F1
Nätverk och strömmar

EDA095 Nätverksprogrammering

Roger Henriksson
Datavetenskap
Lunds universitet

Nätverksprogrammering

- Tekniker för att skriva program som kommunicerar med varandra över ett nätverk.
- Särskild tonvikt på Internet/IP-baserade nätverk, men generellt tillämpbara tekniker.
- Praktisk implementering i Java. Samma principer som i andra programmeringsspråk.

Kursöversikt

- Nätverksbegrepp
- Strömmar
- Accessa dokument på webben
- Trådar
- Socket-baserad kommunikation: TCP, kryptering
- Webbtekniker:
  - HTTP
  - XML
  - CGI/Servlets/JSP/PHP
- Strömmande media
- Paketbaserad kommunikation: UDP, multicast

Klienter och servrar

Vanlig arkitektur för applikationer som erbjuder tjänster via nätverk. Engelska: “client/server architecture”.

Klient

Ett program som kopplar upp sig (ofta via nätverk) till ett annat program för att begära någon tjänst.

Exempel: Webbläsare, e-postprogram.

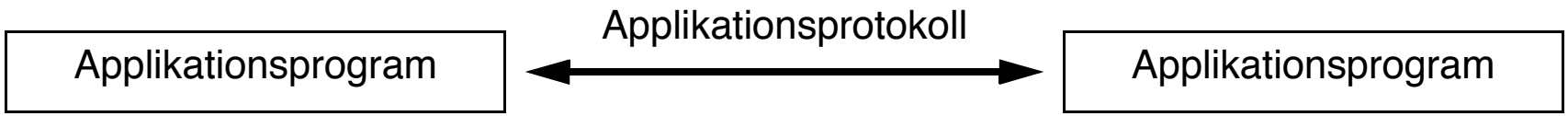
Server

Ett program som erbjuder tjänster åt andra program.

Exempel: Webbserver, mailserver.

Protokoll

För att två program ska kunna förstå varandras meddelanden behövs ett *protokoll*, dvs ett gemensamt språk.



Exempel

HTTP (HyperText Transfer Protocol) - för kommunikation mellan webbläsare och webbserver.

Transaktioner

Kommunikationen mellan en klient och en server kan ofta delas upp i en följd av *transaktioner*.

En transaktion kan omfatta sändning av ett antal relaterade meddelanden i olika riktningar.

Syftet med en transaktion är att få en primitiv – typiskt *atomär* – operation utförd av servern.

Exempel

Hämta en webbsida från en webbserver (med HTTP):

- Klienten skickar en begäran till webbservern.
- Servern skickar den begärda webbsidan.

Session

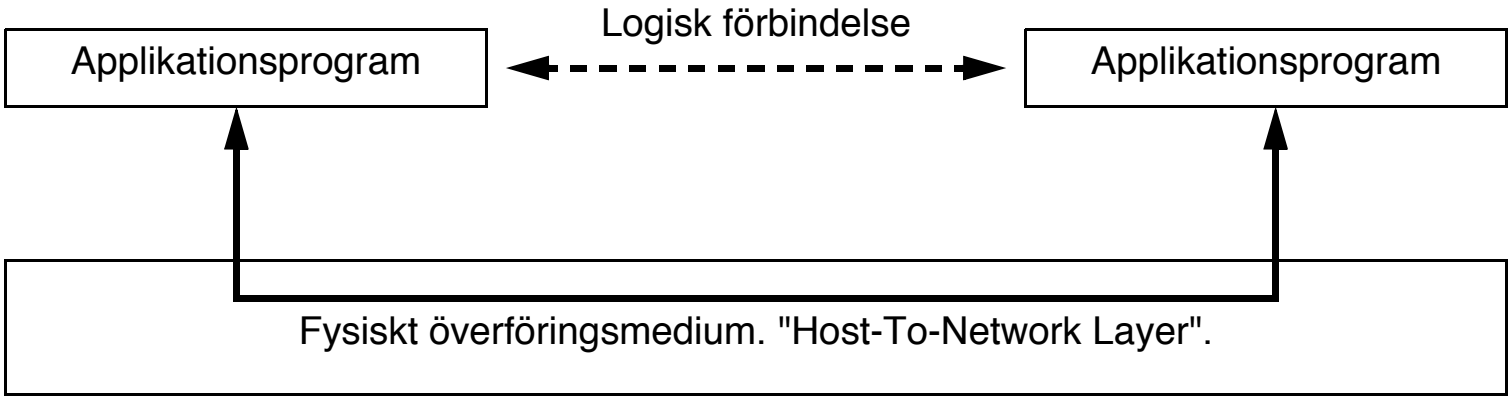
Hela "uppkopplingen", ofta flera transaktioner.

Fysiskt överföringsmedium

Applikationsprotokollet är ett protokoll för kommunikation mellan två applikationsprogram.

För överföringen behövs ett fysiskt överföringsmedium.

Exempel: Ethernet, RadioLAN.



"Host-To-Network"-lagret omvandlar mellan analoga och digitala signaler.

Datagram

Ett datagram är ett sammanhållet meddelande som sänds som en enhet över ett nätverk.

Jämför: Telegram, SMS.

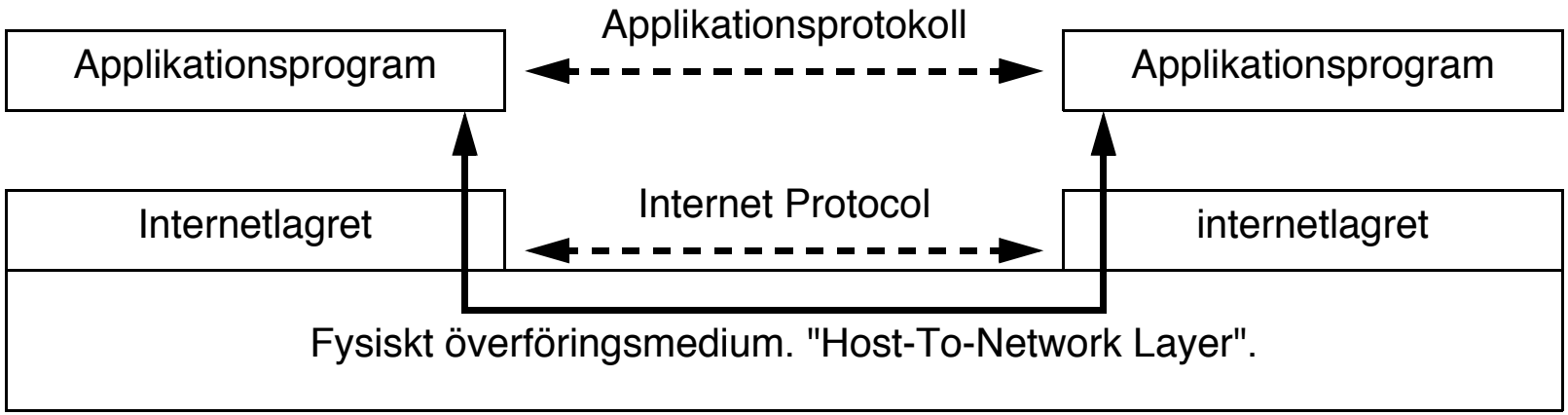
”Host-To-Network”-lagret

- Endast begränsade möjligheter att skicka datagram.
- Ingen felkontroll/omsändning.
- Många standarder.

Opraktiskt!

Internet Layer

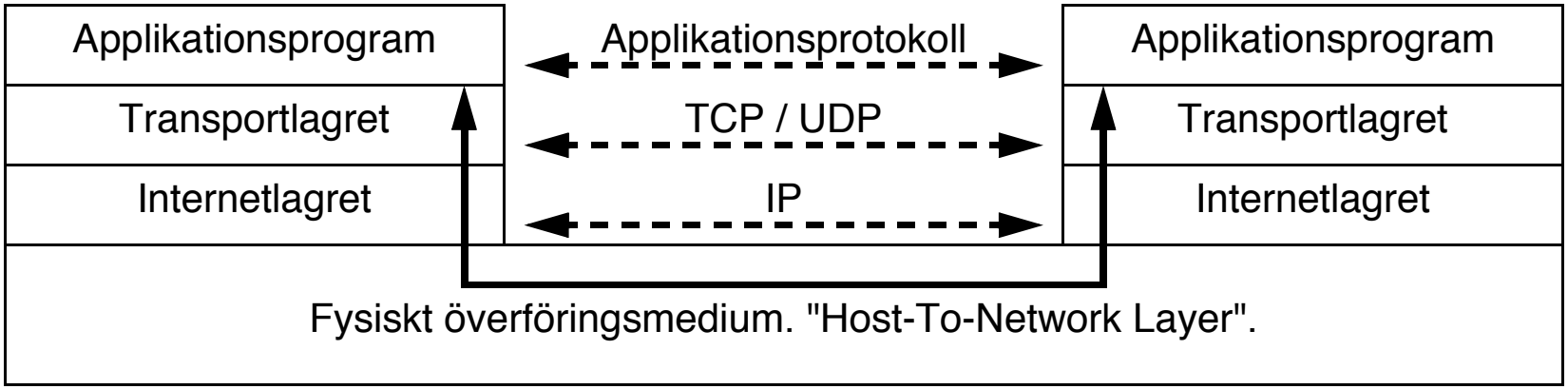
- IP - Internet Protocol
- Viss felkontroll
- Oberoende av fysiskt överföringsmedium



Bättre, men fortfarande komplicerat!

Transport Layer

Bygger vidare på “Internet Layer” / IP.



TCP/UDP

- **User Datagram Protocol (UDP)**
  - Datagram
  - Upp till 65507 byte stora datagram (IPv4)
  - Garanterat korrekta meddelanden.
  - Ej garanterad leverans eller inbördes ordning.
- **Transmission Control Protocol (TCP)**
  - Fast (logisk) uppkoppling över nätverket.
  - Dataström – data behöver inte delas upp i paket.
  - Automatisk felkontroll/omsändning. Garanterad leverans och inbördes ordning.

Adressering

För att ett program ska kunna upprätta en nätverksförbindelse behöver det ett sätt att ange vilket annat program det vill koppla upp sig mot.

En kombination av IP-nummer och portnummer!

IP-nummer

Anger vilken dator programmet kör på. Jämför: gatuadress.

Portnummer

En dator kan ha flera olika program igång som erbjuder nätverkstjänster. Varje program har ett eller flera unika logiska portnummer. Jämför: lägenhetsnummer inom en fastighet.

IP-nummer

- 32 bitar - 4 byte (IPv4)
- 128 bitar - 16 byte (IPv6)

Exempel: 130.235.16.34 (IPv4)

2001:fe0c:0000:0000:0000:0000:00db:1dc0 (IPv6)

2001:fe0c::db:1dc0 (IPv6)

Symboliska (domän-)namn istället:

www.cs.lth.se = 130.235.16.34

Översättning mellan domännamn och IP-adresser görs av en "Domain Name Server" – DNS.

Övergång till IPv6

- IP-numren i IPv4 slut!
- IPv4 / IPv6 kommer att användas parallellt ett tag
- Dubbla IP-adresser under övergångsfasen
- Moderna datorer stöder IPv6
- Idag dåligt stöd bland internetleverantörer

- Liten eller ingen betydelse för oss som applikationsprogrammerare – undvik bara att förutsätta IP-adresser på formen ”X.X.X.X”!

Portnummer

- Identifierar vilken tjänst på en dator man vill ansluta till.
- Rent logisk abstraktion – ingen fysikalisk motsvarighet.

Ett program som accepterar nätverksuppkopplingar, en server, väljer ett ledigt portnummer och väntar på att andra program ska ansluta till just detta portnummer.

Port 1-1023 är reserverade för standardtjänster.

Exempel:

Tjänst	Port	Protokoll
echo	7	TCP/UDP
FTP	21	TCP
Telnet	23	TCP
HTTP	80	TCP

Port 1024-65535 är tillgängliga för vanliga applikationer.

Internetstandarder

Standardprotokoll för kommunikation på Internet fastställs av IETF – Internet Engineering Task Force – en ganska informell organisation.

Beskrivs i form av RFC – Request For Comments.

Exempel:

- RFC768 – UDP
- RFC791 mfl – IP
- RFC1945 – HTTP version 1.0

Java I/O – Strömmar och filer

Ström (eng. Stream)

En ström är en sekvensiell följd av bytes (tecken).

In- och utmatning sker oftast i form av strömmar: Inmatning från tangentbordet, utskrift till ett terminalfönster.

Vi kan skriva bytes till strömmar och vi kan läsa bytes från strömmar. Generell abstraktion.

Vi kan upprätta strömmar över ett nätverk: TCP

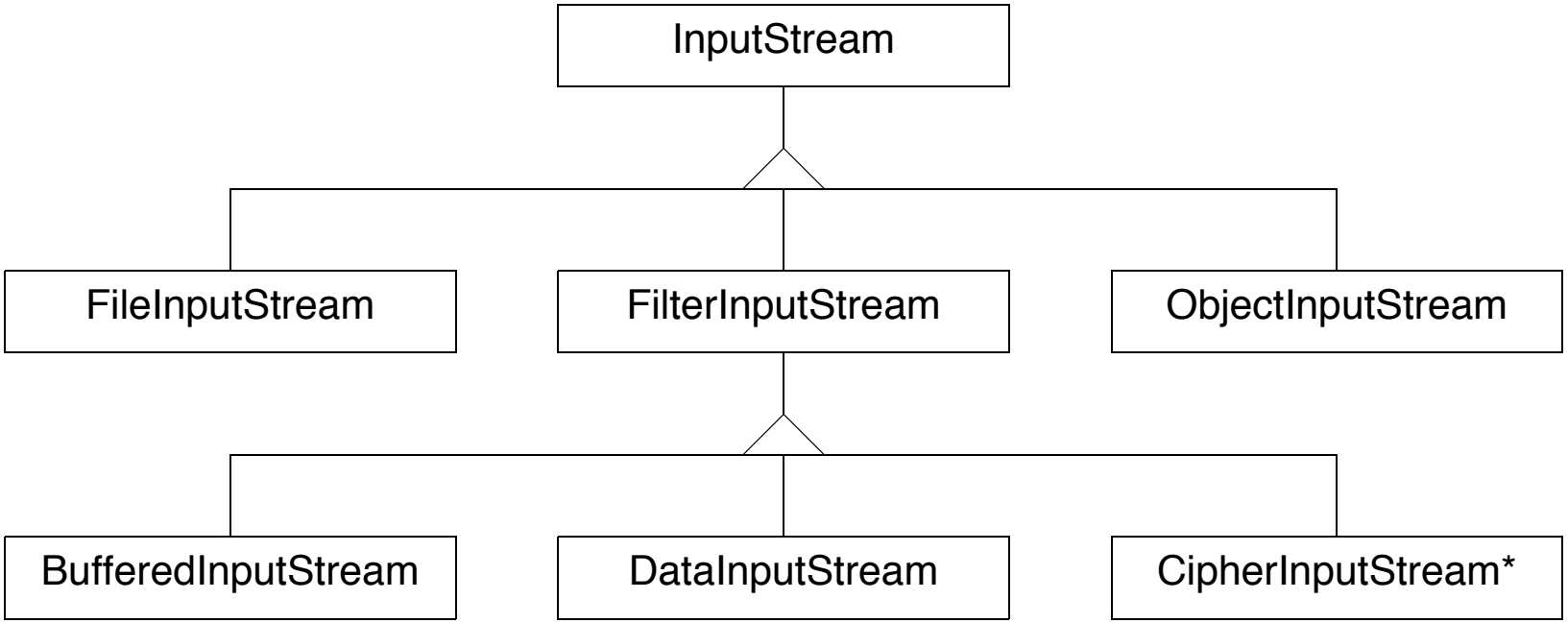
Fil (eng. File)

En fil är en ström som lagrats på ett sekundärminne.

Exempel: Era Javaprogram är lagrade i textfiler.

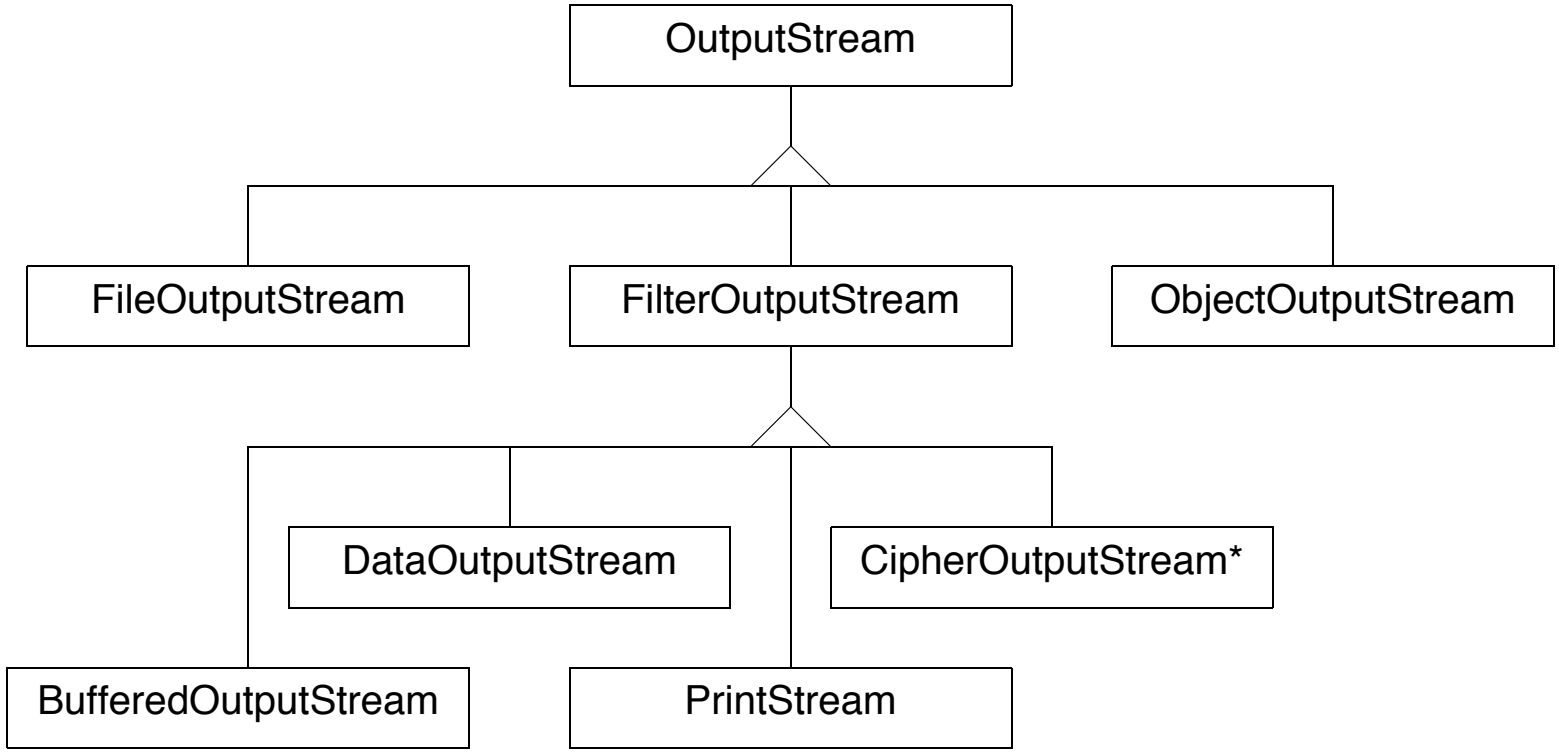
Strömmar i Java – inmatning

Klasserna för att hantera strömmar finns i paketet java.io.



Endast en delmängd...

Strömmar i Java – utmatning



Endast en delmängd...

InputStream

Abstrakt klass som representerar en inkommande ström.

Superklass till övriga InputStream-klasser.

Innehåller metoder för att läsa bytes.

```
public int abstract read() throws IOException;
public int read(byte[] input) throws IOException;
public int read(byte[] input,int offset,int length)
                                        throws IOException;

public long skip(long n) throws IOException;

public int available() throws IOException;

public void close() throws IOException;
```

Läsa en sekvens av bytes – 1

Läs en sekvens av 10 bytes!

```
byte[] input = new byte[10];
for (int i=0;i<input.length;i++) {
        int b = in.read();
        if (b==-1) break;
        input[i] = (byte) b;
}
```

- `b==-1` anger att det inte finns mer att läsa.
- `read()` returnerar en int (-1 … 255) – cast behövs.
- Ineffektivt att läsa en byte i taget.

Läsa en sekvens av bytes – 2

Varför inte använda `int read(byte[] input)`?

```
byte[] input = new byte[10];
int r = in.read(input);
```

- Läser max `input.length` tecken
- Garanterar ej att hela `input` fylls.
- r anger hur många tecken som faktiskt lästes.

Läsa en sekvens av bytes – 3

Använd `int read(byte[] input,int offset,int length)`!

```
byte[] input = new byte[10];
int bytesRead = 0;
int bytesToRead = input.length;
while (bytesRead<bytesToRead) {
      int result=in.read(input,bytesRead,
                         bytesToRead-bytesRead);
      if (result==-1) break;
      bytesRead += result;
}
```

Mark/Reset

Vissa strömmar stödjer att man läser om redan inlästa bytes.

```
public void mark(int readAheadLimit);
public reset() throws IOException;
public boolean markSupported();
```

- `mark()` sätter en markering – `reset()` går tillbaka till denna position.
- `readAheadLimit` anger buffertstorlek.

OutputStream

Abstrakt klass som representerar en utgående ström.

Superklass till övriga OutputStream-klasser.

Innehåller metoder för att skriva bytes.

```
public abstract void write(int b) throws IOException;
public void write(byte[] data) throws IOException;
public void write(byte[] data, int offset, int length)
                                        throws IOException;

public void flush() throws IOException;

public void close() throws IOException;
```

Filter

Objekt av subklasserna till FilterInputStream och FilterOutputStream kan kopplas ihop (kedjas ihop) med InputStream- respektive OutputStream-objekt för att utöka deras funktionalitet.

Exempel:

Använd en BufferedInputStream för att öka prestanda vid inläsning från fil:

```
InputStream is = ...;
BufferedInputStream bis = new BufferedInputStream(is);
```

Motsvarande för utmatning:

```
OutputStream os = ...;
BufferedOutputStream = new BufferedOutputStream(os);
```

Buffring

Buffring av strömmar ökar effektiviteten – högre genomströmning per tidsenhet.

Ju mer data som sänds / tas emot åt gången ju färre underliggande systemanrop / diskaccesser / nätverkspaket.

Klasserna BufferedInputStream / BufferedOutputStream implementerar buffring.

VARNING: Glöm inte att anropa flush()!

Risk för dödläge:

Meddelanden kan fastna i bufferten!

Data(Input/Output)Stream

Subklasser till FilterInputStream/FilterOutputStream.

Kan användas om man vill skriva annat än bytes på en ström. De konverterar mellan bytes och olika andra datatyper.

```
public void writeBoolean(boolean b);
public void writeByte(int b);
public void writeShort(int s);
public void writeChar(int c);
public void writeInt(int i);
public void writeLong(long l);
public void writeFloat(float f);
public void writeDouble(double d);
public void writeChars(String s);
public void writeBytes(String s);
public void writeUTF(String s);
```

```
public boolean readBoolean();
public byte readByte();
public char readChar();
public short readShort();
public int readInt();
public long readLong();
public float readFloat();
public double readDouble();
public String readUTF();
```

Alla operationer kan generera ett IOException.

```
DataInputStream dis = new DataInputStream(is);
int x = dis.readInt();
```

Skicka/ta emot tecken

Ibland vill man skicka tecken istället för bytes och vill slippa att själv göra omvandlingen i sitt program.

Reader/Writer - en parallell klasshierarki till InputStream/OutputStream som hanterar tecken enligt en given teckenkodning.

Vi gör om strömmar till en reader/writer genom att kapsla in den (precis som med FilterInput/OutputStream):

```
OutputStreamWriter out = new
OutputStreamWriter(socket.getOutputStream());
InputStreamReader in = new
                  InputStreamReader(socket.getInputStream());
```

Se kursboken, kapitel 2, och Javas klassdokumentation!

Skicka/ta emot strängar

Klasserna BufferedReader och PrintWriter representerar sträng- och textorienterade strömmar.
De är buffrade och hanterar rader av tecken effektivt via metoden `readLine()`:

```
InputStream is = ...
OutputStream os = ...
BufferedReader in = new
                BufferedReader(new InputStreamReader(is));
PrintWriter out = new PrintWriter(os,true);
```

Se kursboken, kapitel 2, och Javas klassdokumentation!