Föreläsning 7

- Client/Server
- InetAddress
- URL, URLConnection
- UDP

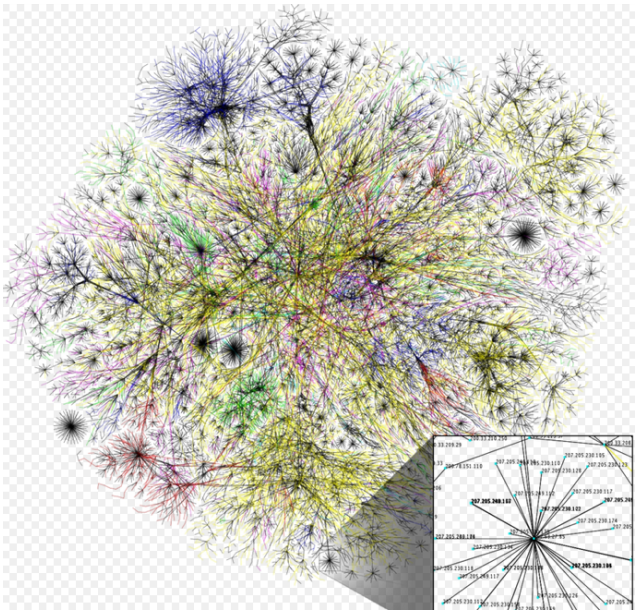
JNP: s 93-110, 117-141, 187-197, 393-416

Kommunikation över nätverk

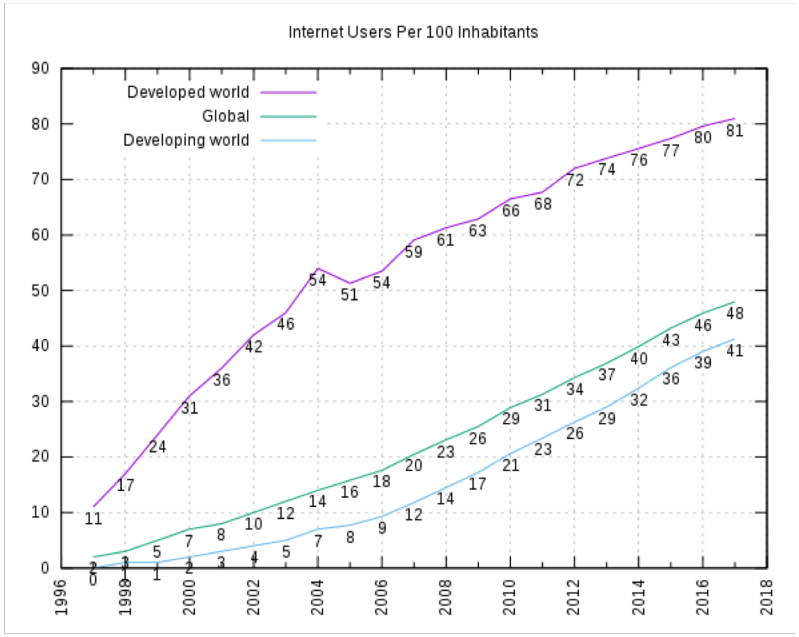
Det är vanligt att program kommunicerar via internet idag. De flesta system bygger på nätverkskommunikation.

Java är utvecklat med gott stöd för nätverkskommunikation. Det är enkelt att skriva program som utnyttjar internet.

Grafisk visualisering av internet



Användare av internet



Source: International Telecommunications Union

Nätverk

Ett nätverk är ett antal datorer och andra enheter som kan skicka och ta emot data från varandra.

Ett *protokoll* definierar hur datorer kommunicerar med varandra, t.ex. hur adressering sker och hur data förpackas.

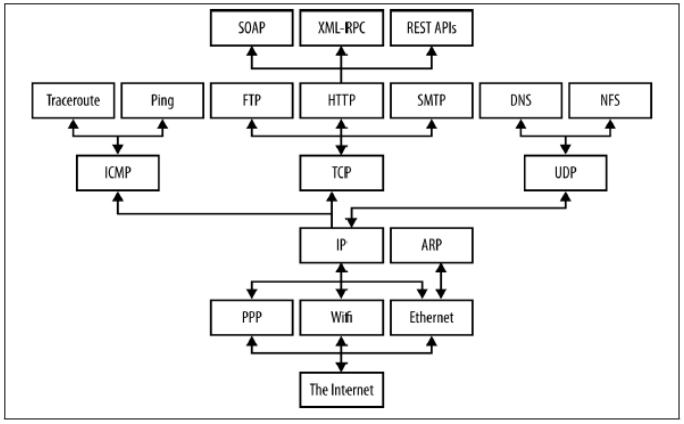
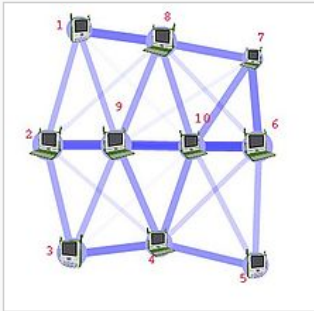


Figure 1-1. Protocols in different layers of a network

Varje dator i nätverket är en *nod* och varje nod har en *unik adress*.

Data skickas i nätverket som *paket*. Ett paket innehåller adress till mottagare och till avsändare.

Vilken väg ett paket tar från avsändare till mottagare beror bl.a. på belastning i olika delar av nätverket.



Routing calculates good paths through a network for information to take. For example from node 1 to node 6 the best routes are likely to be 1-8-7-6 or 1-8-10-6, as this has the thickest routes.

Nätverk

För att få nätverkskommunikation behövs programvara på flera nivåer, s.k. lager eller skikt.

Dator **Dator**

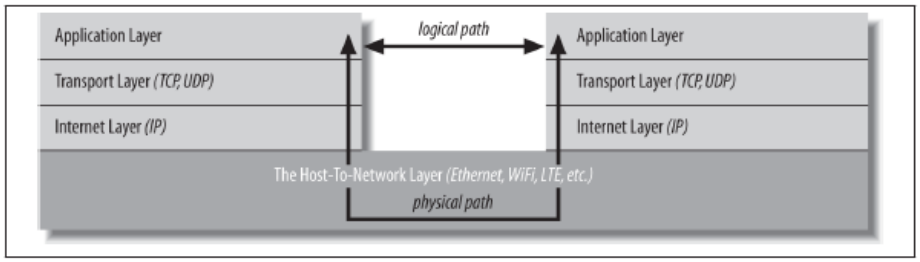


Figure 1-2. The layers of a network

Java
Programvaran är i applikations-skiktet på båda datorerna. Programvaran kommunicerar enbart med transportskiktet via standardklasser.

Applikationsskiktet – kommunikationen är abstraherad och programvaran hanterar dataflödet.

Transportskiktet – protokoll för pålitlig överföring av paket mellan avsändare och mottagare. *TCP* medger stor pålitlighet i dataöverföringen medan *UDP* lämnar till applikationsskiktet att kontrollera att dataöverföringen skett korrekt.

Internetskiktet – sköter adressering och routing av paket. *IP* används i denna modell

Fysiska skiktet – hårdvara och överföring av data mellan två noder

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).

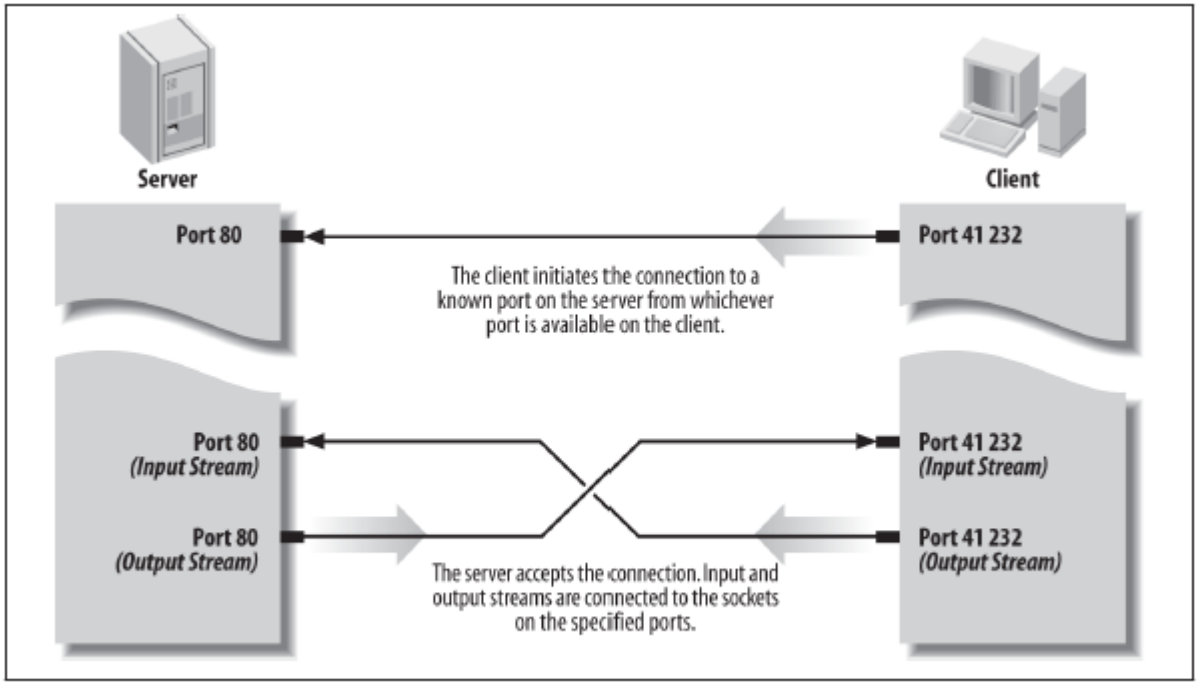


Figure 1-5. A client/server connection

InetAddress

Klassen ***InetAddress*** kapslar in adresseringen vid kommunikation. Varje nod i nätverket identifieras med en unik adress.

Aktuellt objekt är en instans av någon av subklasserna *Inet4Address* eller *Inet6Address* beroende på nätverket adressen ska användas i. Objektet är konstant (immutable) – dess innehåll kan inte ändras.

IPv4

Adressen består av 4 bytes, utskrivna med punkt mellan, t.ex.
92.122.190.114
4 bytes innebär ca 4 miljarder adresser vilket inte täcker dagens behov.

IPv6

Adressen består av 16 bytes utskrivna parvis hexadeximalt med kolon mellan, t.ex.
34E2:AFFE:1544:F553:A44F:100D:5C7A:BE72

Endast ett par procent av all internettrafik adresseras via IPv6 (dec 2016)

InetAddress, bl.a.

boolean	equals(Object obj) Compares this object against the specified object.
byte[]	getAddress() Returns the raw IP address of this InetAddress object.
static InetAddress[]	getAllByName(String host) Given the name of a host, returns an array of its IP addresses, based on the configured name service on the system.
static InetAddress	getByAddress(byte[] addr) Returns an InetAddress object given the raw IP address .
static InetAddress	getByAddress(String host, byte[] addr) Creates an InetAddress based on the provided host name and IP address.
static InetAddress	getByName(String host) Determines the IP address of a host, given the host's name.
String	getCanonicalHostName() Gets the fully qualified domain name for this IP address.
String	getHostAddress() Returns the IP address string in textual presentation.
String	getHostName() Gets the host name for this IP address.
static InetAddress	getLocalHost() Returns the address of the local host.
static InetAddress	getLoopbackAddress() Returns the loopback address.
int	hashCode() Returns a hashcode for this IP address.
boolean	isReachable(int timeout) Test whether that address is reachable.
boolean	isReachable(NetworkInterface netif, int ttl, int timeout) Test whether that address is reachable.
String	toString() Converts this IP address to a String.

InetAddress

Det kan vara bra att få reda på ip-adressen på datorn man sitter vid. Klassmetoden ***getLocalHost()*** returnerar ett InetAddress-objekt. Metoderna ***getHostAddress()*** och ***getHostName()*** ger ip-adressen och datornamnet.

```
public void myIP() {
    try {
        System.out.println("Denna dator:");
        InetAddress myAddress = InetAddress.getLocalHost();
        System.out.println(myAddress.getHostAddress());
        System.out.println(myAddress.getHostName());
    } catch (UnknownHostException ex) {
        System.out.println("Kan ej finna adressen till datorn");
    }
}
```

Ip-adressen till en annan dator ges av klassmetoden ***getByName(host)***:

```
InetAddress address = InetAddress.getByName("www.mah.se");
```

Metoden kan även användas tillsammans med en sträng med ip-adressen till den dator man önskar kommunicera med:

```
InetAddress address = InetAddress.getByName("195.178.227.58");
```

InetAddressEx1.java

URL

Klassen ***URL***, Universal Resource Locator, representerar adressen till en resurs, för vår del en fil, på internet.

En URL till en fil på internet kan se ut så här:

http://ddwap.mah.se/tsroax/expendituresxml.xml

http: - protokoll

ddwap.mah.se – domän

tsroax/expendituresxml.xml – sökväg och fil

Till filen *new1.jpg* på hårddisken kan URLen se ut så här:

file://C:/filer/new1.jpg

Protokollet är ändrat till *file*. Andra vanliga protokoll är t.ex. *ftp* och *https*.

Ett URL-objekt är immutable.

URL

Med ett URL-objekt går det att hämta data från en server.

Först skapar man ett URL-objekt med någon av konstruktorerna:

public URL(String spec)

```
URL url = new URL("http://ddwap.mah.se/tsroax/expendituresxml.xml");
```

public URL(String protocol, String host, String file)

```
URL url = new URL("http", "ddwap.mah.se", "/tsroax/expendituresxml.xml");
```

public URL(String protocol, String host, int port, String file)

```
URL url = new URL("http", "ddwap.mah.se",7000, "/tsroax/expendituresxml.xml");
```

Om servern inte lyssnar på default port.

public URL(URL context, String spec)

```
URL url = new URL( new URL("http://ddwap.mah.se"), "/tsroax/expendituresxml.xml");
```

URL – hämta bild

- Klassen ***ImageIcon*** har konstruktor som tar en URL som argument: ***public ImageIcon(URL location)***
- Med metoden ***openStream()*** erhåller man en inputstream. Ur denna kan man läsa strömmen tecken för tecken.

```
public class URLEx1 {
   // spara bildfil till hårddisken
   private static void saveToFile(URL source, String filename) {
      try( BufferedInputStream bis = new BufferedInputStream(source.openStream());
           BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(filename))) {
         InOut.copy(bis, bos);
      } catch(IOException e) {
         System.err.println(e);
      }
   }

   // visa bild som ImageIcon
   private static void showAsImageIcon(URL source) {
      ImageIcon icon = new ImageIcon(source);
      JOptionPane.showMessageDialog(null, icon);
   }

   public static void main(String[] args) throws MalformedURLException {
      URL url = new URL("http://ddwap.mah.se/tsroax/memory/Memory.jpg");
      showAsImageIcon( url );
      saveToFile( url, "files/Memory.jpg");
   }
}
```

URLImage.java

InOut.java

URL – hämta text

- Med metoden ***openStream()*** erhåller man en inputstream. Ur denna kan man läsa strömmen tecken för tecken.
- System.out är en outputstream med Konsol-fönstret som mål

```
public class URLText {
    private static void showInConsole(URL source) {
        try( InputStreamReader isr = new InputStreamReader(source.openStream(),"UTF-8");
             BufferedReader br = new BufferedReader(isr);
             OutputStreamWriter osw = new OutputStreamWriter(System.out);
             BufferedWriter bw = new BufferedWriter(osw)) {
           System.out.println("Text från " + source.toString());
           System.out.println("--- Läsning tecken för tecken ---");
           InOut.copy(br,bw,1000);
//           System.out.println("------ Läsning rad för rad ------");
//           InOut.copy(br,bw);
       } catch (IOException ex) {
           System.err.println(ex);
       }
   }
   public static void main(String[] args) throws MalformedURLException {
       URL u = new URL("http://ddwap.mah.se/tsroax/expendituresxml.xml");
       showInConsole( u );
   }
}
```

URLText.java

InOut.java

URL - URLConnection

Med metoden ***openConnection()*** i klassen ***URL*** erhåller man som resultat ett ***URLConnection***-objekt.

```
URL url = new URL(”http://ddwap.mah.se/tsroax/expendituresxml.xml");
URLConnection conn = url.openConnection();
```

Detta objekt ger lite mer information om filen som ska hämtas. Men kvaliteten på informationen beror på servern.

- Metoden ***getContentType()*** returnerar MIME-typen, t.ex. text/plain, text/xml, text/html, image/jpg, application/xml osv. (kan returnera null)
- Metoden ***getContentLength ()*** returnerar filens storlek i bytes. (kan returnera -1)
- Metoden ***getDate()*** ger tidpunkten då filen skickades (kan returnera 0)
- Metoden ***getLastModified()*** ger tidpunkten då filen ändrades senast. (kan returnera 0)

Med ett *URLConnection*-objekt kan man erhålla en inputstream med metoden ***getInputStream()***.
Det går även att skriva till servern med outputströmmen som man erhåller med ***getOutputStream()***. Detta kan t.ex. vara aktuellt vid kommunikation med webserver.

URLConnectionEx

UDP - Datagram

Man kan kommunicera med **Datagram** över nätverk. Men denna kommunikation garanterar ej att

- innehållet är oförändrat när det kommer fram.
- datagrammet når mottagaren.
- datagram kommer fram i samma ordning som de sänts.

En dator skickar ett paket, vilket innehåller avsändaradress, mottagaradress och någon form av data. Datorn med mottagaradressen tar emot paketet. Kommunikationen sker över bestämda portnummer.

Både avsändare och mottagare av ett paket behöver ett objekt av typen ***DatagramSocket*** och ett objekt av typen ***DatagramPacket***. Med ***send()***-metoden skickar avsändaren paketet och med ***receive()***-metoden tar mottagaren emot paketet.

UDP - DatagramSocket

Konstruktorer (throws SocketException)

- **public DatagramSocket(int port)**
  Skapar en socket som använder angiven port (1025-65535).
- **public DatagramSocket()**
  Skapar en socket som använder en ledig port.

Metoder

- **public void close()**
  Avslutar socketen.
- **public void receive(DatagramPacket) throws IOException**
  Tar emot datagram. Programmet väntar tills ett meddelande.
- **public void send(DatagramPacket) throws IOException**
  Skickar datagram.
- **public int getLocalPort()**
  Returnerar den port som socketen använder
- **public void setSoTimeout(long millisek)**
  Bestämmer den tid som **receive** ska vänta på ett paket.

UDP - DatagramPacket

Konstruktorer

- **public DatagramPacket(byte[], bytesToSend, InetAddress, port)**
  Skapar ett paket som man använder för att sända.
- **public DatagramPacket(byte[], bytesToSend)**
  Skapar ett paket som man använder för att mottaga.

Metoder

- **public InetAddress getAddress()**
  Returnerar InetAddressen till avsändaren av datagrammet
- **public byte[] getData()**
  Returnerar data-innehållet som ett byte-fält
- **public int getLength()**
  Returnerar längden på byte-fältet
- **public int getPort()**
  Returnerar portnummer hos avsändaren av datagrammet

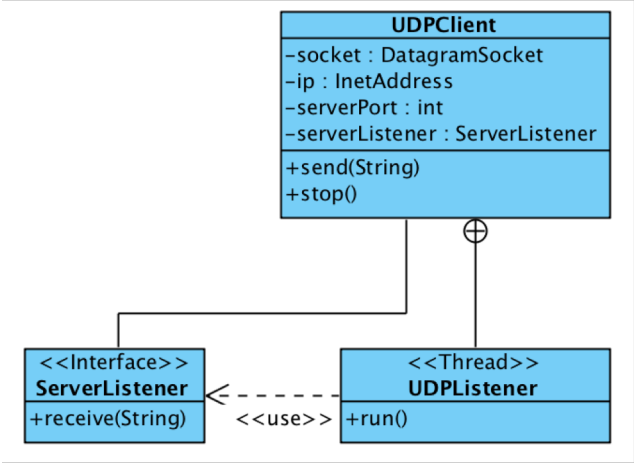
UDP - Klient

```
InetAddress to = InetAdress.getByName(”232.244.50.21”); // ip till server
DatagramSocket socket = new DatagramSocket();    // använd en ledig port
:

// Att skicka
String message=”Meddelande att skicka”;
byte [] data = message.getBytes();
DatagramPacket packet = new DatagramPacket(data, data.length, to, 2345);
try {
   socket.send(packet);
} catch(IOException e) {}

 // Att ta emot
private class UDPListener extends Thread {
   public void run() {
      DatagramPacket packet;
      byte[] readBuffer = new byte[1024];
      while(!Thread.interrupted()) {
         try {
            packet = new DatagramPacket(readBuffer, readBuffer.length);
            socket.receive(packet);
            String response = new String(packet.getData(),0,packet.getLength());
            serverListener.receive(response);
         } catch(SocketTimeoutException e) {
         } catch(Exception e) { // InterruptedException, IOException
            break;
         }
      }
      socket.close();
      socket = null;
   }
}
```

UDPClient

// Konstruktor

```
public UDPClient(String ipAddress, int port, ServerListener serverListener) {
    this.serverPort = port;
    this.serverListener = serverListener;
    try {
        this.address = InetAddress.getByName(ipAddress);
        this.socket = new DatagramSocket();  // ledig port
        this.setSoTimeout( 100 );
        this.listener.start();
    } catch(Exception e) {}
}
```



// send

```
public void send(String str) {
    if(socket!=null) {
        byte[] data = str.getBytes();
        DatagramPacket packet = new DatagramPacket(data,data.length,address,serverPort);
        try {
            socket.send(packet);
        } catch (IOException e) {}
    }
}
```

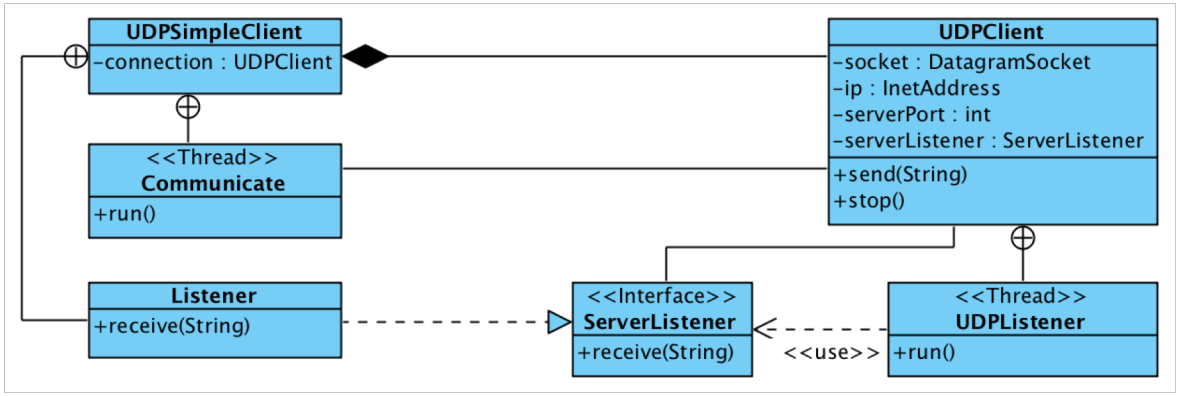
// Att ta emot

```
private class UDPListener extends Thread {
    public void run() {
        // se föregående sida
    }
}
```

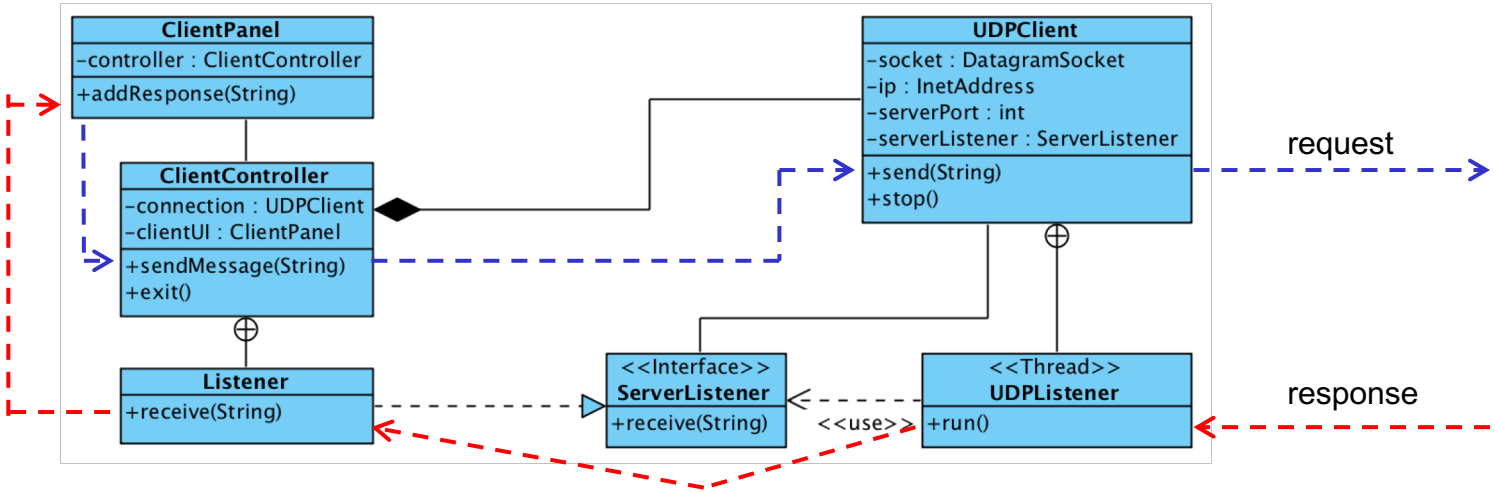
UDPClient

UDPClient

TextClient



GUIClient



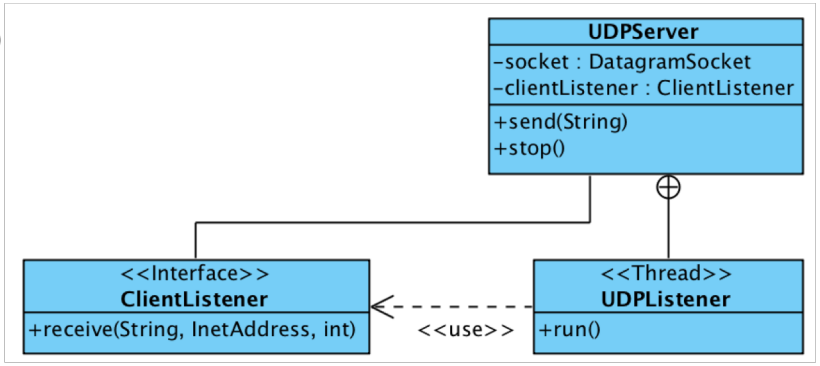
f7.udpclient1

UDP - Server

```
DatagramSocket socket = new DatagramSocket(2345); // port 2345 lyssnar servern på
DatagramPacket packet;

// Ta emot request
String request;
byte[] data = new byte[1024]; // tillräckligt stor buffert
packet = new DatagramPacket(data, data.length);
socket.receive(packet);  // blocking
request = new String(packet.getData(), 0, packet.getLength());

// Skicka response
String response = …;
byte[] data = response.getBytes();
packet = new DatagramPacket(data, data.length, packet.getAddress(), packet.getPort());
socket.send(packet);
```

UDPServer

```
// Konstruktor
public UDPServer(int requestPort, ClientListener clientListener)
   this.clientListener = clientListener;
   try {
      socket = new DatagramSocket(requestPort);
      socket.setSoTimeout( 100 );
      listener.start();
   } catch(Exception e) {}
}
```

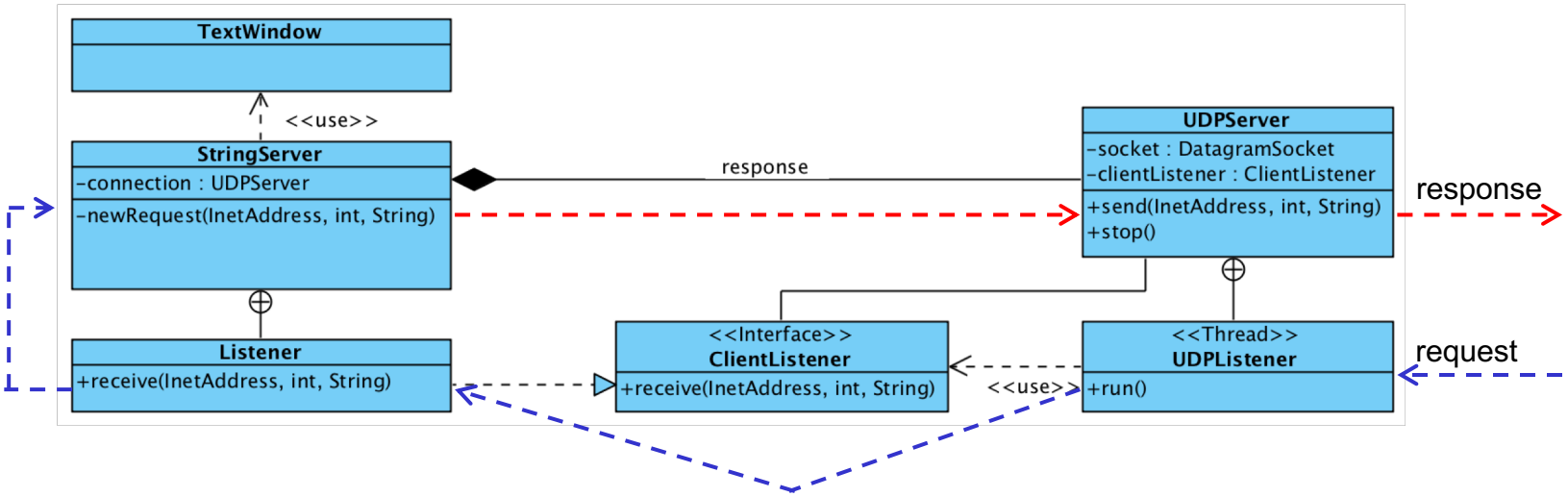



```
// Ta emot request
private class UDPListener extends Thread {
   public void run() {
      DatagramPacket packet;
      byte[] readBuffer = new byte[1024];
      while(!Thread.interrupted()) {
         try {
            p = new DatagramPacket(readBuffer,readBuffer.length);
            socket.receive(p);
            clientListener.receive(new String(p.getData(),0,p.getLength()), p.getAddress(), p.getPort());
         } catch(…) { // samma som i UDPClient
         }
      }
   }
}
```

```
// Skicka response
public void send(InetAddress address, int clientPort, String str) {
   if(socket!=null) {
      byte[] data = str.getBytes();
      DatagramPacket packet = new DatagramPacket(data,data.length,address,clientPort);
      try {
         socket.send(packet);
      } catch (IOException e) {}
   }
}
```

UDPServer

StringServer



f7.udpserver1