79; // finger
    public static void main( String[] args ) throws Exception {
        if ( args.length < 2 ) {
            System.err.println( "Finger <host> <user>" );
            return;
        }
        Socket s = new Socket(args[0], FINGER);
        BufferedWriter sout =
            new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
        sout.write(args[1] + "\n");
        sout.flush();
        BufferedReader sin = new BufferedReader(
            new InputStreamReader(s.getInputStream()));
        String line = "";
        while ((line=sin.readLine()) != null)
            System.out.println(line);
        sout.close(); sin.close(); s.close();
    } // main
```

Point.java

```
import java.io.*;
public class Point implements Serializable {
    private int x;
    private int y;
    public Point(int x, int y) { this.x = x; this.y = y; }
    public void mirrorX() { x = -x; }
    public void mirrorY() { y = -y; }
    public void mirror() { this.mirrorX(); this.mirrorY(); }
    public int getX() { return x; }
    public int getY() { return y; }
    public String toString() { return "("+x+","+y+")"; }
} // Point
```

MirrorClient.java 1/2

```
import java.net.*;
import java.io.*;
public class MirrorClient {
    public static void main(String[] args) throws Exception {
    int port = MirrorServer.PORT;
    Socket s = new Socket(args[0], port);
    System.out.println("Connection made ...");
    InputStream iS = s.getInputStream();
    OutputStream oS = s.getOutputStream();
    // Seuraavan kahden järjestys merkitsevä!
    ObjectOutputStream oOut = new ObjectOutputStream(oS);
    ObjectInputStream oIn = new ObjectInputStream(iS);
```

MirrorClient.java 2/2

```
        for (int i=0; i<10; i++) {
            Point p = new Point((int)(20∗Math.random()), (int)(20∗Math.random()));
            oOut.writeObject(p); oOut.flush();
            System.out.println("Writing done ... sleeping..");
            Thread.sleep(2000);
            Point p1 = (Point) oIn.readObject();
            System.out.println("Put " + p.toString());
            System.out.println("Got " + p1.toString());
        } // for
        oIn.close(); oOut.close(); s.close();
    } // main
} // class MirrorClient
```

MirrorServer.java 1/3

```
import java.net.*;
import java.io.*;
    public class MirrorServer {
        public static final int PORT = 3333;
        public static void main(String[] args) throws Exception {
            ServerSocket ss = new ServerSocket(PORT);
            while (true) {
                Socket cs = ss.accept();
                System.out.println("Connection from ”
                    + cs.getInetAddress() + "port ” + cs.getPort());
                new MirrorHandler(cs).start();
            } // while
        } // main
```

MirrorServer.java 2/3

```
static class MirrorHandler extends Thread {
    private Socket client;
        public MirrorHandler(Socket s) { client = s; }

        public void run() {
            try {
                System.out.println("Spawning thread ...");
                InputStream iS = client.getInputStream();
                OutputStream oS = client.getOutputStream();
                ObjectOutputStream oOut = new ObjectOutputStream(oS);
                ObjectInputStream oIn = new ObjectInputStream(iS);
```

MirrorServer.java 3/3

```
            try {
                while (true) {
                    Point p = (Point) oIn.readObject();
                    p.mirror();
                    oOut.writeObject(p);
                    oOut.flush();
                }
            } catch (IOException e) { client.close(); }
            } catch (Exception e)
                { throw new Error(e.toString()); }
            System.out.println("... thread done.");
        } // run
    } // class MirrorHandler
} // class MirrorServer
```

Tarina oliovirroista, jotka menivät lukkoon

- Miksi edellisessä esimerkissä oliovirtojen luontijärjestyksellä oli merkitystä?
- ObjectInputStreamin konstruktori jää odottamaan, että tietyt virran otsikkotiedot luetaan virrasta ja vahvistetaan.
- Tämän takia kaksisuuntainen yhteys esimerkiksi asiakkaan ja palvelimen välillä voi aiheuttaa lukkiutuman.
- Jos ObjectInputStream luodaan jommallakummalla tai molemmilla puolilla ennen ObjectOutputStreamia, asiakkaan ObjectInputStreamin konstruktori jää odottelemaan otsikkotietoja.
- Samaan aikaan palvelimen puolella ObjectInputStreamin konstruktori odottelee samaa asiaa. On syntynyt lukkiutuma (deadlock).
- Tarinan opetus: ObjectOutputStream on luotava ennen ObjectInputStream-virran avaamista.

Multicast

- Monilähetys, useita vastaanottajia.
- Toimii IP-tasolla; eräät IP-osoitteet eivät ole oikeita koneiden osoitteita, vaan ns. multicast-ryhmien osoitteita. 224.0.0.1 – 239.255.255.255
- Lähettäjä ei eksplisiittisesti spesifioi vastaanottajia vaan vastaanottajien pitää liittyä multicast-ryhmään.
- Toiminta on DatagramSocket:n tapaista. Nyt käytetään aliluokkaa MulticastSocket. Lähetetään datagrammeja.

Luokka MulticastSocket

- **MulticastSocket()** - Luo multicast-soketin. Soketti sidotaan vapaaseen porttiin.
- **MulticastSocket(int port)** - Luo multicast-soketin sitoen sen porttiin *port*.
- **void setTimeToLive(int ttl)** - Soketin kautta lähtevät multicast-paketit saavat kulkea verkossa enintään *ttl* "hyppyä". $0 \leq ttl \leq 255$.
- **int getTimeToLive()** - Soketin TTL-arvo.
- **void joinGroup(InetAddress mcastaddr)** - Liittyy multicast-ryhmään *mcastaddr*.
- **void leaveGroup(InetAddress mcastaddr)** - Multicast-ryhmästä poistuminen.
- **Muita** operaatioita peritty luokasta DatagramSocket.

Esimerkki sokettien käytöstä: IRC-botti

- IRC (Internet Relay Chat): reaaliaikaisen keskustelun mahdollistava pikaviestintäpalvelu
- Jarkko Oikarinen, 1988
- Keskustelukanavia, joille voi liittyä (vrt. multicast)
- IRC-botti: IRC-kanavalla oleva normaalilta käyttäjältä päällisin puolin näyttävä tietokoneohjelma, joka suorittaa tiettyjä tehtäviä ja ottaa vastaan komentoja
- Melko vaivatonta tehdä protokollaa noudattava yksinkertainen ja toimiva asiakasohjelma

Yhteenveto

- Soketit ovat tiedostovirran tapaisia tietovirtoja verkon yli.
- Perustana IP-osoitteet, portit ja UDP & TCP.
- UDP-soketit
- TCP: asiakas- ja palvelinsoketit.
- Palvelinsovelluksen tulee voida palvella useita asiakkaita samanaikaisesti!
- Multicast: multicast-soketeilla.
- JDK 1.4: SSL-soketit luokalla SSLSocket + paljon muuta paketissa javax.net.