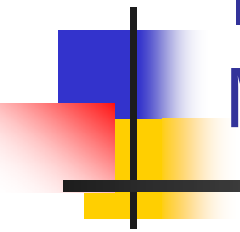
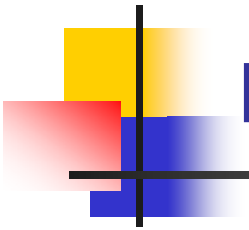


Inf1000 (Uke 8)
Mer om objekter og filbehandling

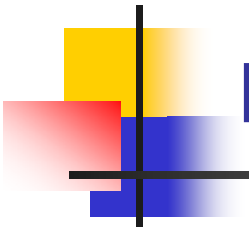
Grunnkurs i programmering
Institutt for Informatikk
Universitetet i Oslo

Anja Bråthen Kristoffersen og Are Magnus Bruaset



Dagens plan

- Mer om metoder/klasser/objekter
  - **`private/public/package`**
- Flere eksempler med objekt orientert programmering
- Filbehandling
  - å ta vare på resultatene for ettertiden
  - å lese inn data fra fil
- Tilslutt noen råd til oblig 3

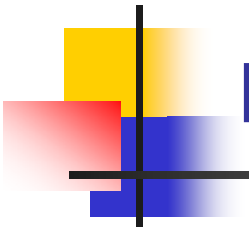


Hva er en metode

- En metode er et valgfritt antall programsetninger som vi har gitt et navn
- All kode i programmet er inne i en eller annen metode (som igjen er inne i en klasse)
  - med unntak av deklarering av klassevariable og objektvariable som skjer utenfor metoder
- Skiller mellom
  - å *deklarere* en metode (= skrive Javakode for metoden)
  - *Utføre* en metode (det som skjer når vi kaller den)
    - Når vi deklarerer en metode, skjer det 'ingen ting'
    - Hvis en metode aldri blir kalt på blir den heller aldri brukt
- En metode blir utført hver gang den kalles. Kallet skjer fra koden i en annen metode:
  - da hopper utførelsen av programmet til starten av den kalte metoden
  - har den kalt metoden parametere, kopieres verdiene brukt i kallet til metodens parameter (de er som lokale variable i den kalte metoden)

Kall på en metode: **`skrivAntallGanger()'`** i objektet **`'b'`** tre ganger

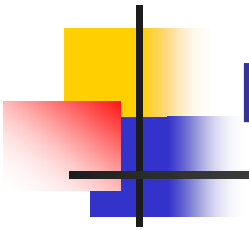
```
class BB {
  int antall = 0;

  void skrivAntallGanger() {
       antall++;
       System.out.println(" Du har kalt meg: " +
                                  antall + " ganger");
  }
}

class AA
{
    public static void  main ( String[] args)  {
      BB b = new BB();
      b.skrivAntallGanger();  // Kall nr. 1
      b.skrivAntallGanger();  // Kall nr. 2
      b.skrivAntallGanger();  // Kall nr. 3
    }
}
```



Hva skjer når vi kaller en metode

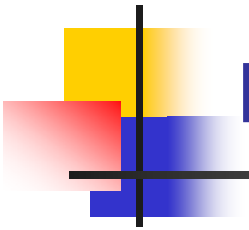
- Når vi kaller en metode, blir det opprettet et **metodeobjekt,** verdiene brukt i kallet blir kopiert over til parametrene.
- Dette metodeobjektet
  - inneholder alle lokale variabler og parametrene til metoden
  - når setningene i metoden utføres, brukes disse variablene og parametrene av metoden
  - metodeobjektet fjernes automatisk når metoden er ferdig utført

```
class C {

  int  skrivAntall(int i){
       System.out.println(" Du har kalt meg med: " + i);
       return i+10;
}}

class D
{
  static int dobbel( int k) {
       return 2*k;
  }

  void gjørMye(C cc, int v) {
       System.out.println(" gjørMye kalt");
       int j = cc.skrivAntall(v);
       System.out.println(" 1.verdien av j: " + j);
       j = dobbel(j);
       System.out.println(" 2.verdien av j: " + j);
       System.out.println(" 3.verdien av skrivAntall(j): "
                                  + cc.skrivAntall(j) );
  }

  public static void  main (String[] args)  {
      C c = new C();
      D megSelv  = new D();

      megSelv.gjørMye(c,2);
}}
```



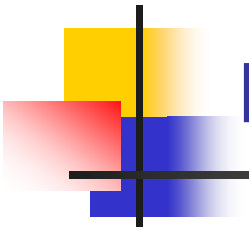
Hvorfor bruke metoder

- Vi deler opp programmet i metoder fordi:
  - Noen programsetninger brukes *flere* steder, eller:
  - Vi vil dele opp programmet i mindre deler
    - Alle metoder bør være små og oversiktlige!
  - Hver del gjør noe veldefinert som fremgår av navnet:
    - regner ut en bestemt formel
    - skriver ut en meny
    - leser noen data fra terminal eller fil
    - skriver ut opplysninger på skjermen
    - .....



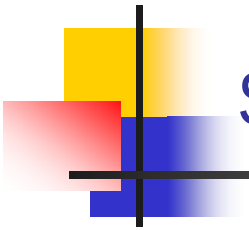
Problemløsning med metoder

- Når vi har laget en metode, og vi har forsikret oss om at den er 'riktig' (dvs. metoden gjør det vi vil at den skal gjøre), så har vi laget en *ny operasjon.*
- Vi kan i resten av koden se på metoden som en tilgjengelig operasjon som vi kan bruke som om den var innebygd i Java
  - eks: skrive ut en meny, regne ut en bestemt formel,..
- Vi trenger da ikke tenke på alle detaljene om *hvordan* denne operasjonen blir utført, bare *at* den blir gjort.
- En metode er et (lite) verktøy / en operasjon som kan gjenbrukes og lett løsningen av vårt større problem (hele systemet)
- Denne måte å programmere på heter *bottom-up* programmering og nyttes mye.
  - Eks: Java-biblioteket kan best forstås som en diger verktøykasse med nyttige operasjoner og datastrukturer vi kan (og ofte bør) bruke for å lage vårt program



Hva er en klasse

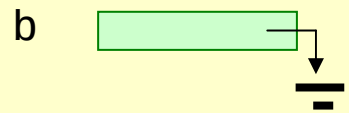
- En klasse er en beskrivelse av hvordan *ett* objekt av en bestemt type i vårt problem er.
  - Inneholder variable som beskriver egenskaper for ett slikt objekt – eks:
    - Navn, adresse, studiepoeng, kurs… for klassen Student
    - Registreringsnummer, eier, type, årsmodel for klassen Bil
  - Inneholder metoder som er fornuftig handlinger for ett slikt objekt – eks:
    - skrivUtVitnemål(), meldPåEmne(),.. i klassen Student
    - beregnÅrsavgift(), skiftEier(),.. i klassen Bil

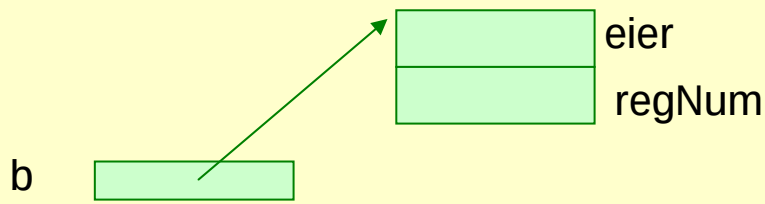


Skille mellom deklarasjon og bruk av en klasse

- Når vi deklarerer en klasse (= skriver Javakoden) skjer det 'ingen ting' i programmet
- Når vi oversetter og starter opp programmet vårt med javac og java, skjer 'lite':
  - De variable og metodene det står static foran er tilgjengelig
  - Ingen kode (med unntak av main) utføres
- Først når vi sier **new** på en klasse, får vi laget et objekt av klassen
  - Objektet inneholder alle variable og metoder som ikke har static foran deklarasjonen (objektvariable og objektmetoder)
  - Når vi sier new, kaller vi en konstruktørmetode i klassen. Først når konstruktørmetoden er ferdig, returnerer new det nye objektet med verdier fra konstruktørmetoden.

```
import easyIO.*;

class Bil {
  String eier,regNum;

  Bil (String eier, String nr) {
       this.eier = eier;
       regNum = nr;
  }
}


class E
{
   public static void  main (String[] args)  {
     Bil b;
     String eier, num;
     In tast = new In();

     System.out.print("Gi eier: ");
     eier = tast.inWord("\n");
     System.out.print("Gi nummer: ");
     num  = tast.inWord("\n");
     b = new Bil(eier,num);
     System.out.println("BIL.eier: " + b.eier +
                        ", nummer: " + b.regNum);
   }
}
```





Repetisjon av this

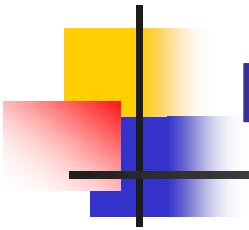
- this: en peker til seg selv.
  - Et hvert objekt inneholder en slik peker.
- this refererer til objektvariabelen.
- For at vi i forrige eksempel ikke skulle få konflikt mellom parameteren eier og objektvariabelen eier måtte vi bruke this når objektvariabelen skulle tilordnes.

```
Bil (String eier, String nr) {
        this.eier = eier;
        regNum = nr;
}
```

parameteren eier

objektvariabelen eier

```
class Konto1 {
  String eier;
  int kontoNum, saldo = 0;

  Konto1(String e) {
       eier = e;
  }

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

  boolean taUt(int beløp){
       saldo = saldo – beløp;
       return saldo > 0;
  } //bank med mulighet for et overtrekk på kontoen.
}

class Bank1
{
  Konto1[] kontiene = new Konto1[100000];

  public static void  main( String[] args)  {
     Bank1 b = new Bank1();

     for (int i = 0; i < b.kontiene.length; i++) {
        b.kontiene[i] = new Konto1("kunde nr." + i);
        b.kontiene[i].settInn(100);
     }
     System.out.println(”Antall kontoer: ” + b.kontiene.length);
     System.out.println(”Saldo konto 1: ” + b.kontiene[0].saldo);

  }
}
```



Forskjeller mellom klasser og metoder

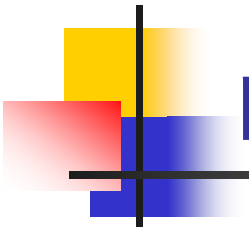
- Begge lager objekter når de kalles, men:
- Et metode-objekt:
  - fjernes når metoden returnerer
  - inneholder lokale variable og parametere som alle er skjult for resten av programmet
- Et objekt laget med **new**:
  - er i hukommelsen etter at det er laget (så lenge det minst er en peker som peker på det)
  - kan inneholde både metoder og variable, som kan nyttes av resten av programmet (med en peker og .)

Ikke alt i et objekt bør være synlig fra resten av programsystemet - innkapsling

- Vi ønsker ofte at resten av systemet bare skal se deler av et objekt
  - eks: **`int saldo`** i Konto1-objektet bør være skjult, resten av programmet skal bare bruke **`settInn()`** og **`taUt()`** metodene.
- Vi kan regulere tilgangen til variable og metoder ved å sette enten :
  - **`private`**
  - **`public`**
  - **`package`**
- foran en metode eller deklarasjonen av en variabel

For ’små’ systemer hvor alle .java filene ligger på samme filområde, gjelder:

- Skriver vi:
  - **ingenting** foran en deklarasjon/metode, så er slike deklarasjoner fullt tilgjengelige for alle annen kode kompilert på samme filområde, men usynlig /sperret for kode kompilert på andre filområder. **`package`** har samme virkning.
  - **`private`** foran en deklarasjon/metode, så er den bare synlig fra kode i metoder deklarert i *samme klasse*, usynlig/sperret for all annen kode
  - **`public`** så er metoden/variabelen synlig for all annen kode.
- Slik delvis sperring av adgang til variable, sikrer oss at vi kan bestemme fullt ut selv i en klasse hvordan en variabel skal endres.



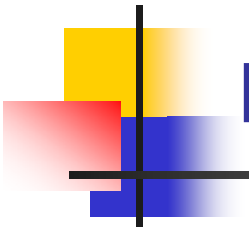
private og get/set –metoder

- I større systemer lar vi aldri andre klasser direkte endre variable, men går alltid via get- og set-metoder. Variablene er da deklarert som **`private`**.

- Eksempel: Hvordan representere en konto med private og get/set –metoder
  - **`settInn()`** og **`taUt()`** og **`overførTilAnnenKonto()`** er set-metoder
  - **`giSaldo()`** er en get-metode

```
class Konto {
   private static int nummer = 0;  //private og static (klassevariabel)
   private int kontonummer, saldo; //private
   private String eier, adresse;   //private

   Konto() {
        saldo = 0;
        bestemKontonummer();
   } //Konstruktørmetode når ingen parametere er brukt i kallet.

   Konto(String e, String adr, int s) {
        eier = e;
        adresse = adr;
        saldo = s;
        bestemKontonummer();
   } //Konstruktørmetode når parametrene e, adr og s er brukti kallet.

   void bestemKontonummer() {
        nummer++;
        kontonummer = nummer;
   } //Finner nytt ubrukt kontonummer

   void settInn(int innskudd) {
        saldo = saldo + innskudd;
   } //beregner ny saldo ved innskudd på konto

   boolean taUt(int uttak) {
        if (uttak > saldo) return false;
        saldo = saldo - uttak;
        return true;
   } /* sjekker om du overtrekker kontoen ved å ta ut beløpet uttak,
        hvis ok beregner ny saldo */
```

```
boolean overførTilAnnenKonto(Konto tilKonto, int beløp)
        if (taUt(beløp)) {
                tilKonto.settInn(beløp);
                return true;
        } else return false;
} /* sjekker om det er nok penger på kontoen, hvis ja
     overfører beløpet og beregner ny saldo */

int giSaldo() {
        return saldo;
   } //returnerer verdien saldo
} // end class Konto

class Bank {
  public static void  main ( String[] args)  {
     Bank b = new Bank();

     Konto a = new Konto();
     a.settInn(500);
     Konto k = new Konto("Anja", "Ifi, UiO", 1000);
     System.out.print("a.saldo: " + a.giSaldo());
     System.out.println(", k.saldo: " + k.giSaldo());
     a.overførTilAnnenKonto(k, 400);
     System.out.print("a.saldo: " + a.giSaldo());
     System.out.println(", k.saldo: " + k.giSaldo());
   }
}
```



Minner om viktige prinsipper

- En metode skal bare bruke objektvariable deklarert i egen klasse.
  - selv om man lett kan få tak i variable fra andre klasser, men bruk av variable fra andre klasser vil føre til rotete programmering hvor det senere blir vanskelig å endre på programmet. For eksempel slette en variabel (som viser seg å være brukt av en annen klasse).
- Hvis en metode viser seg vanskelig eventuelt lang å programmere del den i mindre metoder.
  - små enkle metoder som hovedsakelig kun gjør en oppgave (høyden noen få som klart hører sammen) gjør programmet lett lesbart for andre i ettertid.

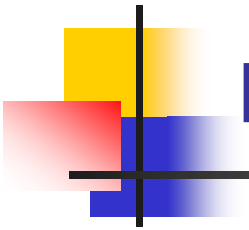
Lese fra fil

Vi må først importere pakken easyIO

Vi åpner filen for lesing

```
import easyIO.*;

class LesFraFil {
   public static void main (String [] args) {
      In fil = new In("filnavn");

      String s = fil.inLine();
      System.out.println("Første linje var: " + s);
   }
}
```

Her leses hele første linje av filen

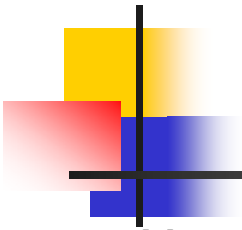


Lesemetoder

Tabellen under viser resultatet av å kalle på `inInt()`, `inChar()`, `inWord()` og `inLine()` etter første, andre og tredje kall.

```
15 ______
_____535___
3__522__
```

Kall nr	`inInt()` eller `inDouble()`	`inChar()`	`inWord()`	`inLine()`
1	15	'1'	"15"	"15 ______"
2	535	'5'	"535"	"_____535___"
3	3	' '	"3"	"3__522__"

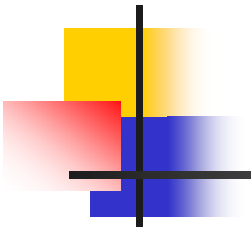


Les bit for bit

- Metoder:
  - inInt() for å lese et heltall
  - inDouble() for å lese et flyttall
  - inWord() for å lese et ord
  - inWord("\n") for å lese første ikketomme linje
  - lastItem() for å sjekke om slutten av filen er nådd

- Eksempel: fil hvor hver rad inneholder et heltall og en tekst:

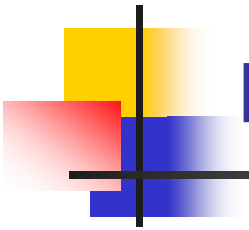
```
In fil = new In("fil.txt");

while (!fil.lastItem()) {
  int k = fil.inInt();
  String s = fil.inWord("\n");
  System.out.println(k + " " + s);
}
```



Eksempel

Programmet leser en fil med desimaltall, hvor tallene er atskilt med blanke tegn og/eller linjeskift. Vi finner slutten av filen med lastItem().

```
import easyIO.*;

class LesFraFil2 {
   public static void main (String [] args) {
      double[] x = new double[100]; // antar max 100 tall på fil
      In innfil = new In("fil.txt");
      int ant = 0;

      while (!innfil.lastItem()) {
         x[ant] = innfil.inDouble();
         ant = ant + 1;
      }
         // Nå ligger det verdier i x[0], x[1], ...., x[ant-1]
   }
}
```



Lese linje for linje

- Metoder:
  - inLine() for å lese en linje (ikke tomme)
  - readLine() for å lese en linje (også tomme)
  - endOfFile() for å sjekke om slutten av filen er nådd

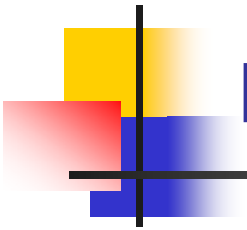
- Eksempel:

```
In fil = new In("fil.txt");

while (!fil.endOfFile()) {
  String s = fil.inLine();
  System.out.println(s);
}
```

Eksempel

Program som leser en tekstfil linje for linje:

```
import easyIO.*;

class LesFraFil3 {
   public static void main (String [] args) {
      String [] s = new String[100]; // antar max 100 linjer på innfilen
      In innfil = new In("fil.txt");
      int ant = 0;

      while (!innfil.endOfFile()) {
         s[ant] = innfil.readLine();
         ant = ant + 1;
      }

      // Nå ligger det tekststrenger i s[0], s[1], ...., s[ant-1]
   }
}
```



Lese tegn for tegn

- Metoder:
  - inChar() for å lese et tegn (også blanke og linjeskift)
  - endOfFile() for å sjekke om slutten av filen er nådd
- Eksempel:

```
In fil = new In("fil.txt");

while (!fil.endOfFile()) {
  char c = fil.inChar();
  System.out.println(c);
}
```

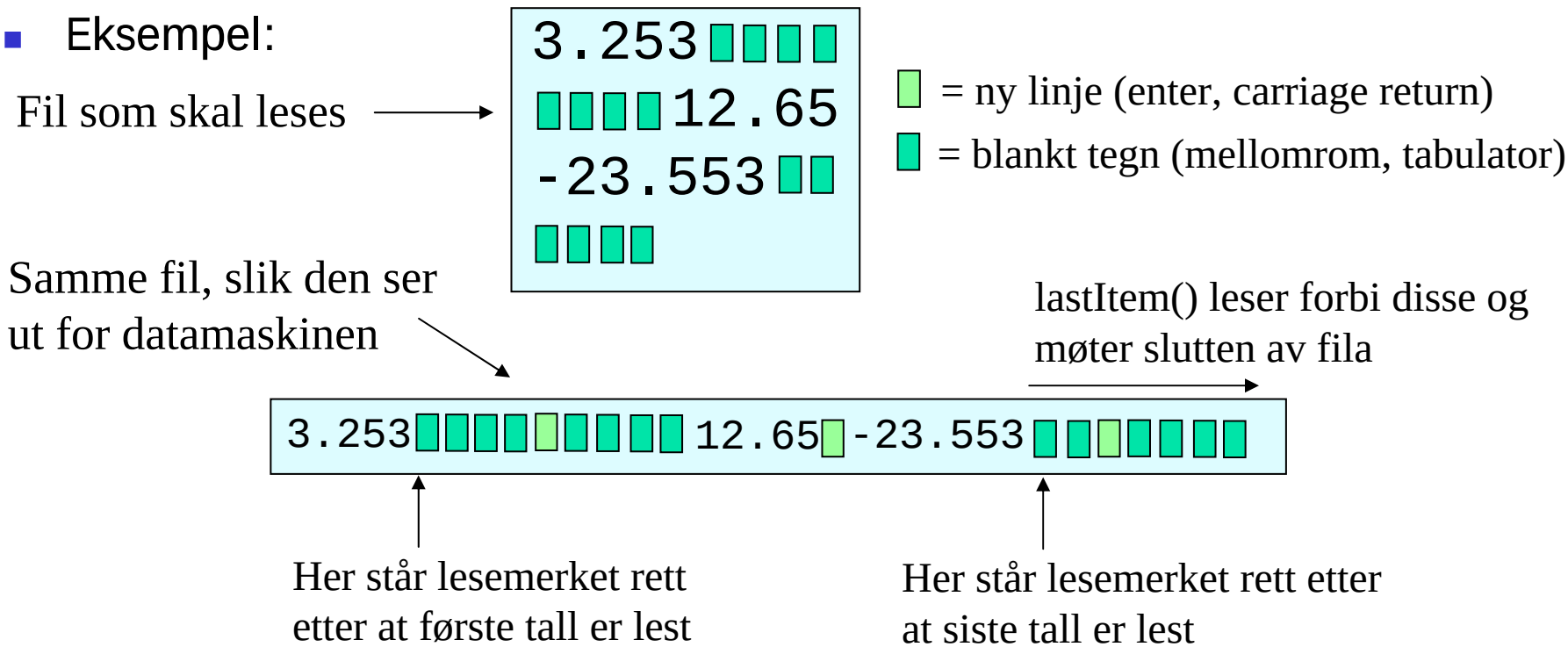
Eksempel

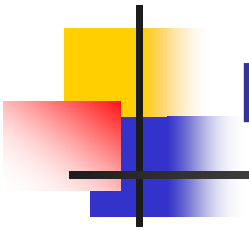
Program som leser en tekstfil tegn for tegn og skriver ut på skjerm, sammen med antall tegn i filen:

```
import easyIO.*;

class LesFraFil4 {
   public static void main(String[] args) {
      In innfil = new In("fil.txt");
      int antall = 0;

      while (!innfil.endOfFile()) {
         System.out.print(innfil.inChar());
         antall++;
      }

      System.out.println("Antall tegn: " + antall);
   }
}
```

lastItem og endOfFile

- `endOfFile()` sjekker om siste tegn på fila er lest
- `lastItem()` søker seg fram til første ikke-blanke tegn og returnerer **true** hvis slutten av fila nås og **false** ellers.
- Eksempel:





Når filens lengde er kjent

- Når et program skal lese en fil, må programmet ha en mulighet til å avgjøre når slutten av filen er nådd - ellers kan det oppstå en feilsituasjon.
- Metodene lastItem() og endOfFile() kan benyttes til dette.
- Noen ganger er filens lengde kjent på forhånd:
  - lengden er kjent før programmet kjøres
  - lengden ligger lagret i begynnelsen av filen

  Da kan vi i stedet benytte en for-løkke.

Eksempel: fil med kjent lengde

Program som leser en fil med 10 desimaltall, hvor tallene er atskilt med blanke tegn og/eller linjeskift:

```
import easyIO.*;

class LesFraFil5 {
   public static void main (String [] args) {
      double [] x = new double[10];
      In innfil = new In("fil.txt");

      for (int i=0; i<10; i++) {
         x[i] = innfil.inDouble();
      }

      // Nå kan vi evt. gjøre noe med verdiene i arrayen x
   }
}
```

Nok at tallene er atskilt

Programmet på forrige foil ville gitt akkurat samme resultat for alle disse filene:

```
15.2
6.23
3.522
3.6
8.893
-3.533
65.23
22.01
45.02
7.2
```

```
15.2    6.23
3.522   3.6
8.893   -3.533
65.23   22.01
45.02   7.2
```

```
15.2    6.23    3.522   3.6
8.893   -3.533
65.23   22.01   45.02

        7.2
```

Eksempel: fil med lengde-info

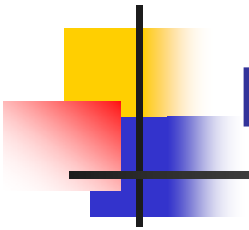
Program som leser en fil med desimaltall, hvor tallene er atskilt med blanke tegn og/eller linjeskift. Antall tall som skal leses ligger øverst i filen.

```
import easyIO.*;

class LesFraFil6 {
   public static void main (String [] args) {
      double[] x;     // bestemmer ikke lengden ennå
      In innfil = new In("fil.txt");

      int lengde = innfil.inInt();  // nå vet vi lengden
      x = new double[lengde];

      for (int i=0; i<lengde; i++) {
         x[i] = innfil.inDouble();
      }
      // Nå kan vi evt. gjøre noe med verdiene i arrayen x
   }
}
```

Eksempel: fil med sluttmerke

Program som leser en fil med desimaltall, hvor tallene er atskilt med blanke tegn og/eller linjeskift. Slutten av filen er markert med tallet -999.

```
import easyIO.*;

class LesFraFil7 {
   public static void main (String [] args) {
      double[] x = new double[100]; // antar max 100 tall på fil
      In innfil = new In("fil.txt");
      double siste = 0;
      int ant = 0;

      while (siste != -999) {
         siste = innfil.inDouble();
         if (siste != -999) {
            x[ant] = siste;
            ant = ant + 1;
         }
      }
      // Nå ligger det verdier i x[0], x[1], ...., x[ant-1]
   }
}
```

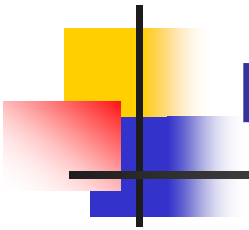


Lese en fil med mer komplisert format

- Anta at vi skal lese en fil med følgende format:
  - Først er det en linje med 3 overskrifter (separert av blanke tegn)
  - Deretter kommer det en eller flere linjer, som hver består av et heltall, et desimaltall og en tekststreng (separert av blanke tegn)
- Eksempel:

Antall	Pris	Varenavn
35	23.50	Oppvaskkost
53	33.00	Kaffe
97	27.50	Pizza
...	...	...
...	...	...

- Dataene som leses skal programmet ta vare på for senere formål.



Framgangsmåte

- Den første linja er spesiell, og vi tenker oss her at den ikke er så interessant - vi ønsker bare å få lest forbi den. Det kan vi gjøre med inLine().
- De andre linjene har samme format, så vi kan lage en løkke hvor hvert gjennomløp av løkken leser de tre delene på en linje. Vi bruker da henholdsvis inInt(), inDouble() og inWord().
- For å vite når filen er slutt, kan vi enten bruke endOfFile() eller lastItem(). Siden vi leser filen bitvis, er det mest naturlig å bruke lastItem(). Da får vi heller ikke problemer dersom det skulle ligge noen blanke helt på slutten av filen.
- Vi hopper over detaljene. PRØV SELV.

Noen nyttige hjelpemidler (ikke pensum)

- Sjekke om det finnes en fil med et bestemt navn:

```
if (new File("filnavn").exists()) {
    System.out.println("Filen finnes");
}
```

- Slette en fil:

```
if (new File("filnavn").delete()) {
    System.out.println("Filen ble slettet");
}
```

- Avgjøre hvilket filområde programmet ble startet fra:

```
String curDir = System.getProperty("user.dir");
```

- Lage liste over alle filer og kataloger på et filområde:

```
String[] allefiler = new File("filområdenavn").list();
```

Merk: klassen File ligger i pakken java.io som derfor må importeres først.

Skrive til fil

Vi må først importere pakken easyIO

Vi åpner filen for skriving

```
import easyIO.*;

class SkrivTilFil {
   public static void main (String [] args) {
      Out fil = new Out("filnavn");
      fil.outln("Dette er første linje");
      fil.close();
   }
}
```

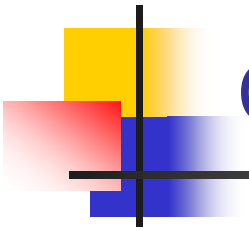
Her skrives en linje med tekst til filen

Vi må huske å lukke filen til slutt

Hvilke skrivemetoder finnes?

Datatype	Eksempel	Beskrivelse
int	`fil.out(x);` `fil.out(x, 6);`	Skriv x Skriv x høyrejustert på 6 plasser
double	`fil.out(x, 2);` `fil.out(x, 2, 6);`	Skriv x med 2 desimaler Skriv x med 2 desimaler på 6 plasser
char	`fil.out(c);`	Skriv c
String	`fil.out(s);` `fil.out(s, 6);`	Skriv s Skriv s på 6 plasser (venstrejustert)
	`fil.outln();`	Skriv linjeskift
	`fil.close();`	Lukk filen

Merk: dersom antall plasser spesifiseres og det ikke er plass til det som skal skrives ut, vil det som skrives ut avsluttes med tre punktumer: **...**



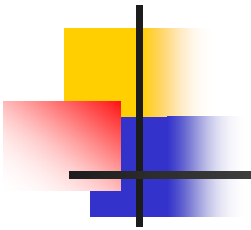
Oppgave

- Lag et program som:
  - leser en tekstfil
  - erstatter alle forekomster av en bestemt tekststreng s med en ny tekststreng t
  - skriver resultatet til en ny tekstfil

```
import easyIO.*;

class ByttOrdMain {
  public static void main (String [] args) {
    ByttOrd b = new ByttOrd();
    b.erstattOrd(args[0], args[1], args[2], args[3]);
  }
}

class ByttOrd {
  void erstattOrd(String innfilNavn, String utfilNavn,
       String ord, String nyttord) {
     In innfil = new In(innfilNavn);
     Out utfil = new Out(utfilNavn);

     while (!innfil.endOfFile()) {
        String s = innfil.inLine();
        String t = "";
        int n = ord.length();
        int k = s.indexOf(ord);
        while (k >= 0) {
           t = t + s.substring(0, k) + nyttord;
           s = s.substring(k+n);
           k = s.indexOf(ord);
        }
        t = t + s;
        utfil.outln(t);
     }
     utfil.close();
  }
}
```

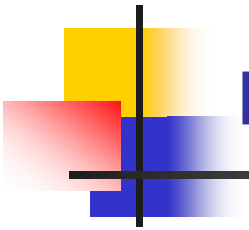
Noen ord i forkant av oblig 3

- Mange vil synes oblig 3 er:
  - mye å gjøre
  - ny måte å tenke på
  - stort sprang fra de små eksemplene på forelesningene
- De obligatoriske oppgavene er ikke i første rekke ment å teste kunnskapsnivået deres, men å videreutvikle det og forberede dere for nytt stoff (og naturligvis for eksamen).
- Oppgavene har m.a.o. et helt annet siktemål enn f.eks. en deleksamen.
- Derfor: det er høyst normalt om du synes oppgaven er mye jobb. Til gjengjeld vil de fleste av dere lære mye av obligen.
- Vi skal nå se på noen generelle råd vedrørende løsning av en større oppgaver.



Råd 1: Skriv programmer "ovenfra og ned"

- Bestem først hvilke klasser som skal være med (og deres rolle).
- Fyll inn de mest sentrale variablene (de som utgjør datastrukturen), og skriv eventuelle nye klasser som trengs i datastrukturen
- Skriv metodene på toppnivå (dvs. de som styrer den overordnede programflyten, f.eks. en kommandoløkke). Kall på metoder ved behov, selv om disse ennå ikke er skrevet.
- Skriv metodene du kaller på ovenfor, og fortsett til programmet er ferdig.



Råd 2: skriv metoder "utenfra og inn"

Når du skal skrive en metode, bestem først av alt hva som er input og output til metoden:

- Input:

  Eventuelle parametere til metoden

  Kan også være klassevariable/objektvariable.

- Output:

  Eventuell returverdi fra metoden

  Kan også være modifikasjoner av klassevariable/objektvariable

Råd 3: Deleger oppgaver

- Et viktig kjennetegn ved god programmering er at man delegerer oppgaver når det er naturlig – dvs. kaller på metoder for å utføre deloppgaver.
- Dermed blir hver enkelt del av programmet oversiktlig, og faren for feil minimeres. Det blir også lettere å finne feil senere.
- Eksempel:
  - Hvert case i en kommandoløkke kaller på en metode som utfører den ønskede kommandoen, i stedet for at alt gjøres inni selve kommandoløkken.
- NB: ikke overdriv delegering. Det er f.eks. ofte ikke naturlig at hvert eneste objekt har metoder for å lese fra terminal - det kan i mange tilfeller være bedre å gjøre slike ting sentralt (og heller kalle på metoder i objektene for å oppdatere deres variable).

Råd 4: formater alltid programkoden underveis Det øker lesbarheten

- Dårlig:

```
  class Eksempel {
public static void main (String [] args) {
     int x = 0;
  for (int i=0; i<10; i++) {
                    x = x + 1;
                          } if (x < 0) {
             System.out.println("Det var rart");
  }}}
```

- Bra:

```
class Eksempel {
   public static void main (String [] args) {
     int x = 0;

     for (int i=0; i<10; i++) {
        x = x + 1;
     }

     if (x < 0) {
        System.out.println("Det var rart");
     }
   }
}
```

Råd 5: ikke endre på forutsetningene i oppgaven

- Selv om mange oppgaver tar utgangspunkt i problemer fra virkeligheten, er de ikke nødvendigvis laget for å være særlig realistiske.

- Ikke endre på forutsetningene i slike tilfeller - uansett hvor fristende det måtte være. Løs oppgaven slik den er formulert.

- Hvis du mener oppgaven gir rom for flere tolkninger på et punkt: gjør dine egne (rimelige) forutsetninger - og skriv i oppgaven hva disse er.