Kapittel 14:
Filer og strømmer

Kort versjon

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

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.

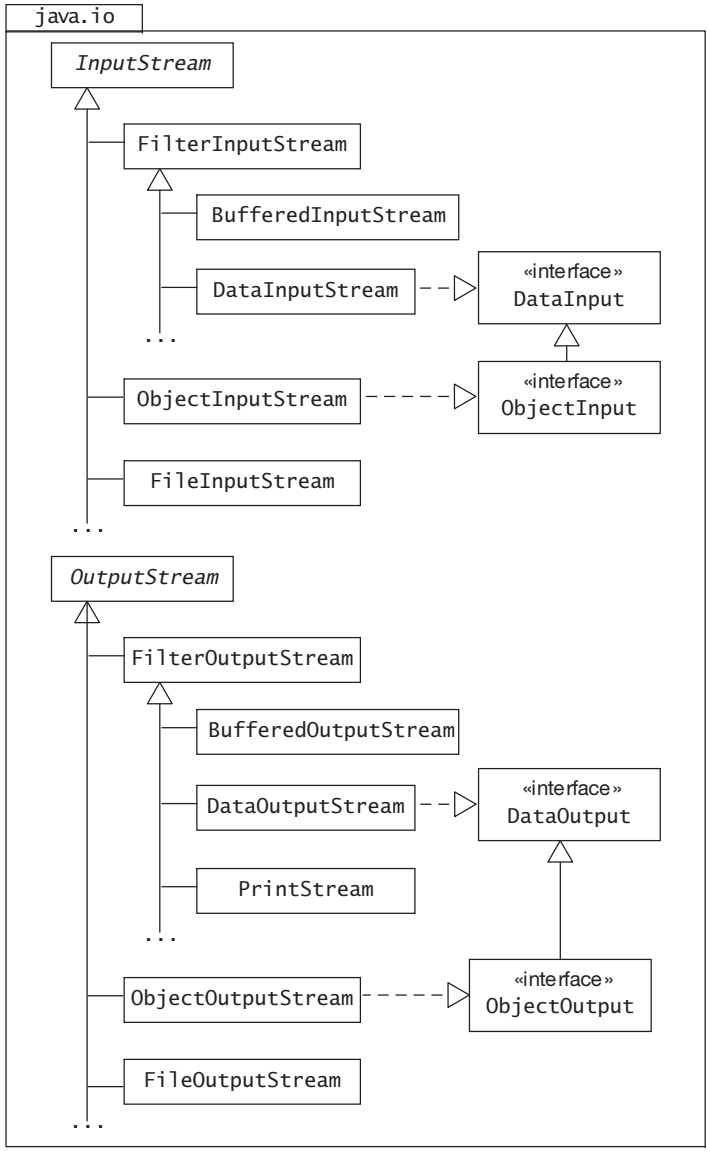
Strømmer

- 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*.

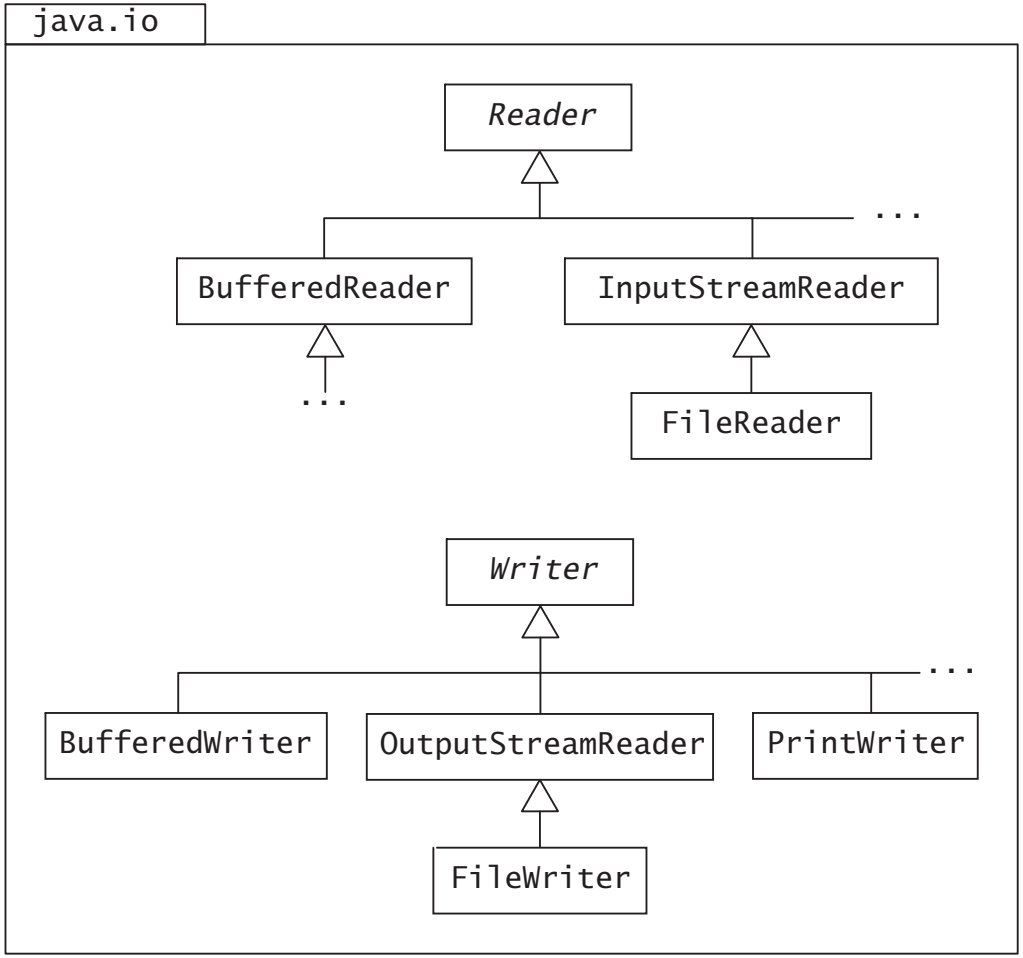
Bytestrømmer og tegnstrømmer

- *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* identifiserer en fil i filsystemet:

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

String filsti1 = "firma\\ansattfil.dat"; // Windows
String filsti2 = "firma/ansattfil.dat";  // Unix
```

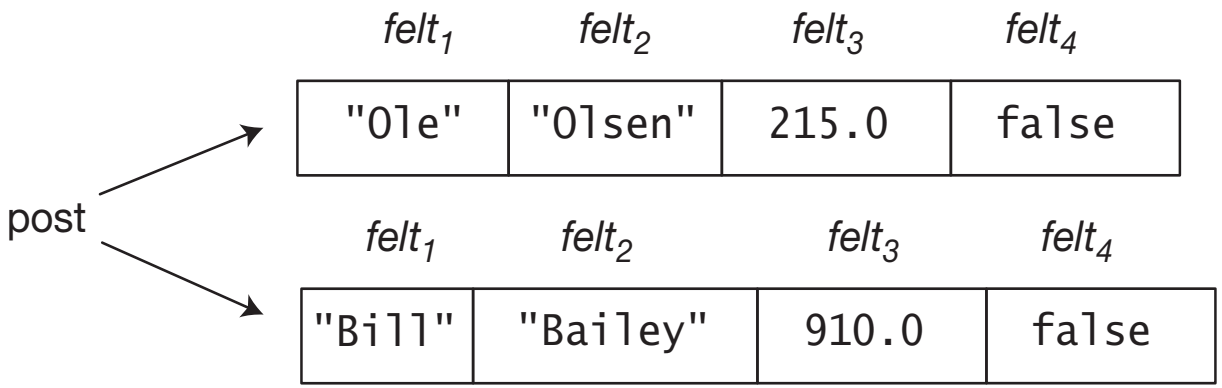
```
String filnavn = "firma" + File.separator + "ansattfil.dat"; // Plattformuavhengig
```

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 felt-verdi.

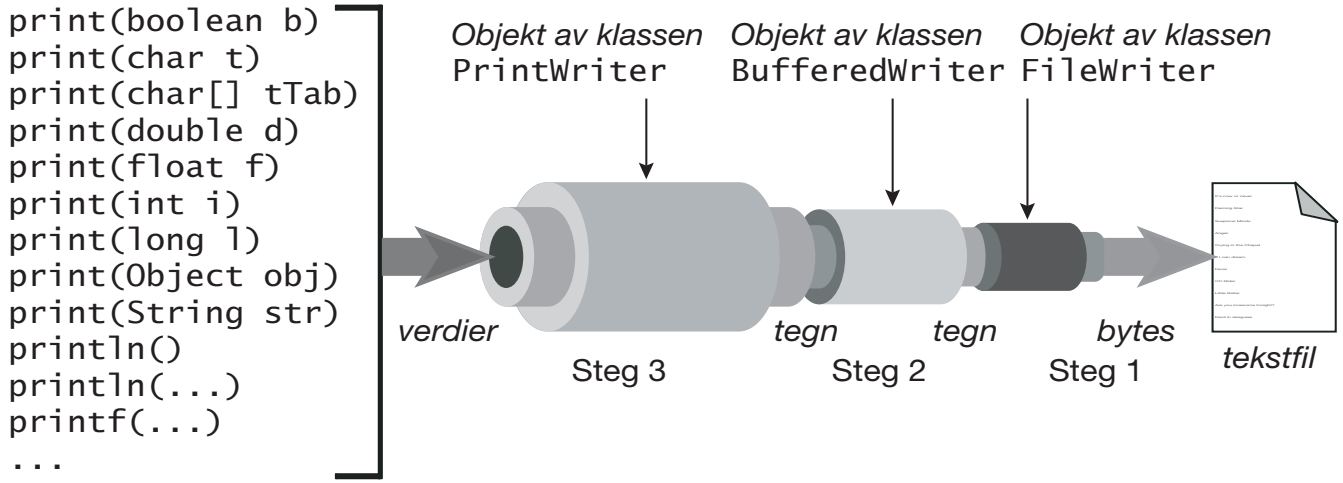


- Se Program 14.1: Klassene `Ansatt`, `PersonellRegister` og `Firma`.

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




```
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 tekstFilSkriver = 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(tekstFilSkriver);     // (3a)
   ```

   – Tegn blir bufret internt for effektiv skriving til filen.

3. Definer et `PrintWriter`-objekt som kobles til `BufferedWriter`-objektet:

   ```
   PrintWriter tekstSkriver = 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());
}
```

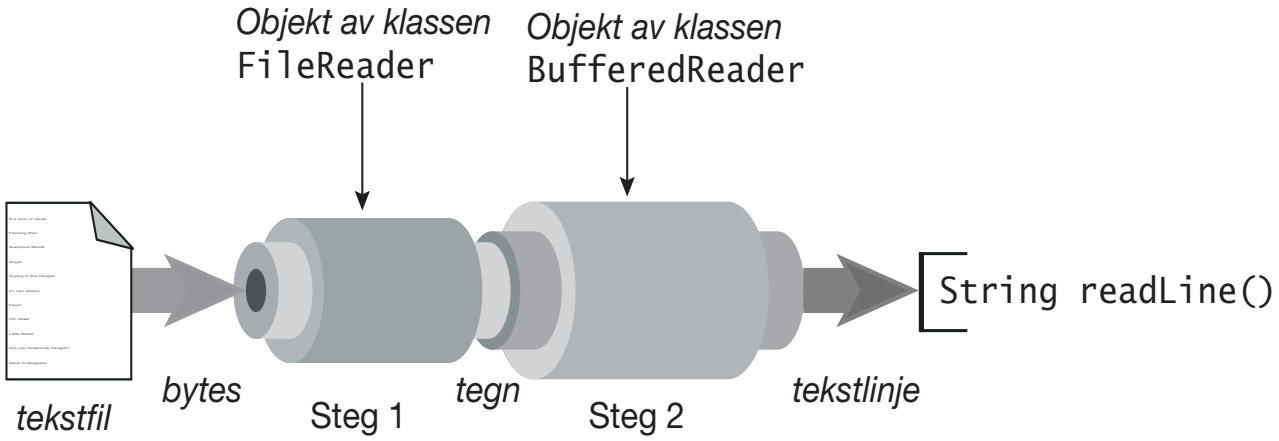
   – Linje-slutttegn avslutter hver post på filen.

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):

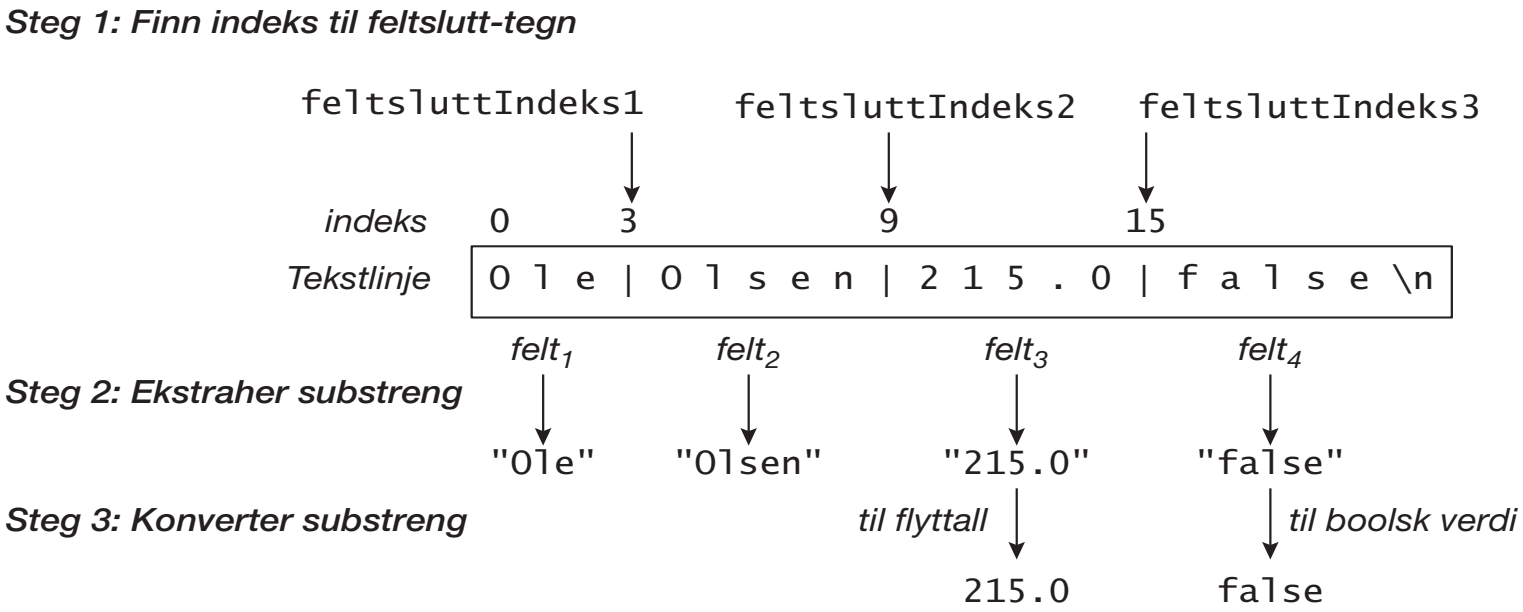
```
int feltsluttIndeks1 = post.indexOf(FELT_SLUTT_TEGN);                // (16)
fornavn = post.substring(0, feltsluttIndeks1);                       // (17)
```

   – 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)



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

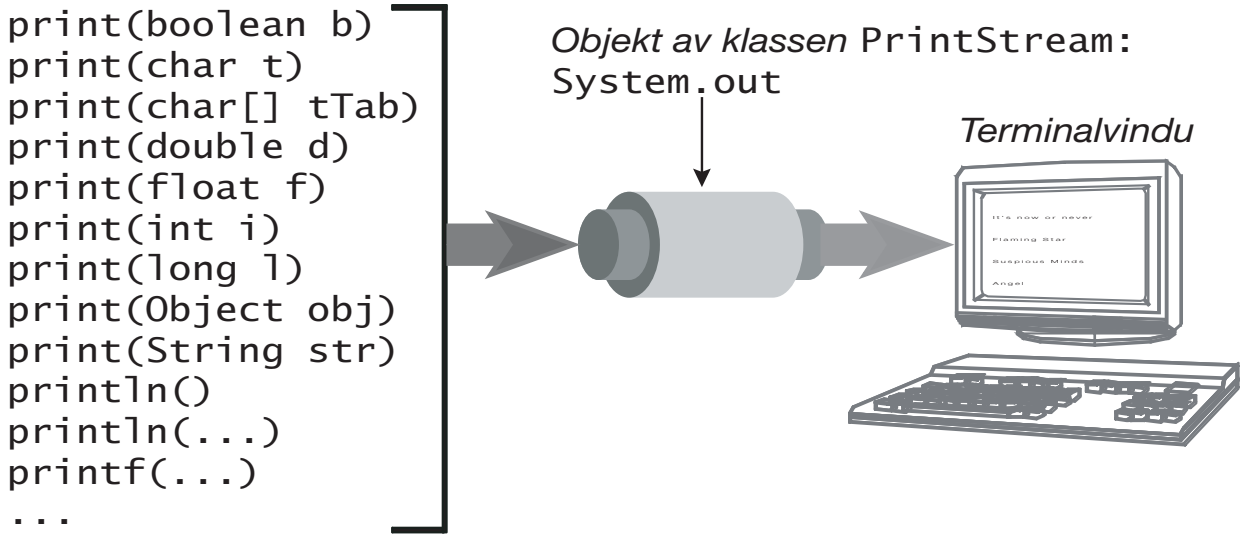
- Hvert Java-program under utføring får automatisk knyttet til seg følgende tre byte-strømmer:

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.

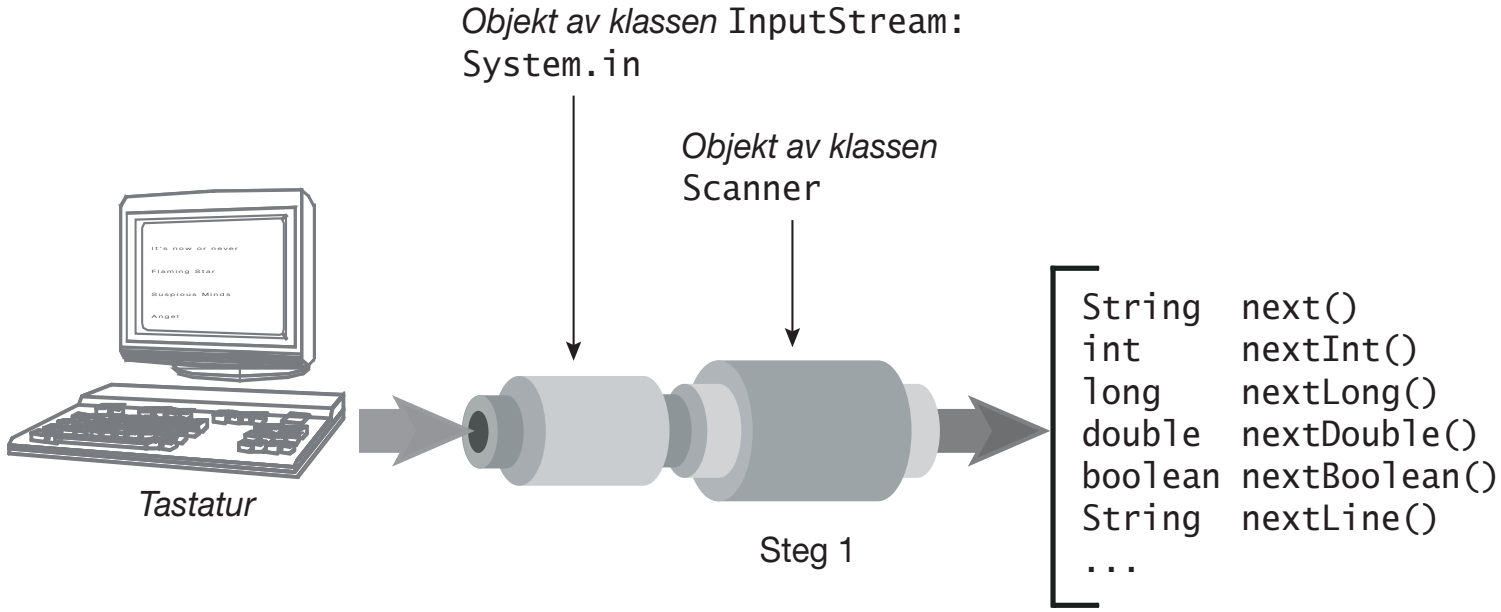
Lookahead metoder	Analyse av ord i inndata

`boolean hasNext()`
Returnerer `true` dersom det er ord igjen inndata.
	`String next()`
	Returnerer neste ord i inndata.
	`String nextLine()`
	Flytter innlesing forbi innværende linje, og returnerer inndata som ble hoppet over.

`boolean hasNextInt()`
`boolean hasNextLong()`
`boolean hasNextDouble()`
`boolean hasNextBoolean()`
Returner `true` dersom neste ord i inndata kan tolkes som typen som tilsvarer metodenavnet.
	`int nextInt()`
	`long nextLong()`
	`double nextDouble()`
	`boolean nextBoolean()`
	Tolker neste ord i inndata som typen som tilsvarer metodenavnet.

`void close()` Lukker denne scanneren og evt. dens kilde.	

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 list 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`)