

Kapittel 14: Filer og strømmer

Redigert av:

Khalid Azim Mughal (khalid@ii.uib.no)

Kilde:

Java som første programmeringsspråk (3. utgave)

Khalid Azim Mughal, Torill Hamre, Rolf W. Rasmussen

Cappelen Akademisk Forlag, 2006.

ISBN: 82-02-24554-0

<http://www.ii.uib.no/~khalid/jfps3u/>

(NB! Boken dekker opptil Java 6, men notatene er oppdatert til Java 7.)

Emneoversikt

-
- Strømmer
 - Filer
 - Dataposter
 - Tekstfiler
 - try-blokk med automatisk ressurs håndtering
 - Terminal I/O
 - Binære filer
 - Objektserialisering
 - Direkte filtilgang
-

Inn- og ut-data

- *Inndata*: Data som et program leser utenfra.
 - Inndata kommer fra en *kilde* som produserer dataene.
 - F.eks. tastatur, en fil
- *Utdata*: Data som et program skriver ut
 - Utdata skrives ut til et *mål* som kan motta dataene.
 - F.eks. skjerm, en fil
- En *datafil* tilbyr permanent lagring av data fra et program på et eksternt medium.

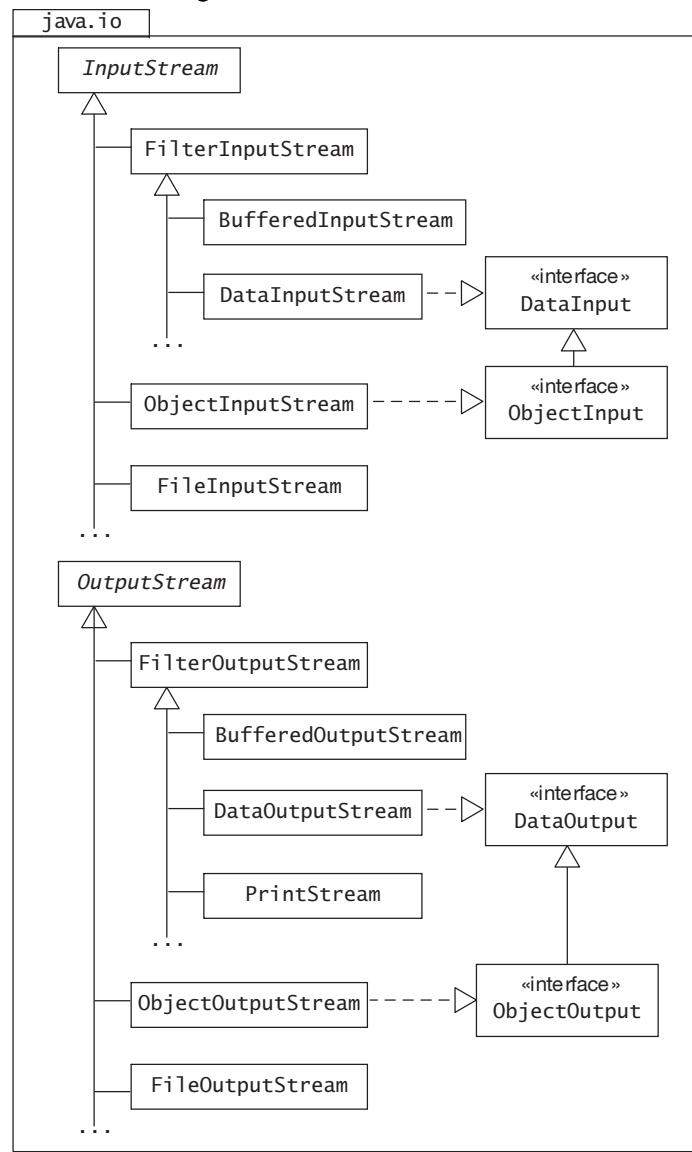
Strømmer

- En *strøm* er et objekt som et program kan bruke til å skrive data til et mål (*ut-strøm*) eller lese data fra en kilde (*inn-strøm*).
- *Sekvensielle strømmer*: Data kan bare leses eller skrives med én verdi om gangen, dvs. som en *sekvens*.

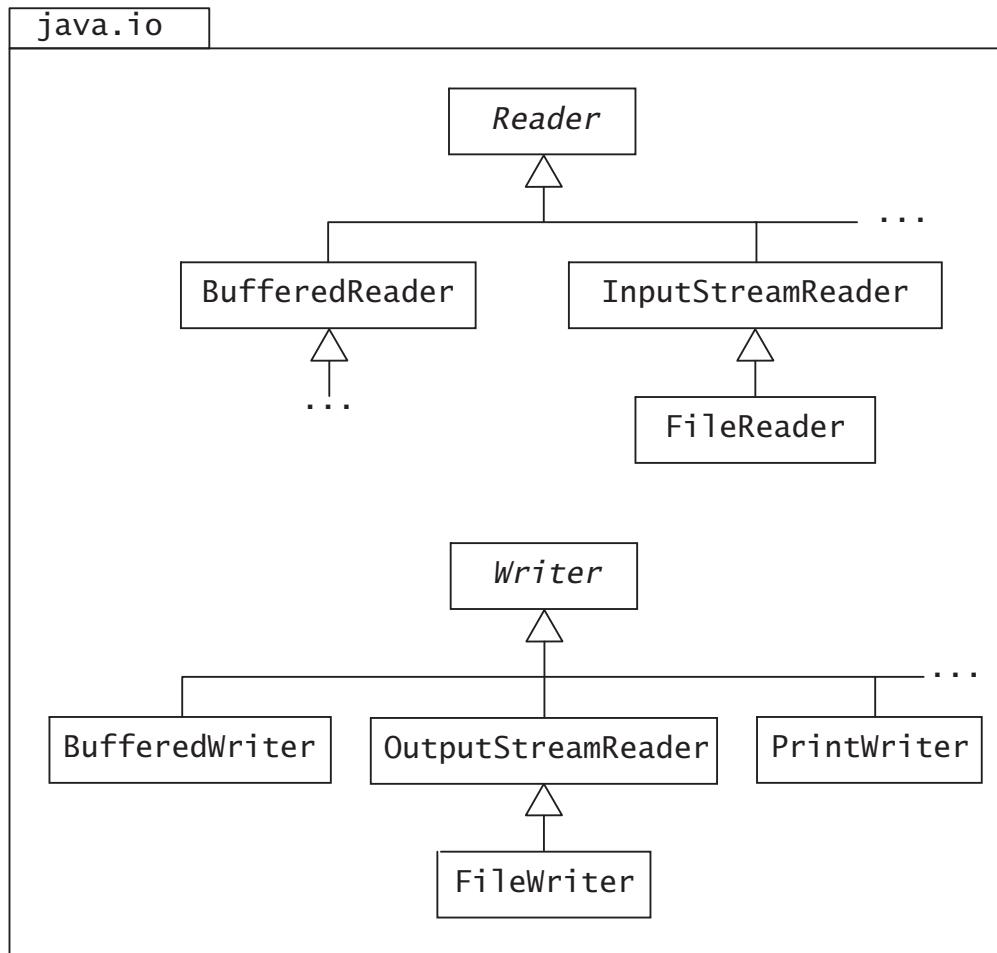
Bytestrømmer og tegnstrømmer

- *Bytestrømmer* håndterer sekvenser av bytes.
 - Data bestående av åtte bits.
- *Tegnstrømmer*: håndterer sekvenser av tegn.
 - Data bestående av 16-bits Unicode-tegn.

Bytestrømmer



Tegnstrømmer



Filbehandling

- En *fil* betegner et spesifikt lagerområde på et eksternt medium, for eksempel harddisk, der informasjon er lagret.
 - Data i filer blir lagret som en sekvens av *bytes*.
- *Binærfil*: Data tolket som en sekvens av bytes.
- *Tekstfil*: Data tolket som en sekvens av tegn.

Standard lengde for predefinerte datatyper (Tabell 14.1)

Predefinerte typer	Antall bytes
boolean	1
char	2
int	4
long	8
float	4
double	8

Filsti

- En *filsti* identifierer en fil i filsystemet:

```
String datafilnavn = "ansattfil.dat"; // Filsti angir filen.
```

```
String filsti1 = "firma\\ansattfil.dat"; // Windows
```

```
String filsti2 = "firma/ansattfil.dat"; // Unix
```

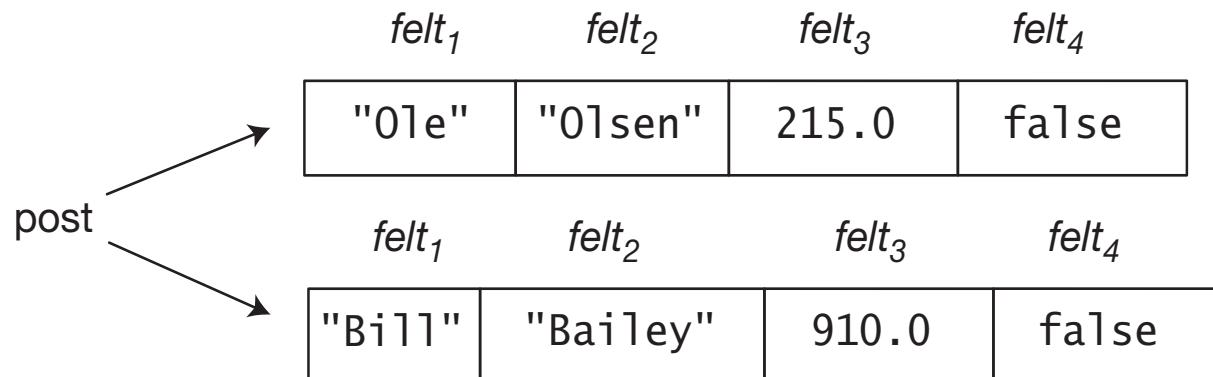
```
String filnavn = "firma" + File.separator + "ansattfil.dat"; // Plattformuavhengig
```

Dataposter (Figur 14.3)

- Problemstilling: lagre opplysninger om ansatte (Program 14.1) på en fil.

```
String fornavn;  
String etternavn;  
double timelønn;  
boolean kjønn;
```

- En *post* er en oppstilling av ett eller flere datafelt.
- Et *felt* består vanligvis av en primitiv verdi, men en streng kan inngå som en feltverdi.



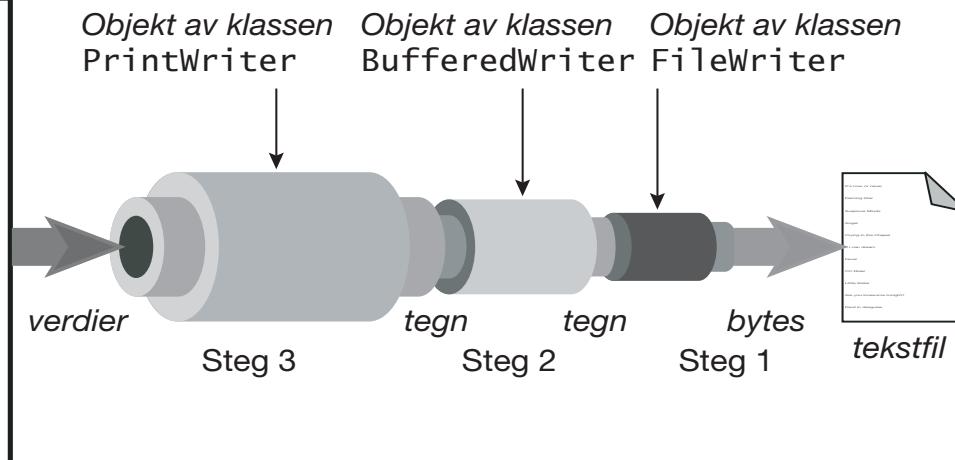
- Se Program 14.1: Klassene Ansatt, PersonellRegister og Firma.

Tekstfiler

- En *tekstfil* består av tekstlinjer.
- En *tekstlinje* består av en sekvens av tegn avsluttet med en *linjeslutt-streng*.
 - Metodene i Java sørger for at denne strengen blir tolket riktig.
- Løsningen for å håndtere tegn mellom Java-program og omverdenen:
 - Bytes som er lest av byte-inn-strømmen blir oversatt til Unicode-tegn av tegn-inn-strømmen.
 - Unicode-tegn (som skal skrives) oversettes til bytes av tegn-ut-strømmen og blir skrevet ut av byte-ut-strømmen.

Utskrift til en tekstfil

```
print(boolean b)  
print(char t)  
print(char[] tTab)  
print(double d)  
print(float f)  
print(int i)  
print(long l)  
print(Object obj)  
print(String str)  
println()  
println(...)  
printf(...)  
...
```



```
FileWriter tekstFilSkriver = new FileWriter(tekstfilnavn,utvid); // Steg 1  
BufferedWriter bufretSkriver = new BufferedWriter(tekstFilSkriver); // Steg 2  
PrintWriter tekstSkriver = new PrintWriter(bufretSkriver); // Steg 3
```

Utskrift til tekstfiler (fort.)

- Se filen `FirmaTxt.java`.
1. Definer et `FileWriter`-objekt som åpner filen det skal skrives til:

```
FileWriter tekstuSkriver = new FileWriter(datafilnavn, utvid); // (3)
```

 - Filen opprettes dersom den ikke finnes og har skrive-rettigheter.
 - Den andre parameteren angir om filinnhold skal overskrives eller utvides.
 2. Definer et `BufferedWriter`-objekt som kobles til `FileWriter`-objektet:

```
BufferedWriter bufretSkriver = new BufferedWriter(tekstuSkriver); // (3a)
```

 - Tegn blir bufret internt for effektiv skriving til filen.
 3. Definer et `PrintWriter`-objekt som kobles til `BufferedWriter`-objektet:

```
PrintWriter tekstuSkriver = new PrintWriter(bufretSkriver); // (4)
```

 - Java-verdier blir konvertert til tegn-representasjon.

4. Skriv verdier ut som tekst med `print()`-metoder definert i `PrintWriter`- klassen.

```
void skrivAnsattData(PrintWriter tekstSkriver, Ansatt ansatt) {      // (8)
    // Felt adskilt med et feltslutt-tegn.
    tekstSkriver.print(ansatt.hentFornavn() + FELT_SLUTT_TEGN);
    tekstSkriver.print(ansatt.hentEtternavn() + FELT_SLUTT_TEGN);
    tekstSkriver.print(ansatt.hentTimeLønn());
    tekstSkriver.print(FELT_SLUTT_TEGN);
    tekstSkriver.println(ansatt.hentKjønn());
}
```

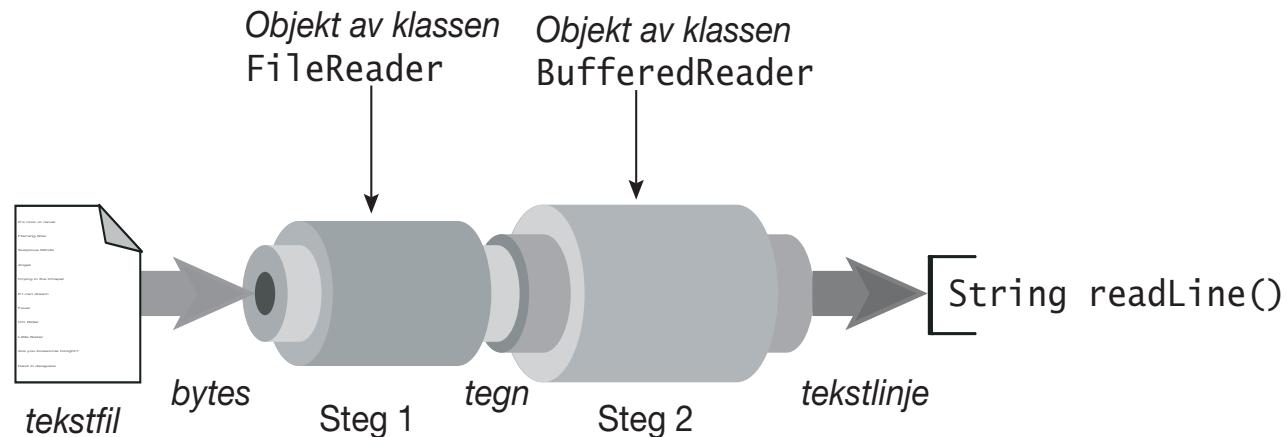
– Linje-slutttegn avslutter hver post på filen.

5. Avslutt ved å lukke strømmen:

```
tekstSkriver.close();
```

– Denne metoden sørger for at underliggende strømmer blir evt. spylt og frigjort.

Innlesing fra en tekstfil (Figur 14.5)



```
FileReader     tekstFilLeser = new FileReader(tekstfilnavn);      // Steg 1  
BufferedReader tekstLeser    = new BufferedReader(tekstFilLeser); // Steg 2
```

Innlesing fra tekstfiler (Figur 14.5, Figur 14.6 og Program 14.2):

1. Definer et `FileReader`-objekt som åpner filen det skal lese fra:

```
FileReader tekstFilLeser = new FileReader(datafilnavn); // (9)
```

2. Definer et `BufferedReader`-objekt som kobles til `FileReader`-objektet:

```
BufferedReader tekstLeser = new BufferedReader(tekstFilLeser); // (10)
```

3. Les én linje om gangen med `readLine()`-metoden som returnerer et `String`-objekt:

```
String post = tekstLeser.readLine(); // (15)
```

– Tegn i et felt kan hentes ut (steg 1 og 2 i Figur 14.6):

```
int feltsluttIndeks1 = post.indexOf(FELT_SLUTT_TEGN); // (16)
```

```
fornavn = post.substring(0, feltsluttIndeks1); // (17)
```

– Felttegn må konverteres til tilsvarende primitiv verdi (steg 3 i Figur 14.6):

```
String doubleStr = post.substring(feltsluttIndeks2 + 1, feltsluttIndeks3);
```

```
timelønn = Double.parseDouble(doubleStr); // (18)
```

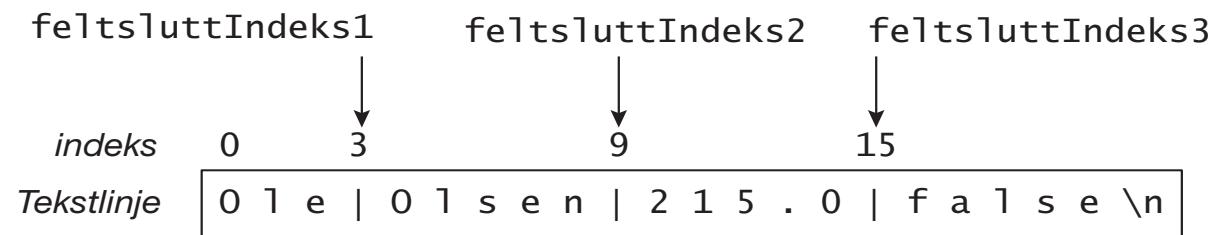
```
String sannhetsverdi = post.substring(feltsluttIndeks3 + 1);
```

```
kjønn = sannhetsverdi.equals("true"); // (19)
```

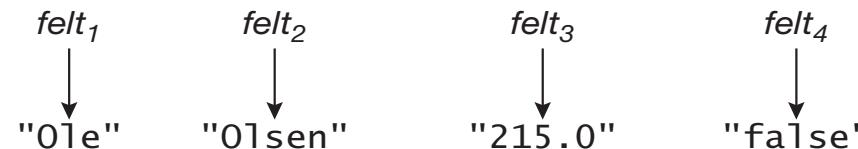
4. Avslutt ved å lukke strømmen.

Innlesing og konvertering av tekst til primitive verdier (Figur 14.6)

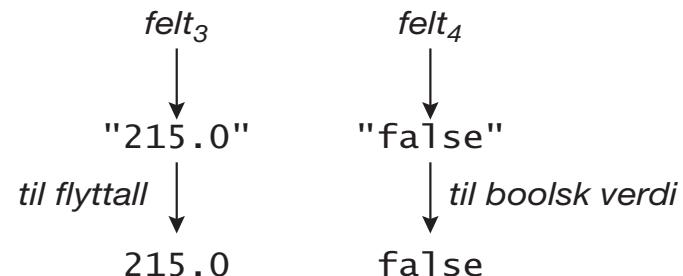
Steg 1: Finn indeks til feltslutt-tegn



Steg 2: Ekstraher substrang



Steg 3: Konverter substrang



Spalting av en tekstlinje til felt-verdier (alt. til steg 3)

- Se filen FirmaTxtARH.java.
- Vi kan bruke `split()`-metoden i `String`-klassen til dette formålet.

```
// Les en post (som en tekstlinje).
String post = tekstLeser.readLine(); // (15)

// Vi spalter tekstlinjen (dvs en post) v.h.a. split()-metoden som returnerer
// en tabell av strenger som tilsvarer felt i posten.
String[] felt = post.split("\\" + FELT_SLUTT_TEGN); // (16)

// Les de forskjellige feltene.
String fornavn = felt[0]; // (17)
String etternavn = felt[1];
double timelønn = Double.parseDouble(felt[2]); // (18)
boolean kjønn = felt[3].equals("true"); // (19)
```

Lukking av strømmer v.h.a. try-blokken

- Automatisk Ressurs Håndtering (ARH), eng. *Automatic Resource Management* (ARM).
- Vi kan bruke følgende syntaks til try-blokken som automatisk lukker *ressurser* (som er tilknyttet strømmer) ved å kalle `close()`-metoden, etter utføringen av try-blokken er ferdig, uansett om det kastes et unntak.

```
try (ressurs-opprettelse) {  
    // Ressurser brukes i try-blokken.  
}
```

- Ressurser må oppfylle kontrakten `java.io.Closeable` som definerer `close()`-metoden.
 - Alle strømmer implementerer denne kontrakten.
- Eksempel: Klassen `FirmaTxtARH`

```
void skrivAlleAnsatteTilTekstFil(String datafilnavn, boolean utvid) {  
    try ( FileWriter tekstFilSkriver = new FileWriter(datafilnavn,utvid);      // (3)  
         BufferedWriter bufretSkriver = new BufferedWriter(tekstFilSkriver); // (3a)  
         PrintWriter tekstSkriver = new PrintWriter(bufretSkriver) ){        // (4)  
        // skriv til fil.  
    } catch (IOException unntak) { System.out.println("Feil ved skriving: " + unntak); }  
}
```

Referansene `tekstFilSkriver`, `bufretSkriver` og `tekstSkriver` are *lokale* variabler i try-blokken.

Terminalvindu I/O

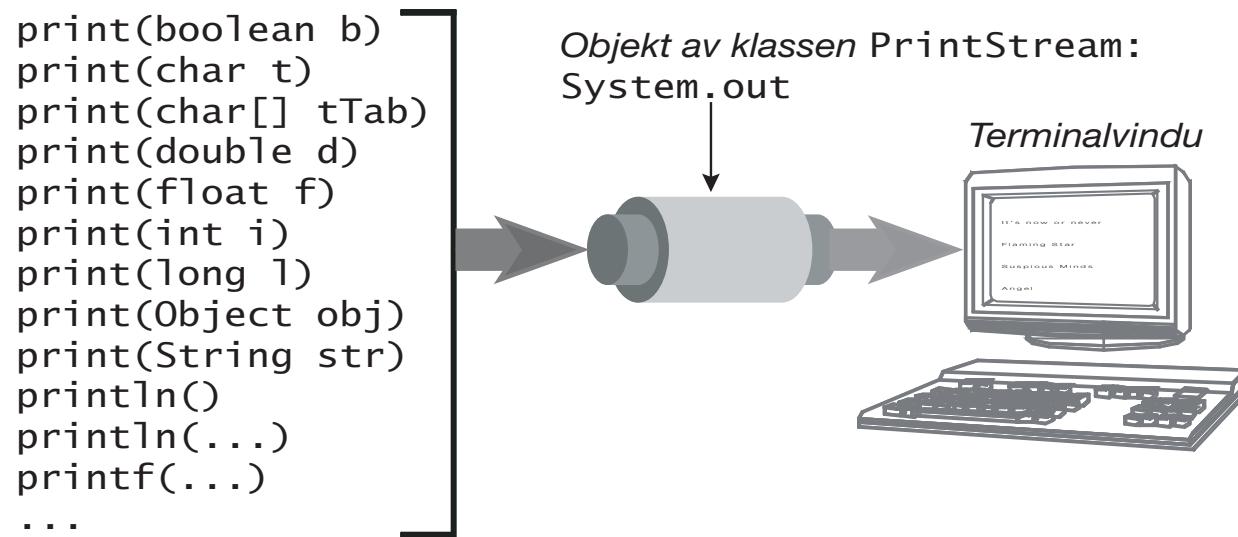
- Hvert Java-program under utføring får automatisk knyttet til seg følgende tre bytestrømmer:

1. `System.in` leser bytes fra tastaturet.
2. `System.out` skriver bytes til skjermen.
3. `System.err` for å skrive feilmeldinger til skjermen.

Utskrift til terminalvinduet

- `System.out` er et objekt av Klassen `PrintStream`, og brukes til å sende utskrift til terminalvinduet (Figur 14.7):

```
System.out.println("Livet er for kort. Spis desserten først!");
```



Avsøking med `java.util.Scanner`-klassen

- En `scanner` konverterer inndata fra dens kilde til *ord* (eng. *tokens*).
 - *Whitespace* (mellomrom, tabulatortegn, linjeskift) skiller ordene.
 - Kilden kan være en `String`, en `InputStream` eller en `Reader`, og sendes som parameter i konstruktørkallet.

Lookahead metoder

`boolean hasNext()`

Returnerer `true` dersom det er ord igjen inndata.

`boolean hasNextInt()`

`boolean hasNextLong()`

`boolean hasNextDouble()`

`boolean hasNextBoolean()`

Returnerer `true` dersom neste ord i inndata kan tolkes som typen som tilsvarer metodenavnet.

`void close()`

Lukker denne scanneren og evt. dens kilde.

Analyse av ord i inndata

`String next()`

Returnerer neste ord i inndata.

`String nextLine()`

Flytter innlesing forbi innværende linje, og returnerer inndata som ble hoppet over.

`int nextInt()`

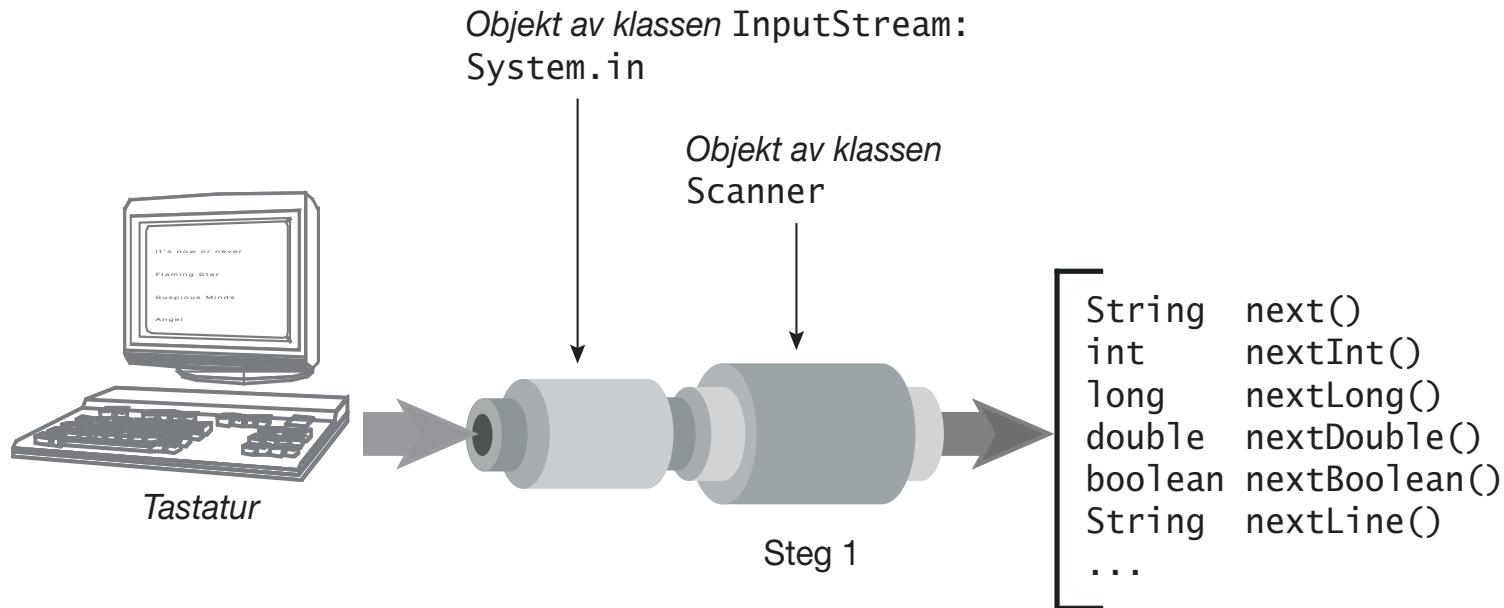
`long nextLong()`

`double nextDouble()`

`boolean nextBoolean()`

Tolker neste ord i inndata som typen som tilsvarer metodenavnet.

Innlesing fra terminalvinduet



```
Scanner tastatur = new Scanner(System.in); // Steg 1
```

- `System.in` er en byte-inn-strøm.
- Et `Scanner`-objekt kan kobles til `System.in` for innlesning.
- Metoder i `Scanner`-klassen brukes til å lese verdier som blir tastet.

Eksempel: Innlesing fra terminalvinduet

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class LesingFraTerminalMedScanner {
    public static void main(String[] args) {
        // En Scanner kobles til System.in, dvs. til terminalen.
        Scanner innleser = new Scanner(System.in);

        // Leser navn.
        String fornavn;
        String etternavn;
        String navn;
        String svar;
        do {
            System.out.print("Skriv fornavn: ");
            fornavn = innleser.next();
            innleser.nextLine(); // tøm inneværende linje for tegn.
```

```

        System.out.print("Skriv etternavn: ");
        etternavn = innleser.next();
        innleser.nextLine();
        navn = etternavn + " " + fornavn;
        System.out.println("Navn: " + navn);
        System.out.print("Ett til? (j/n): ");
        svar = innleser.next();
    } while (svar.equals("j"));

    // Leser en liste med heltall.
    List<Integer> intList = new ArrayList<Integer>();
    System.out.println("Skriv en liste med heltall," +
                       " avslutt med en verdi som ikke er heltall:");
    while(innleser.hasNextInt()) {
        int i = innleser.nextInt();
        intList.add(i);
    }
    for (int i : intList)
        out.println(i);
    // Lukk scanner.
    innleser.close();
}
}

```

Kjøring av programmet:

Skriv fornavn: **Junior**

Skriv etternavn: **Java**

Navn: Java Junior

Ett til? (j/n): **n**

Skriv en liste med heltall, avslutt med en verdi som ikke er heltall:

132 454

43

-1

slutt

132

454

43

-1

Klassen Tastatur (Program 14.3)

- Klassen Tastatur definerer statiske metoder for innlesning av verdier.

```
public final class Tastatur {  
    private Tastatur() {};  
    private static Scanner leser = new Scanner(System.in);  
  
    public static int lesInt() {  
        while (true)  
            try { return leser.nextInt(); }  
            catch (InputMismatchException ime) { reportError(); }  
    }  
    // ...  
    private static void reportError() {  
        leser.nextLine(); // Tøm linjen først.  
        System.out.println("Feil i inndata. Prøv på nytt!");  
    }  
}
```

- Metoden `nextInt()` kan kaste et ikke-kontrollert unntak som kan være en god idé å håndtere.

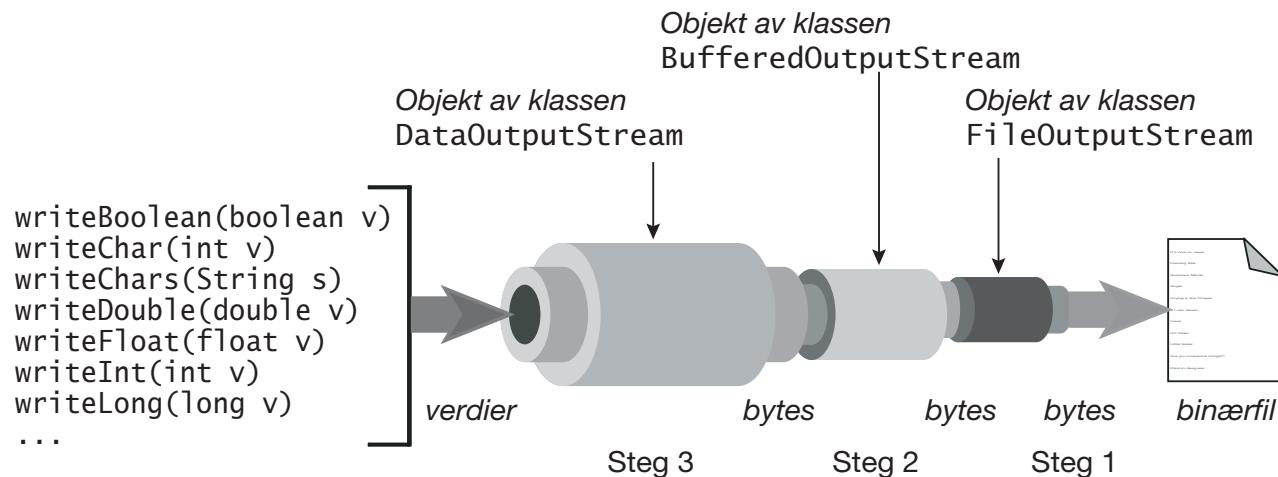
–

- Se Program 14.4 (klassen FirmaTerminalVindu)

Binære filer

- En binærfil lagrer *binære representasjoner* av primitive verdier definert i Java.
- Tallverdier på filen opptar lagerplass i henhold til lengden definert i Java-programmeringsspråket.
- Tegn blir representert i Unicode-tegnsettet på i en binærfil.
- De boolske verdiene `true` og `false` blir lagret på filen med binær representasjon gitt ved henholdsvis `(byte)1` og `(byte)0`.
- Programmet må sørge for at verdiene som skrives ut blir tolket riktig ved innlesing.

Utskriving av binære verdier



```
FileOutputStream utFilStrøm = new FileOutputStream(datafilnavn, utvid); // Steg 1  
BufferedOutputStream bufretUtStrøm = new BufferedOutputStream(utFilStrøm); // Steg 2  
DataOutputStream utBinStrøm = new DataOutputStream(bufretUtStrøm); // Steg 3
```

Saksgangen for å skrive binære verdier på en fil

- Se klassen FirmaBin -- buker try-blokk med ARH.

1. Definer et `FileOutputStream`-objekt som åpner filen det skal skrives til:

```
FileOutputStream utFilStrøm = new FileOutputStream(datafilnavn, utvid);
```

2. Definer et `BufferedOutputStream`-objekt som kobles til `FileOutputStream`-objektet:

```
BufferedOutputStream bufretUtStrøm = new BufferedOutputStream(utFilStrøm);
```

3. Definer et `DataOutputStream`-objekt som kobles til `BufferedOutputStream`-objektet:

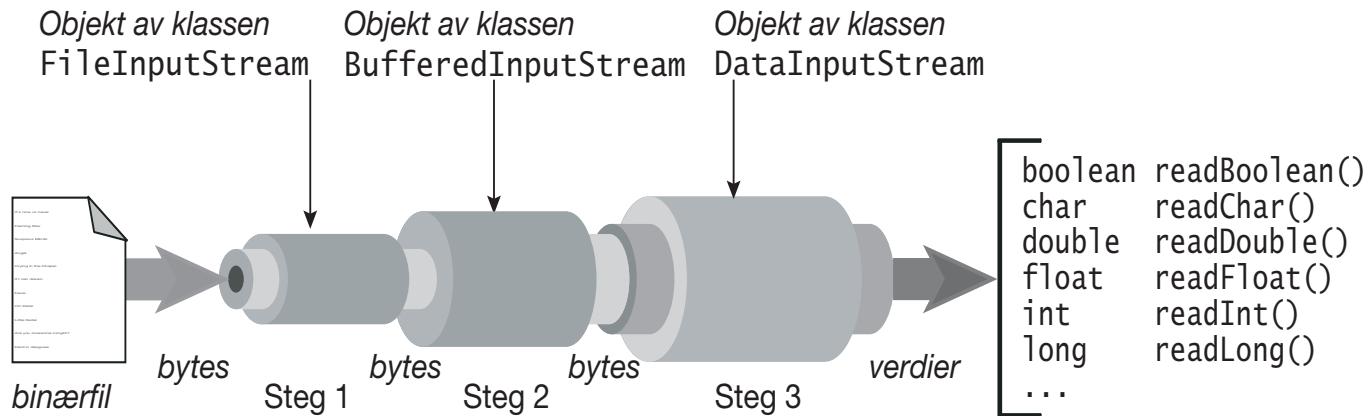
```
DataOutputStream utBinStrøm = new DataOutputStream(bufretUtStrøm);
```

4. Skriv verdier binært med relevante `writeX()`-metoder definert i `DataOutputStream`-klassen.

```
// Et strengfelt avslutes med FELT_SLUTT_TEGN.  
utBinStrøm.writeChars(ansatt.hentFornavn() + FELT_SLUTT_TEGN);  
utBinStrøm.writeChars(ansatt.hentEtternavn() + FELT_SLUTT_TEGN);  
utBinStrøm.writeDouble(ansatt.hentTimelønn());  
utBinStrøm.writeBoolean(ansatt.hentKjønn());
```

5. Strømmene lukkes automatisk i try-blokk med ARH.

Innlesing av binære verdier



```
FileInputStream innFilStrøm = new FileInputStream(datafilnavn); // Steg 1  
BufferedInputStream bufretInnStrøm = new BufferedInputStream(innFilStrøm); // Steg 2  
DataInputStream innBinStrøm = new DataInputStream(bufretInnStrøm); // Steg 3
```

Innlesing av binære verdier

- Prosedyren for å lese binære verdier fra en fil er følgende (klassen FirmaBin):
 - Bruker en try-blokk med ARH.

1. Definer et `FileInputStream`-objekt som åpner filen det skal leses fra:

```
FileInputStream innFilStrøm = new FileInputStream(datafilnavn);
```

2. Definer et `BufferedInputStream`-objekt som kobles til `FileInputStream`-objektet:

```
BufferedInputStream bufretInnStrøm = new BufferedInputStream(innFilStrøm);
```

3. Definer et `DataInputStream`-objekt som kobles til `BufferedInputStream`-objektet:

```
DataInputStream innBinStrøm = new DataInputStream(bufretInnStrøm);
```

4. Les verdier binært med relevante `readX()`-metoder definert i `DataInputStream`-klassen.

```
timelønn = innBinStrøm.readDouble();  
kjønn     = innBinStrøm.readBoolean();
```

5. Strømmene lukkes i try-blokk med ARH.

Håndtering av slutt på inndata for binær innlesing

- Se klassen `FilSluttDemo`.
- Slutt på inndata håndteres ved å fange kontrollerte unntak av typen `EOFException`.
- Metoden `main()` sørger for å opprette strømmene og behandle `IOExceptions` som metoden `lesFraBinFil()` kaster videre.

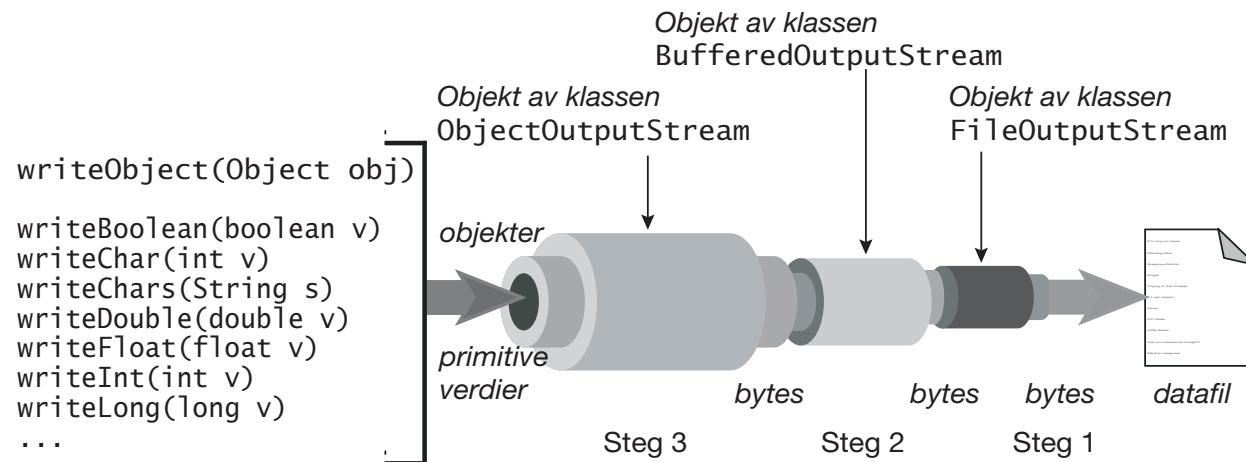
Objektserialisering

- Prosessen for å skrive ut et objekt kalles *serialisering*.
 - All relevant informasjon om et objekt skrives ut som en bytesekvens.
- Prosessen for å lese inn disse opplysningene for å gjenskape et objekt kalles *deserialisering*.
 - Objektet kan "gjenskapes" ved å tolke dets bytesekvens.
- Klassene `ObjectOutputStream` og `ObjectInputStream` definerer henholdsvis *objekt-ut-strømmer* og *objekt-inn-strømmer* som tilbyr metoder for å serialisere og deserialisere objekter.
- Klassene kan også brukes til å skrive og lese binære verdier og objekter.
- For at et objekt til en klasse skal kunne serialiseres, må klassen implementere `java.lang.Serializable`-kontrakten.

```
interface Serializable {} // Tom kropp.
```

```
class Ansatt implements Serializable {...}
```

Utskriving av objekter og binære verdier



```
FileOutputStream utFilStrøm = new FileOutputStream(datafilnavn, utvid); // Steg 1  
BufferedOutputStream bufretUtStrøm = new BufferedOutputStream(utFilStrøm); // Steg 2  
ObjectOutputStream utObjStrøm = new ObjectOutputStream(bufretUtStrøm); // Steg 3
```

Utskriving av objekter

- Metoden `writeObject()` i klassen `ObjectOutputStream` brukes til å skrive objekter.
`utObjStrøm.writeObject(nyAnsatt);`
- Metoden skriver ut tilstanden til et objekt, som utgjør verdiene til *feltvariabler* i objektet.
 - Statiske datamedlemmer blir ikke tatt med.
- Følgende metodekall serialisere hele ansatt-tabellen:
`utObjStrøm.writeObject(ansattTabell);`
 - Legg merke til at kallet ovenfor fører til at hele ansatttabellen blir serialisert, inklusive Ansatt-objekter i tabellen.
- Ved innlesingen blir samme tilstand gjenskapt som før utskrivingen.

Saksgangen for å serialisere objekter til en fil

- Se klassen FirmaObj, Ansatt, AkkordArbeider, ArbeidsLeder, FastAnsatt, TimebetaltArbeider.
 - Bruker en try-blokk med ARH.

1. Definer et FileOutputStream-objekt som åpner filen det skal skrives til:

```
FileOutputStream utFilStrøm = new FileOutputStream(datafilnavn, utvid);
```

2. Definer et BufferedOutputStream-objekt som kobles til FileOutputStream-objektet:

```
BufferedOutputStream bufretUtStrøm = new BufferedOutputStream(utFilStrøm);
```

3. Definer et ObjectOutputStream-objekt som kobles til FileOutputStream-objektet:

```
ObjectOutputStream utObjStrøm = new ObjectOutputStream(bufretUtStrøm);
```

4. Skriv ut objekter med writeObject()-metoden:

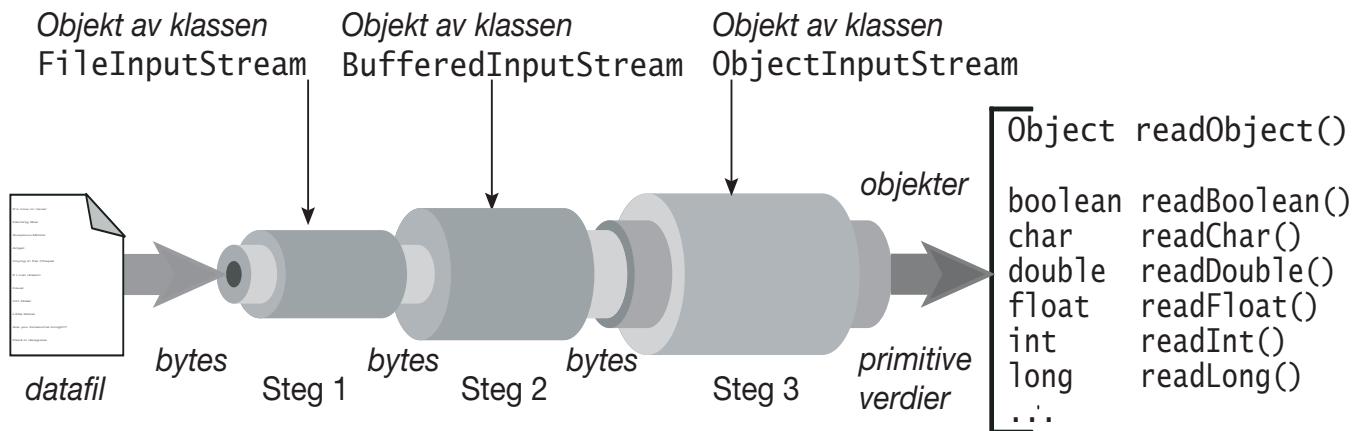
```
utObjStrøm.writeObject(ansattTabell); // Skriver ut hele tabellen.
```

I tillegg til å skrive objekter kan vi også skrive primitive verdier, for eksempel:

```
utObjStrøm.writeInt(antallAnsatte);
utObjStrøm.writeInt(antallKvinner);
```

5. Strømmene lukkes automatisk i try-blokk med ARH.

Innlesing av objekter og binære verdier



```
FileInputStream innFilStrøm = new FileInputStream(datafilnavn);           // Steg 1  
BufferedInputStream bufretInnStrøm = new BufferedInputStream(innFilStrøm); // Steg 2  
ObjectInputStream innObjStrøm = new ObjectInputStream(bufretInnStrøm);    // Steg 3
```

Innlesing av objekter

- Under deserialiseringen må man holde rede på at riktig antall objekter blir lest i riktig rekkefølge.
- Metoden `readObject()` i `ObjectInputStream`-klassen brukes til å lese inn ett serialisert objekt om gangen, *inklusive dets delobjekter*:

```
Object objRef = innObjStrøm.readObject();
```

- Programmet må selv holde rede på hvilken klasse objektet egentlig tilhører:

```
if (objRef instanceof Ansatt) {                                // (1)
    Ansatt enAnsatt = (Ansatt) objRef;                      // (2)
    double lønn = enAnsatt.beregnUkelønn(47.5);           // (3)
}
```

- Metoden `readObject()` gjenskaper hele objektet:

```
Object ref = innObjStrøm.readObject();
if (ref instanceof PersonellRegister) {
    PersonellRegister nyttPersonellRegister = (PersonellRegister) ref;
    nyttPersonellRegister.skrivAnsattInfoTilTerminal();
}
```

Saksgangen for å deserialisere objekter fra en fil

- Bruker en try-blokk med ARH til å deklarere strømmene.

1. Definer et `FileInputStream`-objekt som åpner filen det skal leses fra:

```
FileInputStream innFilStrøm = new FileInputStream(datafilnavn);
```

2. Definer et `BufferedInputStream`-objekt som kobles til `FileOutputStream`-objektet:

```
BufferedInputStream bufretInnStrøm = new BufferedInputStream(innFilStrøm);
```

3. Definer et `ObjectInputStream`-objekt som kobles til `FileInputStream`-objektet:

```
ObjectInputStream innObjStrøm = new ObjectInputStream(bufretInnStrøm);
```

4. Les et objekt med `readObject()`-metoden.

– Kontroller objektets type før vi kaller metoder definert spesifikt i objektets klasse.

```
// Les et objekt fra filen.  
Object nyttObj = innObjStrøm.readObject();  
// Kontroller at det er en ansatttabell.  
if (nyttObj instanceof Ansatt[])  
    ansattTabell = (Ansatt[]) nyttObj;  
else  
    System.out.println("Ansatt-tabell ikke funnet.");
```

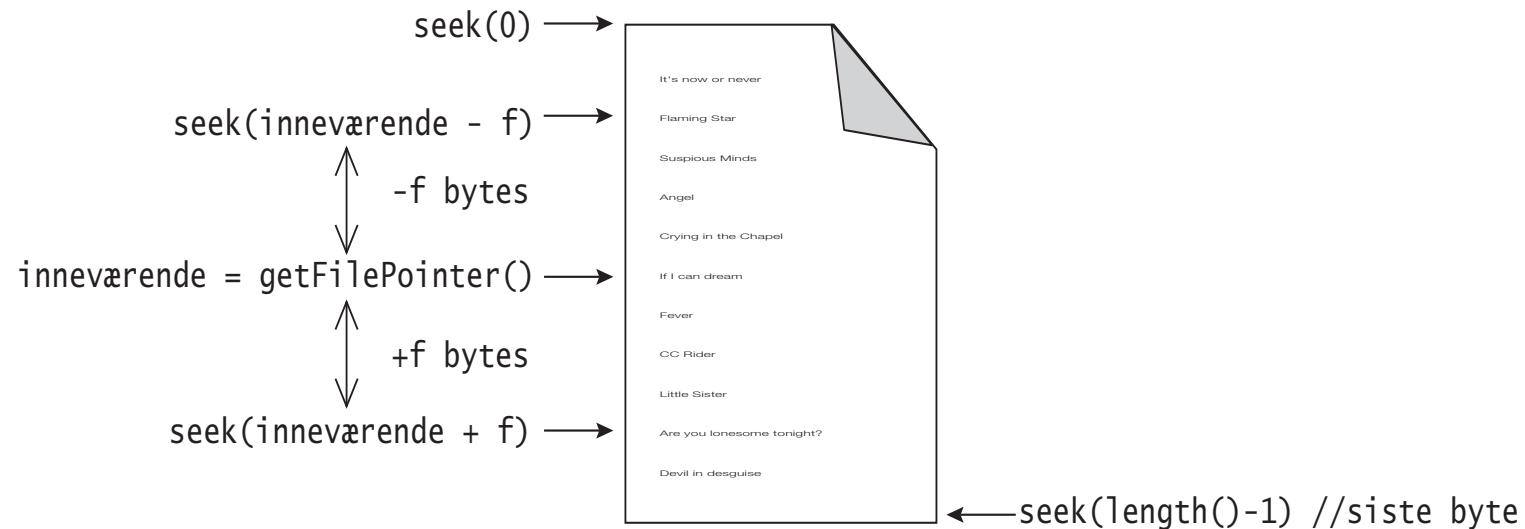
5. Strømmene lukkes automatisk i try-blokk med ARH.

Mer om serialisering

- Serialiseringsprosessen er sekvensiell.
 - Dette medfører at objektene blir deserialert i den rekkefølgen de ble serialisert.
- Serialiseringsmekanismen unngår duplisering av objekter
(se klassen `SerialiseringUtenDuplisering`).

Direkte filtilgang (Figur 14.13)

- Klassen RandomAccessFile tilbyr det som kalles *direkte filtilgang* (Tabell 14.2).
 - Kan både lese og skrive fra en vilkårlig *posisjon* i en fil.



Direkte filtilgang (Tabell 14.2)

java.io.RandomAccessFile	
RandomAccessFile(String filnavn, String filmodus) throws FileNotFoundException	Åpner fil med angitt filnavn.
long length()	Henter antall bytes lagret i filen, det vil si <i>lengden</i> til filen.
void setLength(long nyLengde)	Lengden til filen
long getFilePointer()	Henter filpekerens inneværende posisjon.
void seek(long forskyvning)	Flytter filpekeren til posisjon angitt ved verdien i parameteren <i>forskyvning</i> .
int skipBytes(int n)	Hopper <i>n</i> bytes fremover fra inneværende posisjon i filen.
boolean readBoolean() char readChar() double readDouble() float readFloat() int readInt() long readLong()	Klassen definerer metoder for innlesing av binære verdier.

java.io.RandomAccessFile

void writeBoolean(boolean v) void writeChar(int v) void writeDouble(double v) void writeFloat(float v) void writeInt(int v) void writeLong(long v)	Klassen definerer metoder for skriving av binære verdier.
void writeChars(String s)	Metoden skriver hvert tegn i strengen som to bytes til strømmen, det vil si som et Unicode-tegn.
void close()	Metoden lukker filen.

Bruk av klassen RandomAccessFile

- For å opprette direkte tilgang til en fil må filen åpnes ved å angi filnavnet under opprettelsen av et RandomAccessFile-objekt:

```
try {  
    RandomAccessFile raf = new RandomAccessFile(datafilnavn, "rw");  
} catch (FileNotFoundException unntak) { ... }
```

- Filmodusen "rw" tillater kombinering av både lese- og skriveoperasjoner på filen.
- Filmodusen "r" tillater bare lesing fra en eksisterende fil.
- Antall bytes i en fil kalles *fullengde*, som returneres av metoden `length()`.
- En *filpeker* (eng. *file pointer*) angir hvor i filen lesing eller skriving kan utføres.
 - En *markør* mellom to etterfølgende bytes på filen angir hvor lesing og skriving kan skje.
 - Posisjoneringen av filpekeren med metoden `seek()` gjøres alltid ved å spesifisere riktig antall bytes fra *starten* av filen.

- Å hoppe over fem tegn fra inneværende posisjon:
`raf.skipBytes(10);`
- For å utvide en fil ved å supplere data på slutten, må filpekerverdien settes lik fil-lengden:
`raf.seek(raf.length()); // Sett filpekeren til slutten av filen.`
- Klassen `RandomAccessFile` kan også brukes til å lese og skrive *binære* verdier.
- En `RandomAccessFile`-strøm må også lukkes etter bruk.

Eksempel: Bruk av direkte filtilgang (Program 14.9)

- Klassen `FirmaDirekte` leser og skriver en ansatt-post i posisjon angitt ved filpekeren.
- Alle poster har samme faste lengde.
 - Dette betyr at samme felt i alle poster må ha fast feltlengde.
 - For verdier av predefinerte datatyper er det gitt hvor mange bytes de opptar i filen.
 - For at et felt med en strengverdi skal ha fast lengde, kan vi bruke følgende løsning:
 - Dersom strengen er mindre enn feltlengden, fylles feltet med et bestemt tegn.
 - Dersom strengen er større enn feltlengden, blir strengen trunkert slik at den passer i feltet.

```
final static char FYLLE_TEGN          = '\u0000';
final static int FORNAVN LENGDE       = 20; // Unicode-tegn
final static int ETTERNAVN LENGDE     = 30; // Unicode-tegn
final static int TIMELØNN LENGDE      = 8;  // bytes
final static int KJØNN LENGDE         = 1;  // byte
final static int POST LENGDE BYTES    = 2*FORNAVN LENGDE +
                                         2*ETTERNAVN LENGDE +
                                         TIMELØNN LENGDE +
                                         KJØNN LENGDE; // bytes
```

- Metoden `skrivFastLengdeStreng()` i linje (9) tar strenglengden i betraktnign og sørger alltid for at hele feltet blir fylt:

```
int antallFylleTegn = MAKS_FELET LENGDE - str.length();
for (int k = 0; k < antallFylleTegn; k++)
    str += FYLLE_TEGN;
raf.writeChars(str.substring(0, MAKS_FELET LENGDE));
```

- Metoden `lesFastLengdeStreng()` i linje (12) sørger for at et felt som inneholder en streng, blir lest riktig:

```
String str = "";
int i;
for (i = 0; i < MAKS_FELET LENGDE; i++) {
    char tegn = raf.readChar();
    if (tegn == FYLLE_TEGN) break;
    else str += tegn;
}
// Hopp over ev. sekvens av FYLLE_TEGN som opptar 2 bytes hver.
raf.skipBytes(2*(MAKS_FELET LENGDE - i - 1));
return str;
```

- Metoden `initDirekteTilgangFil()` oppretter et `RandomAccessFile`-objekt i en `try`-blokk i linje (7).
 - Den *nullstiller* først innholdet i filen ved å sette fillengden til null.

- Metoden `utvidMedEnAnsatt()` definert i linje (8) utvider filen med en ny post:

```
// Posisjoner filpeker etter siste post.  
raf.seek(raf.length());  
// Skriv ansatt-info til filen.  
ansatt.skrivAnsattData(ansatt);
```

og oppdaterer opplysninger om antall poster lagret ved starten av filen:

```
raf.seek(0);  
int antallRegistrert = raf.readInt();  
raf.seek(0);  
raf.writeInt(antallRegistrert + 1);
```

- Metoden `registrerNyTimeLønnForAnsattPåFil()` i linje (10) oppdaterer direkte inn i filen timelønnen til en ansatt som er angitt ved et *ansatt-nummer*:

```
raf.seek(4 + ansattNummer*AnsattDirekte.POST LENGDE BYTES);
```

Etter at posten er lest til et `Ansatt`-objekt, blir timelønnen justert.

Deretter blir filpekeren spolt tilbake slik at tidligere feltverdier i posten kan overskrides:

```
long inneværendePosisjon = raf.getFilePointer();  
raf.seek(inneværendePosisjon - POST LENGDE BYTES);  
skrivAnsattData(enAnsatt);
```