INF1000-SIKT - Notat om I/O i Java

Tuva Kristine Thoresen
tuvakt@ulrik.uio.no

30. oktober 2013

Innhold

I dette notatet skal vi se litt på I/O i Java. Dette notatet er ment for studenter på inf1000-sikt, men kan også brukes av andre studenter.

1 Innlesning fra terminal

Det første vi skal se på er innlesning fra terminal. Dette bruker vi ofte i interaksjon med brukere av programmet. Programmet ber om input, som brukeren så taster inn. Programmet må så lese inn dette og bearbeide informasjonen.

For å få til innlesning fra terminal bruker vi en ferdiglagd Java-klasse som heter `Scanner`. Denne klassen inneholder mange nyttige funksjoner for innlesning.

For å bruke denne klassen må vi først importere de nødvendige programmene. Disse ligger i `java.util` og kan importeres på følgende måte:

```
import java.util.Scanner;
```

Neste steg er å opprette et innlesningsobjekt. Det må vi gjøre slik at vi får tak i hele `Scanner`-klassen og dens funksjonalitet. Dette gjøres slik:

```
Scanner in = new Scanner(System.in);
```

Når vi leser fra terminalen leser vi fra standard input. Dette får vi tak i gjennom `System.in`.

1.1 Innlesning av tekst

`Scanner` har to metoder som vi bruker for å lese inn tekst:

- `nextLine()` - leser inn en linje av tekst.
- `next()` - leser inn et ord.

Disse metodene kan brukes på følgende måte:

```
System.out.print("Tast inn en setning: ");
String linje = in.nextLine();

System.out.print("Tast inn et ord: ");
String ord = in.next();
```

1.2 Innlesning av tall

For å lese inn tall med `Scanner` kan vi bruke disse to metodene:

- `nextInt()` - leser inn et heltall.
- `nextDouble()` - leser inn et desimaltall.

Eksempel på bruk av disse to metodene:

```
System.out.print("Tast inn et heltall: ");
int heltall = in.nextInt();

System.out.print("Tast inn et desimaltall: ");
double desimaltall = in.nextDouble();
```

1.3 Eksempelprogram

Nå skal vi lage et lite program som leser inn et heltall, et desimaltall og en setning fra brukeren.

```
import java.util.Scanner;

class InputEks {
    public static void main(String[] args) {

        // oppretter innlesningsobjekt:
        Scanner in = new Scanner(System.in);

        // leser inn et heltall:
        System.out.print("Tast inn din alder: ");
        int alder = in.nextInt();

        // leser inn et desimaltall:
        System.out.print("Tast inn din hoyde (i meter): ");
        double hoyde = in.nextDouble();

        // leser inn linjeskiftet:
        in.nextLine();

        // leser inn en setning:
        System.out.print("Tast inn ditt fulle navn: ");
        String navn = in.nextLine();
    }
}
```

Vi er nødt til å lese inn en ekstra linje, før vi leser inn navnet. Dette er fordi metodene `nextInt()` og `nextDouble()` bare leser inn tallene, og ikke linjeskiftet. Dermed får vi problemer neste gang vi skal lese inn en setning.

1.4 Alternativ innlesning av tall

For å unngå det ekstra kallet på `nextLine()` etter man har lest inn et tall, kan man unngå bruk av metodene `nextInt()` og `nextDouble()`. Isteden kan man lese inn en linje, og så konvertere denne teksten til tall etterpå. Dette kan skje på følgende måte:

```
String linje = in.nextLine();
int heltall = Integer.parseInt(linje);
```

Her konverterer vi teksten `linje` til et heltall ved kall på metoden `Integer.parseInt()`. Denne metoden tar en tekst som inneholder et tall (for eksempel "4") som input og konverterer denne til et heltall (her: med verdi 4). På tilsvarende måte kan man konvertere en tekst til et desmialtall, med metoden `Double.parseDouble()`.

Eksempelprogrammet vårt kan da se slik ut:

```
import java.util.Scanner;

class InputAlt {
    public static void main(String[] args) {

        // oppretter innlesningsobjekt:
        Scanner in = new Scanner(System.in);

        // leser inn et heltall:
        System.out.print("Tast inn et heltall: ");
        String linje = in.nextLine();
        int heltall = Integer.parseInt(linje);

        // leser inn et desimaltall:
        System.out.print("Tast inn et desimaltall: ");
        String tekst = in.nextLine();
        double desimaltall = Double.parseDouble(tekst);

        // leser inn en setning
        System.out.print("Tast inn ditt fulle navn: ");
        String navn = in.nextLine();
    }
}
```

Her ser vi at vi slipper det ekstra kallet på `nextLine()`.

2 Innlesning fra fil

Når vi leser inn fra terminalen med `Scanner` leser vi fra `System.in`. For å lese inn fra fil, kan vi også bruke `Scanner`. Forskjellen er at nå må vi sende med filen som argument til `Scanner`.

Det er dessverre noen små problemer med å lese fra fil. Vi har nemlig ingen garanti for at filen vi prøver å lese fra, faktisk finnes! Java lar oss løse dette ved at vi først prøver å lese fra filen. Hvis dette går galt, får vi en feilmelding som vi må håndtere.

Vi må først huske å importere de nødvendige programmene:

```
import java.util.Scanner;
import java.io.*;
```

og så kan vi begynne å lese. Vi oppretter først et innlesningsobjekt, og forteller det at det skal lese fra filen *innfil.txt*:

```
try {
  File f = new File("innfil.txt");
  Scanner innFil = new Scanner(f);

} catch(IOException e) {
  e.printStackTrace();
}
```

Legg merke til `try-catch`-blokkene. Disse forteller Java at vi prøver å lese fra fil, og at vi fanger opp feilen hvis noe går galt.

Hvis noe går galt kaster programmet vårt en `IOException`. Det er rett og slett en feilmelding som sier at noe gikk galt med innlesningen fra fil. Denne feilen fanger vi opp i `catch`-blokka, og gjør noe med den. Her har vi lyst til å skrive ut feilmeldingen vi får til terminalen. Det gjør vi ved kall på metoden `printStackTrace()` på unntaket `e`.

2.1 Innlesning av en typisk fil

Innlesning fra fil fungerer på samme måte som innlesning fra terminal. Vi kan bruke de samme `Scanner`-metodene for å lese inn informasjon:

```
String linje = innFil.nextLine();
int heltall = innFil.nextInt();
double desimaltall = innFil.nextDouble();
```

Men, ofte har filer en bestemt struktur som gjentar seg. La oss se på en fil på et slikt format:

```
Ola
Tuva
Heidi
Lars
Gunnar
Ingrid
.....
```

Her har vi en fil med en masse navn, et navn på hver linje. Vi vet ikke hvor mange linjer fila vår har - hvordan kan vi da lese fra den?

Løsningen ligger i en metode i `Scanner`:

- `hasNextLine()`: Denne metoden returnerer `true` hvis det finnes flere linjer i fila, `false` ellers.

For å lese inn hele filen, kan vi da gjøre følgende:

```
while (innFil.hasNextLine()) {
  String linje = innFil.nextLine();
}
```

Hvis man har lyst til å lese et og et ord i fila kan man bruke metoden `hasNext()`:

```
while (innFil.hasNext()) {
  String ord = innFil.next();
}
```

2.2 Splitte opp en streng

I noen tilfeller møter vi på filer med følgende format:

```
ord1;ord2;ord3;ord4;
ord1;ord2;ord3;ord4;
....
```

Vi har lyst til å få tak i de ulike ordene og gjøre noe med de i programmet vårt. En strategi for å lese denne filen er å lese inn en og en linje, dele denne opp i ord, og gjøre noe med ordene.

For å få til dette kan man bruke en metode i klassen `String` som heter `split()`. Denne metoden deler en tekst opp i ord, basert på et gitt skilletegn, og lagrer ordene i en array. Teksten selv endres ikke. Et eksempel på bruk:

```
String linje = "hei;meg;deg";
String[] ord = linje.split(";");
```

Resultatet av kallet på metoden `split(";")` er en array, der hver plass inneholder et av ordene i setningen:

```
String[] ord = {"hei", "meg", "deg"};
```

For å lese inn filen gitt over, kan vi da gjøre følgende:

```
while(innFil.hasNextLine()) {
  String linje = innFil.nextLine();
  String[] ord = linje.split(";");
  String ord1 = ord[0];
  String ord2 = ord[1];
  // osv...
}
```

Vi leser inn linja, splitter den opp basert på skilletegnet ";", og gjør noe med ordene etterpå.

3 Skrive til fil

For å skrive til fil bruker vi klassen `FileWriter`. Denne klassen ligger i `java.io` og må importeres herfra:

```
import java.io.*;
```

Som ved innlesning fra fil kan det oppstå feilsituasjoner, så vi trenger også her `try-catch`-blokkene.

```
try {
  FileWriter utFil = new FileWriter("utfil.txt");
} catch (IOException e) {
  e.printStackTrace();
}
```

Her trenger vi bare å sende med navnet på filen vi vil skrive til. Merk at vi ikke krever at denne filen finnes fra før, `FileWriter` oppretter filen for deg.

Hvis vi har lyst til å skrive i slutten av en allerede eksisterende fil, legge til informasjon, sender vi med et ekstra argument til `FileWriter`:

```
try {
  FileWriter utFil = new FileWriter("utfil.txt", true);
} catch (IOException e) {
  e.printStackTrace();
}
```

3.1 Skriving

For å skrive til fil, bruker vi metoden `write`:

```
try {
  FileWriter utFil = new FileWriter("utfil.txt", true);

  // skrive til filen:
  utFil.write("her er en tekst, woho! \n");
  utFil.write("setning nummer 2\n");

  // lukk filen:
  utFil.close();
} catch (IOException e) {
  e.printStackTrace();
}
```

Det er to viktige ting å huske på her:

- Metoden `write()` gir deg ikke linjeskift, så du må huske å legge til teksten "$\backslash n$" på slutten av setningen din. Denne gir deg linjeskift.
- Det er viktig å lukke skriveren når du er ferdig, uten dette blir ikke filen skrevet til. Dette gjør du med metoden `utFil.close()`.

3.2 *Formattert utskrift

Dette er *-stoff og er ment for de mest interesserte studentene.

For å skrive formattert, kan vi bruke en Java-klasse som heter `PrintWriter`. `PrintWriter` tilbyr blant annet følgende metoder:

- `print(String s)` - som skriver ut en tekst.
- `println(String s)` - som skriver ut en tekst med linjeskift.
- `printf(String s, argumenter)` - som skriver formattert utskrift.

Når vi oppretter et `PrintWriter`-objekt sender vi med den gamle filskriveren vår, `FileWriter`. Et eksempel på formattert utskrift med desimaltall (`"%f"`), tekster (`"%s"`) og heltall (`"%d"`) kan du se her:

```
try {
  FileWriter fileWriter = new FileWriter("out.txt");
  PrintWriter printWriter = new PrintWriter(fileWriter);

  String navn = "Tuva";
  double desimal = 3.14159265359;
```

```
  int heltall = 365;

  // formattert utskrift
  printWrite.printf("%-5s | %2.2f | %d \n", navn, desimal, heltall);

  printWriter.close();
} catch(IOException e) {
  e.printStackTrace();
}
```

Dette gir gir følgende formatterte resultat:

```
Tuva  | 3,14 | 365
```

Under kommer en liten oversikt over de vanligste formatteringsvalgene:

- `%s` formatterer en tekst.
- `%d` formatterer et heltall.
- `%f` formatterer et desimaltall.
- `%10s` formatterer en tekst som tar 10 karakterer med plass. Merk at hvis teksten er lengre enn 10 karakterer, vil den bruke større plass.
- `%-10s` venstrestiller teksten.
- `%.2f` formatterer et desimaltall med to sifre etter komma.