Løsningsforslag ukeoppg. 8: 12. - 18. okt (INF1000 - Høst 2011)

*Mer om **tekster, metoder, og objekter** (kap. 1-8 og 12 i "Rett på Java" **3. utg.**)*

> ***NB!** Legg merke til at disse er **løsningsforslag**. Løsningene dine trenger ikke å være like med disse forslag for å være riktige. Det er vanlig i programmering at samme oppgave kan løses på mange vidt forskjellige måter, og alle fremgangsmåter er ok i INF1000 så lenge de leder fram til riktig resultat og oppfyller kravene som står i oppgaveteksten.*

Mål
Få øvelse i teorien du trenger til å løse Oblig 3, og litt repetisjon.

🔑 Oppgave merket med nøkkelsymbol er plukket ut som spesielt representativ for de viktigste temaene fra ukens forelesning, og alle bør ha som minimumsmål å løse denne selvstendig.

Oppgaver

For de som ønsker å lese en annerledes og fin forklaring av klasser og objekter anbefaler vi "objekter.pdf" i Marit Nybakkens notater, skrevet av en populær tidligere gruppelærerinne i kurset.

1. 🔑 **Vanlige feilmeldinger i Oblig 3:**
   Finn feilene i følgende program og foreslå hvordan de kan rettes. Programmet består av to klasser: Student, som bare har én objektvariabel: navn; og Gruppe, som har en array med opptil 40 studenter og en *konstruktør* med programkode som illustrerer bruk av klasser og objekter.

```
class Student {
   String navn;
}

class Gruppe {
   Student[] studenter = new Student[40]; // Array med Student-objekter
   int antStudenter; // Antall plasser i arrayen over som er i bruk

   public static void main(String[] args) {
      Gruppe g = new Gruppe();

      skrivAntall();
      /* Spørsmål:
       * a) Hvorfor gir linjen over følgende feilmelding?  Hvordan unngå det?
       *        Gruppe.java:12: non-static method skrivAntall() cannot
       *                        be referenced from a static context
       *            skrivAntall();
       *            ^
       *  Tips: Du kan lese mer om dette på side 201 (og 157) i læreboka.
       *    Problemet skyldes at main-metoden er en "klassemetode", dvs. deklarert
       *    med "static", og derfor kan man ikke referere til "objektvariabler"
       *    eller "objektmetoder" (dvs. variabler og metoder som ikke er deklarert
       *    me nøkkelordet static) direkte fra en static metode uten å gå
       *    via en peker, f.eks. pekeren g ovenfor.  Dette illustrerer
       *    hvorfor vi vanligvis bare har noen få programsetninger i
       *    main-metoden som bare setter programmet i gang, ved å kalle
       *    på andre metoder i programmet.
       */
```

```
Svar:
a) Globale variabler og metoder deklarert uten nøkkelordet static
   kalles for "objektvariabler" og "objektmetoder".  Slike variabler
   og metoder kan ikke aksesseres direkte fra main (uten å gå via en
   objektpeker) fordi main-metoden er static - den er en såkalt
   "klassemetode" og ikke "objektmetode".

   I dette tilfellet skulle kallet på objektmetoden skrivAntall()
   ha gått via en objektpeker, f.eks. slik: g.skrivAntall(),
   fordi kallet på metoden er gjort i static-metoden main.
```

```
   }

   Gruppe() {
      // b) Den vanligste feilmeldingen i Java er "cannot find symbol".
      //    Følgende linje gir feilmeldingen vist under.  Hva er feil i dette
      //    tilfellet, og hvordan kan vi rette det?
      antallStudenter = 0;
      /*                Gruppe.java:32: cannot find symbol
       *                symbol  : variable antallStudenter
       *                location: class Gruppe
       *                        antallStudenter = 0;
       *                        ^
       *  Tips: Årsaken til feilmeldingen "cannot find symbol" er alltid
       *    at noe ikke er deklarert riktig (f.eks. at det ikke er deklarert
       *    på riktig sted eller er stavet feil).  Feilen er heldigvis
```

```
        *    lett å fikse: se i feilmeldingen hva det var som ikke var
        *    deklarert, og sjekk at du har deklarert det riktig (på riktig
        *    sted i programmet, og stavet riktig).  Du kan finne hva det er
        *    som ikke var riktig deklarert på 2. linje i feilmeldingen, der
        *    det står "symbol".  I eksemplet over er problem-symbolet
        *    "variable antallStudenter".  Legg også merke til linjenummeret
        *    og posisjonen på linja som Java-kompilatoren også gir deg.
        */
```

```
b) antStudenter er stavet feil.
```

```
        // c) Følgende linje gir "NullPointerException" her.  Hvilken setning
        //    har vi glemt å utføre før dette?  (Husk at studenter[] er en array)
        studenter[antStudenter].navn = "Trine";
        antStudenter++;
        /*  Tips: Feilmeldingen NullPointerException skyldes oftest at man har
         *    forsøkt å bruke prikk-notasjon på en peker som hadde verdien null.
         *    Det er ikke lov (det gir ingen mening å si f.eks. null.navn).
         *
         *  Slik fikser du NullPointerException-feil:
         *    1. Se i feilmeldingen hvilket linjenummer Java fant feilen i.
         *    2. Se hvilke pekere i den angitte linjen som kan ha vært null
         *       under kjøring av programmet, særlig blandt de som har prikk-
         *       notasjon etter seg, f.eks. hvis linjen inneholder uttrykket:
         *         hyblene[rad][kol].leietager.navn  ...så kan null-pekeren
         *       være hyblene[rad][kol] eller hyblene[rad][kol].leietager
         *    3. Endre programmet slik at pekeren som forårsaket feilen ikke
         *       kan være null når programmet kommer til den aktuelle linjen.
         *       Hvis problemet var en peker med prikknotasjon kan du legge
         *       til en if-test på at pekeren != null før den aktuelle linjen.
         *    Det som står foran prikken må altså være noe som peker på
         *    et allerede-opprettet objekt av riktig klasse (og ikke null), på
         *    det tidspunktet under programutførelsen når setningen blir utført.
         *
         *    4. Hvis linjen har flere pekere og du ikke finner hvilken som ga
         *       NullPointerException kan du legge inn en testutskrift rett før
         *       linja, og skrive ut én av dem, f.eks.
         *         System.out.println("test1:" + hyblene[rad][kol].leietager);
         *       Med denne fremgangsmåten vil du alltid kunne finne nullpekeren.
         */
```

```
c) Her mangler å opprette studentobjekt, f.eks. slik:
    studenter[antStudenter] = new Student();
```

```
        // d) Skriv det som mangler for at studenter[1].navn skal bli "Martin".
        Student s1 = new Student();
        s1 _________ = "Martin";
        studenter[antStudenter++] = _________ ;
```

```
d) s1.navn = "Martin";
studenter[antStudenter++] = s1 ;
```

```
        // e) Hva mangler her for at if-testen skal kunne gi true?
        //    Tips: Husk at tekster må sammenlignes med andre tekster.
        if (studenter[1].equals("Martin")) {
            System.out.println(true);
        }
```

```
e) Det mangler .navn:
if (studenter[1].navn.equals("Martin")) {
```

```
        // f) Hvorfor gir følgende løkke NullPointerException på linjen med
        //    System.out..? (anta at eneste kode som er blitt utført når
        //    programmet kommer hit er det som står i linjene over, med feilene
        //    rettet).  Også, hvordan kan betingelsen i første linje endres for å
        //    unngå NullPointerException?  Tips: Tenk arrayplasser og != null.
        for (int i = 0; i < studenter.length; i++) {
            Student s2 = studenter[i];
            System.out.println(s2.navn);
        }
```

```
f) Her blir det NullPointerException fordi det bare er noen få elementer
   i arrayen som inneholder Student-objekt, og resten har null-peker.
   Problemet er at "i" går for langt, og forbi plassene i arrayen der det
   ligger data.  En løsning er å endre betingelsen i for-løkka slik at "i"
   bare teller opp til antStudenter:
      for (int i = 0; i < antStudenter; i++) {

   En annen mulig løsning er å sjekke at elementet i arrayen
   studenter[] på plass "i" ikke er null:
      for (int i = 0; studenter[i] != null && i <
   studenter.length; i++) {
```

```
        // g) Finn en annen måte å unngå NullPointerException her uten å endre
        //    første linje under, men ved å legge til en if-setning inne i løkka:
        for (int i = 0; i < studenter.length; i++) {
            Student s3 = studenter[i];
```

```
g)  if (s3 != null) {
        System.out.println(s3.navn);
    }
```

```
        }

        // h) Hvorfor gir følgende linje i programkoden denne feilmeldingen:
        //    ArrayIndexOutOfBoundsException: 40
        studenter[40] = new Student();
```

```
h) Fordi indeksene i arrayen går fra 0 til 39.
```

```
        // i) Hva vet vi om uttrykket merkert med "___" på neste linje hvis
        //    linjen gir feilmeldingen "ArrayIndexOutOfBoundsException: -1":
        s1 = studenter[ ___ ];
```

```
i) Uttrykket har verdi -1, så det kan f.eks. ha vært:
     s1 = studenter[j];   ...der j har verdien -1 på dette tidspunktet.
```

```
        // j) Hva er feil her?  Feilmeldingen dette gir er:
        //                  Gruppe.java:103: incompatible types
        //                  found   : java.lang.String
        //                  required: Student
        studenter[2] = "Eva";
```

```
j) Dette er et forsøk på å lagre en String-verdi inne i en Student-peker.
   Det går ikke.  Tekster må lagres i tekst-variabler, f.eks.:
         studenter[2].navn = "Eva";
```

```
        // k) Hva er feil her?  Feilmeldingen er: cannot find symbol
        //                  symbol  : constructor Student(java.lang.String)
        Student lars = new Student("Lars");
```

```
k) Som du ser av feilmeldingen, så finner ikke Java-kompilatoren noen
   konstruktør i klassen Student med en String som parameter.  Løsningen er
   enten å lage en slik konstruktør i klassen Student (med én String som
   parameter), eller bruke den vanlige automatisk-definerte konstruktøren
   (som ikke har parametre), og lagre navnet i en egen programsetning:
       Student lars = new Student();
       lars.navn = "Lars";
```

```
        // l) Hva er feil her?  Feilmeldingen dette gir er:
        //                  Gruppe.java:114: incompatible types
        //                  found   : int
        //                  required: boolean
        if (antStudenter = 0) {
            system.out.println("Ingen student registrert!");
        }
```

```
l) Det skulle stått to lik-tegn: "==".  Når det bare står ett lik-tegn
   er hele uttrykket en tilordning, som gir 0 som verdi, og det passer
   ikke som betingelse i en if-setning (betingelser må ha boolsk verdi).
```

```
        // m) Hvorfor klager Java også med: "package system does not exist"?
```

```
m) System er stavet feil.
```

```
        // n) Når vi har rettet alle feilene i a) til m) ovenfor, så gir fortsatt
        //    if-setningen under NullPointerException.  Hvordan unngår vi det?
        boolean funnet;
        for (int i = 0; i < studenter.length; i++) {
            Student stud = studenter[i];
            if (stud.navn.equals("Eva")) {
                 funnet = true;
                 System.out.println(stud.navn);
            }
        }
```

```
n) Ved f.eks. å legge til en ekstra-betingelse i if-setningen, f.eks.:
    if (stud != null && stud.navn.equals("Eva")) {
```

```
        // o) Løkken ovenfor stopper ikke når navnet "Eva" blir funnet.
        //    Hvordan kan vi få løkken til å stoppe når navnet blir funnet?
```

```
o) Ved å legge til "! funnet" som en ekstra-betingelse i selve for-løkken:
       for (int i = 0; i < studenter.length && !funnet; i++) {
```

```
        // p) Løkken ovenfor gir ingen melding til bruker når navnet ikke blir
        //    funnet.  Hvordan kan vi programmere utskrift av en slik melding?
        //    Og hvordan kan vi unngå at Java da skal klage om at "variable
        //    funnet might not have been initialized"?
```

```
p) Ved å sette startverdien "false" i funnet, og legge til rett etter
    løkken en if-setning som skriver ut en melding om "ikke funnet"
    hvis !funnet.
```

```
        // q) Anta at navnet til studenter[2] er "Eva".  Hvorfor endrer ikke
        //    følgende kode navnet til studenter[2]?  Hvordan kan det ordnes?
        //    (slik at studentobjektet i "ny" overføres til studenter[2]).
        Student ny = new Student();
        Student studPeker;
        ny.navn = "Heidi";
        studPeker = studenter[2];
        studPeker = ny;
    }
```

```
q) Vi har 2 student-objekter i minnet: Eva som er pekt på av studenter[2],
   og Heidi som er pekt på av ny.  Det som kodebiten gjør er å sette
   studPeker til å peke på det første, og deretter flyttes (bare) denne
   studPeker til å peke på ny i stedet.  Det endrer ikke på hva pekeren
   studenter[2] peker på.  For å ordne det må man bruke en tilordning
   hvor studenter[2] står på venstre side av "="-tegnet, f.eks. ved
   å legge til følgende som en ny linje rett etter kodebiten i oppgaven:
       studenter[2] = studPeker;
```

```
    void skrivAntall() {
       System.out.println("Antall studenter: " + antStudenter);
    }

    // r) Hvorfor får vi feilmeldingen: "<identifier> expected" på denne linjen:
    System.out.println("Slutt");
```

```
r) Det er bare deklarasjoner (av variabler og metoder) som
   kan plasseres her, i dette området utenfor metoder.  Andre typer
   setninger, som "System.out.." er bare lov å plassere inne i metoder.
```

```
 }
```

Flere debuggings-tips:

Både feilmeldingene fra kompilatoren (javac) og kjøresystemet til Java (java) gir deg vanligvis linjenummeret der feilen oppsto, og navnet til feilen. Disse opplysningene er nyttige for å finne og rette feilen. Start alltid med å rette den første feilen som Java fant.

Tips til krøllparentes-feil:

En annen type feil som er lett å gjøre i Java er krøllparentes-feil, f.eks. å glemme en krøllparentes et sted, ha en for mye, en som "vender" feil vei, o.l. Disse problemene gir opphav til mange typer feilmeldinger, bl.a.:

- «**class, interface, or enum expected**», «**<identifier> expected**»
- «**reached end of file while parsing**», «**'}' expected**»,
- «**illegal start of type**», «**illegal start of expression**», «**'else' without 'if'**», m.fl...

To gode tips for å unngå og rette slike feil er:

1. Alltid holde programmet riktig formatert med "innrykk" underveis mens du skriver det: Start et nytt "nivå" med innrykk rett etter hver "{" (åpnings-krøllparentes) du skriver, dvs. at alle linjene etter en slik krøllparentes får f.eks. 4 flere mellomrom foran enn linjene over ...frem til første "}", da du forminsker ett innrykk-nivå, osv. *Alle programmer i disse ukeoppgavene er formatert på denne måten*

```
    Alle programmer i disse ukeoppgavene er formatert på denne måten.


    2.  Når et av de ovennevnte feilene dukker opp, bruk funksjonen i Emacs
          (eller Eclipse, e.l.) som setter inn riktig innrykk i hele programmet.
          I Emacs kan det gjøres ved å markere hele programmet (Ctrl-x h) og velge
          "Java > Indent Line or Region" fra menyen øverst i Emacs-vinduet.
          Da blir det satt riktig innrykk i hele programmet og det blir lett å se
          hvor feilen er, ved å starte fra toppen og fortsette nedover til du
          finner noe som har havnet på feil innrykksnivå: Sjekke at alle klasser
          havnet i "første innrykks-nivå" (dvs. at det ikke står noen mellomrom
          før nøkkelordet class), og at første linje i alle metoder
          havnet i 2. nivå av innrykk (dvs. at det er 4 mellomrom rett før
          nøkkelordet void), og at selve innmaten i metodene er i 3. nivå
          (har 8 mellomrom foran), osv.



 Flere tips til feilsøking og -retting finner du i Marit Nybakkens feilmeldinger.pdf.
```

2. **Utskrift av tabell:** *kap. 3, oppg. 2 (side 72)*
   [*NB! Alle oppgavene denne uken er repetisjonsoppgaver.*]
   (a) Lag et program som benytter programpakken easyIO til å skrive ut følgende tabell på skjermen. *Tips: Se side 53 i læreboka.*

```
Bilmerke        Årsmodell       Reg.nr.
----------------------------------------
Mercedes        1999            UE65660
Ford            2003            ZE95523
Toyota          2006            DK53401
----------------------------------------
```

```
// Basert på løsningsforslaget i lærebokens hjemmeside:
// http://www2.universitetsforlaget.no/java/eksempler.php
import easyIO.*;

class Biltabell {
    public static void main(String[] args) {
        Out skjerm = new Out();
        int bredde = 16;
        String linje = "----------------------------------------";

        // Overskrifter:
        skjerm.out("Bilmerke", bredde);
        skjerm.out("Årsmodell", bredde, Out.LEFT);
        skjerm.outln("Reg.nr.");

        skjerm.outln(linje);

        // Innmat i tabellen, linje for linje:
        skjerm.out("Mercedes", bredde);
        skjerm.out("1985", bredde, Out.LEFT);
        skjerm.outln("UE65660");

        skjerm.out("Ford", bredde);
        skjerm.out("1987", bredde, Out.LEFT);
        skjerm.outln("ZE95523");

        skjerm.out("Toyota", bredde);
        skjerm.out("2003", bredde, Out.LEFT);
        skjerm.outln("DK53401");

        skjerm.outln(linje);
    }
}
```

   (b) *(For spesielt interesserte.)* Løs samme oppgave ved hjelp av **printf** i stedet for EasyIO. Du kan bruke %s for å skrive ut en String, og %-16 for å angi venstrejustering av String-en på 16 plasser, for eksempel:

```
System.out.printf("%-16s", "Ford");
```

```
class Biltabell {
    public static void main(String[] args) {
        int bredde = 16;
        String linje = "----------------------------------------";
```

```
        // Overskrifter:
        System.out.printf("%-16s%-16s%s\n", "Bilmerke", "Årsmodell", "Reg.nr.");

        System.out.println(linje);

        // Innmat i tabellen, linje for linje:
        System.out.printf("%-16s%-16d%s\n", "Mercedes", 1985, "UE65660");
        System.out.printf("%-16s%-16d%s\n", "Ford", 1987, "ZE95523");
        System.out.printf("%-16s%-16d%s\n", "Toyota", 2003, "DK53401");

        System.out.println(linje);
    }
}
```

3. **Filer, store bokstaver, og args:** *kap. 3 (side 56), og oppg. 7 (side 73)*
(a) **Filkopi:** Lag et program som leser inn en fil og kopierer innholdet over til en annen fil. Filen skal leses inn *ett tegn av gangen* ved hjelp av `inChar()`. Du kan ta utgangspunkt i programmet vist under, fra side 56 i læreboka, som leser en fil et tegn av gangen, men skriver innholdet i filen ut på skjermen. Endre programmet slik at det skriver ut til til en annen, nyopprettet fil, i stedet for å skrive ut på skjermen; endre klassenavnet til Kopi, og utvid programmet slik at det ber bruker taste inn filnavnene for de to filene (original og kopi).

```
import easyIO.*;

class Tegnleser {
    public static void main(String[] args) {
        In fil = new In("minfil.txt");
        int antall = 0;

        while (!fil.endOfFile()) {
            char tegn = fil.inChar();
            System.out.print(tegn);
            antall++;
        }
        System.out.println("Antall tegn: " + antall);
    }
}
```

```
import easyIO.*;

class Kopi {
    public static void main(String[] args) {
        Out skjerm = new Out();
        In tast = new In();

        skjerm.out("Filnavn original: ");
        String innfilnavn = tast.inLine();

        skjerm.out("Filnavn kopi: ");
        String utfilnavn = tast.inLine();

        In innfil = new In(innfilnavn);
        Out utfil = new Out(utfilnavn);

        int antall = 0;
        while (!innfil.endOfFile()) {
            char tegn = innfil.inChar();
            utfil.out(tegn);
            antall++;
        }
        innfil.close();
        utfil.close();
        System.out.println("Antall tegn kopiert: " + antall);
    }
}

KJØREEKSEMPEL:
> java Kopi
Filnavn original: minfil.txt
Filnavn kopi: minfil2.txt
Antall tegn kopiert: 38
```

(b) **Store bokstaver:** Ta utgangspunkt i programmet vist ovenfor, og endre det slik at det skriver ut tegnene fra den innleste filen til skjerm, men med alle små bokstaver konvertert til *store bokstaver*. Følgende setninger viser hvordan man kan konvertere innholdet i en `char`-variabel `c` til store eller små bokstaver:

```
char c = 'x';
char c2 = Character.toUpperCase(c);
char c3 = Character.toLowerCase(c);
```

```
import easyIO.*;

class StoreBokstaver {
    public static void main(String[] args) {
        In innfil = new In("minfil.txt");
        int antall = 0;

        while (!innfil.endOfFile()) {
            char tegn = innfil.inChar();
            tegn = Character.toUpperCase(tegn);
            System.out.print(tegn);
            antall++;
        }
        System.out.println("Antall tegn: " + antall);
    }
}

KJØREEKSEMPEL:
> java StoreBokstaver
A
CANIS FAMILIARIS BETYR HUND
15
3.14
Antall tegn: 38
```

(c) **Antall ord:** Når man skriver artikler for publisering er det ofte grenser for hvor mange ord de kan inneholde. Lag et program som teller *antall ord* i en fil. Filnavnet kan du be bruker taste inn når programmet starter.

```
import easyIO.*;

class AntallOrd {
    public static void main(String[] args) {
        In tast = new In();

        System.out.print("Filnavn (for telling av antall ord): ");
        String filnavn = tast.inLine();
        In innfil = new In(filnavn);

        int antall = 0;
        while (!innfil.endOfFile()) {
            innfil.inWord();
            antall++;
        }
        System.out.println("Antall ord: " + antall);
    }
}

KJØREEKSEMPEL:
$ java AntallOrd
Filnavn (for telling av antall ord): minfil.txt
Antall ord: 3
```

(d) **args[]:** Lag deretter en annen utgave av programmet som tar filnavnet fra første *kommandolinjeargument*, dvs. `args[0]`. "Kommandolinjeargumenter" er evt. tilleggs-ord som bruker angir i selve `java`-kommandolinjen når hun kjører programmet. For eksempel, hvis bruker starter programmet med følgende kommando:

```
$ java AntallOrd fil.txt ord2
```

...så putter Java de to siste tilleggs-ord i arrayen `String args[]` (som vi har sett øverst i alle programmene våre til nå uten å bruke det). I dette tilfellet vil Java sørge for at `args[0]` får verdien "`fil.txt`" når programmet starter, og `args[1]` får verdien "`ord2`".

```
import easyIO.*;

class AntallOrd {
    public static void main(String[] args) {
        In innfil = new In(args[0]);
        int antall = 0;

        while (!innfil.endOfFile()) {
            innfil.inWord();
            antall++;
        }
        System.out.println("Antall ord: " + antall);
    }
}

KJØREEKSEMPEL:
> java AntallOrd minfil.txt
Antall ord: 3
```

4. **Filbehandling linje for linje:** *(eksempel side 57 i læreboka)*
(a) Studér følgende program, fra side 57 i læreboka, som leser en fil en linje av gangen, og skriver den ut på skjermen med linjenummer foran i hver linje. Endre programmet slik at det i stedet for å skrive ut alle linjene bare skriver ut en melding til slutt om *hvor mange linjer* og *hvor mange tegn* filen inneholder. For å telle antall tegn kan du bruke en variabel som summerer verdiene av `linje.length()`.

```
import easyIO.*;

class Linjeleser {
    public static void main(String[] args) {
        In fil = new In("minfil.txt");
        int linjenummer = 0;

        while (!fil.endOfFile()) {
            String linje = fil.readLine();
            linjenummer++;
            System.out.println(linjenummer + " " + linje);
        }
    }
}
```

```
import easyIO.*;

class Linjeleser {
    public static void main(String[] args) {
        In innfil = new In("minfil.txt");
        int antLinjer = 0;
        int antTegn = 0;

        while (!innfil.endOfFile()) {
            String linje = innfil.readLine();
            antLinjer++;
            antTegn += linje.length() + 1;  // + 1 for å ta med linjeskiftet "\n"
        }
        System.out.println("Antall linjer: " + antLinjer);
        System.out.println("Antall tegn: " + antTegn);
    }
}

KJØREEKSEMPEL:
> java Linjeleser
Antall linjer: 4
Antall tegn: 38
```

(b) Ta utgangspunkt i programmet vist ovenfor, og endre det slik at det skriver ut linjene på skjerm, men med alle små bokstaver konvertert til **store bokstaver**. Følgende setninger viser hvordan man kan konvertere bokstavene i en `String`-variabel til store bokstaver:

```
String s = "Jeg ER 18 år";
String s2 = s.toUpperCase();
// Nå er s2 tekststrengen "JEG ER 18 ÅR"
```

```
import easyIO.*;

class Linjeleser {
    public static void main(String[] args) {
        In innfil = new In("minfil.txt");
        int linjenummer = 0;

        while (!innfil.endOfFile()) {
            String linje = innfil.readLine();
            linje = linje.toUpperCase();
            linjenummer++;
            System.out.println(linjenummer + " " + linje);
        }
    }
}

KJØREEKSEMPLER:
> java Linjeleser
1 A
2 CANIS FAMILIARIS BETYR HUND
3 15
4 3.14
```

5. **Public/private, klasse-/objekt-variabler:** *i kapittel 8 (side 177)*

(a) **Private:** Vi tar en ny titt på bankkonto-programmet fra oppgave 4 i Ukeoppgaver 6. Endre deklarasjonen

(a) **Private:** Vi tar en ny titt på bankkonto-programmet fra oppgave 4 i ukeoppgaver 6. Endre deklarasjonen av *saldo* i klassen Konto til å være *private*, og vis hvordan vi da kan få tak i *saldo* fra den andre klassen. *Tips: Bruk get-metoden.*

```
class KontoEksempel {
    public static void main(String[] args) {
        Konto k1 = new Konto();
        k1.bestemKontonr();

        k1.settInn(500);
        System.out.println("Saldo er: " + k1.saldo);

        k1.taUt(300);
        System.out.println("Saldo er: " + k1.saldo);
    }
}

class Konto {
    int kontonr;
    int saldo;
    String eier, adresse;
    double rente = 2.5; // 2.5% per år
    static int nummer = 0; // Klassevariabel

    void bestemKontonr() {
        nummer++;
        kontonr = nummer;
    }

    void settInn(int innskudd) {
        saldo = saldo + innskudd;
    }

    boolean taUt(int uttak) {
        if (uttak > saldo) {
            return false;
        }
        saldo = saldo - uttak;
        return true;
    }

    int getSaldo() {
        return saldo;
    }
}

KJØREEKSEMPEL:
$ java KontoEksempel
Saldo er: 500
Saldo er: 200
```

```
Endre følgende linje i klassen Konto fra:
        int saldo
Til:
        private int saldo

Og endre følgende linje som står to steder i klassen KontoEksempel fra:
        System.out.println("Saldo er: " + k1.saldo);
Til:
        System.out.println("Saldo er: " + k1.getSaldo());
```

(b) **Klassevariabler:** De fleste metoder og variabler i programmet ovenfor er *objekt*-variabler og *objekt*-metoder, men det er én *klasse*-variabel og én *klasse*-metode i programmet. Finn disse, og diskuter hvordan de er annerledes enn objekt-variantene. Hvordan fungerer klasse-variabelen som står i programmet, og hva ville skjedd hvis vi tok bort nøkkelordet *static* fra deklarasjonen av variabelen?

Klassemetoden er metoden **main** i klassen KontoEksempel, og klassevariabelen er **int nummer** i klassen Konto.

Forskjellen mellom **klassevariabler** og objektvariabler er at hvert objekt man oppretter av klassen (f.eks. ved hjelp av nøkkelordet new) får sin egen utgave av objektvariablene (som kan ha sin egen verdi), men ikke av klassevariablene. Det finnes bare én verdi i hver klassevariabel som vil alltid være den samme i alle objektene av klassen.

Forskjellen mellom **klassemetoder** og objektmetoder er at klassemetoder bare kan aksessere klassevariabler (og evt. lokale variabler deklarert i selve metoden) og klassemetoder, men ikke objektvariabler eller objektmetoder; mens **objektmetoder** derimot kan aksessere både objekt- og klasse-variabler og -metoder.

Hvis vi tar bort nøkkelordet *static* fra deklarasjonen av variabelen *nummer* i klassen Konto, så gjør vi

Hvis vi tar bort nøkkelordet *static* fra deklarasjonen av variabelen *nummer* i klassen Konto, så gjør vi den om til fra å være klassevariabel til å være en vanlig objektvariabel. Det vil føre til at hvert kontoobjekt vil nå få sin egen kopi av variabelen, med egen verdi, som dermed vil få startverdi 0 i alle Konto-objektene. Det vil resultere i at alle kontoer får samme kontonummer, som vil være 1 i alle kontoer.

Løsningsforslag

Kommer... Noen av oppgavene fra læreboka har løsningsforslag på Lærebokens hjemmeside.

Tibakemelding om dette oppgavesettet kan du skrive i bloggen eller sende på mail til josek *[a]* ifi.uio.no