INF1000 – Uke 11

TIPS Oblig 4
Strukturering av programmer
Noen flere eksempler

Oblig4: Komme igang

1. Les hele oppgaven nøye
2. Tenk på hvilke klasser du trenger
3. Lag et enkelt klassediagram og et skall til programmet
4. Så kan vi ta oppgavene en etter en

Noen generelle tips

- Lag først klassene og fyll det med de viktigste variablene og metodene
- Start på det høyeste nivået og skriv programflyten på det høyeste nivået
- Skriv programmet ovenfra og ned
- Lag tomme metoder som du kan fylle ut siden
- Tenk på hva som er input og output for metoder
- Deleger oppgaver til klasser/objekter
- Hold koden ryddig, formatert og kompilerbar
- Det er alltid lov å gå tilbake å endre noe. Vi kan bruke mange runder før vi er fornøyd med et program

Filer

- Vi skal lese data fra to filer og analysere dem (Stasjoner-1.txt, Vaerdata-1.txt)
- Stasjoner

Nummer | Navn | MoH | Kommune | Fylke

```
4780 GARDERMOEN 202 ULLENSAKER AKERSHUS
```

- Målinger

Stasjonsnr | Dag | Mnd | Max vind | Nedbør | Min tmp | Max tmp

```
4780  15  01  8.7  0.0  1.4  5.0
```

Finne klassene

- Meteorologisk institutt har en rekke værstasjoner rundt om i Norge. For hver slik værstasjon får vi oppgitt et entydig stasjonsnummer, stasjonens navn, stasjonens høyde over havet, kommunen og fylket hvor stasjonen ligger.

- Ved disse værstasjonene måles hver dag en rekke data, og vi skal her konsentrere oss om følgende fire data per dag: Maks vindhastighet, mmNedbør, Min temp og Max temp. Data for én måned for en værstasjon samles og rapporteres ofte som en enhet.

Finne klassene

- Meteorologisk institutt har en rekke værstasjoner rundt om i Norge. For hver slik værstasjon får vi oppgitt et entydig stasjonsnummer, stasjonens navn, stasjonens høyde over havet, kommunen og fylket hvor stasjonen ligger.

- Ved disse værstasjonene måles hver dag en rekke data, og vi skal her konsentrere oss om følgende fire data per dag: Maks vindhastighet, mmNedbør, Min temp og Max temp. Data for én måned for en værstasjon samles og rapporteres ofte som en enhet.

Se på substantivene

- Hvilke passer som klasser
  – Det som ***er*** noe

- Hvilke er variable i klassene
  – Egenskaper ved klassene

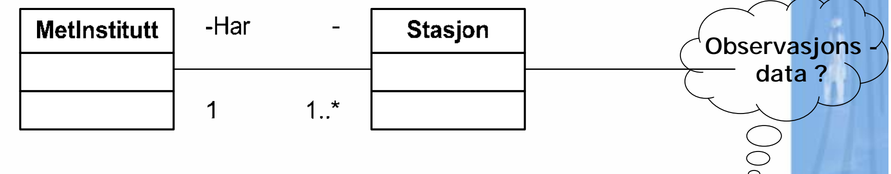
Og:

- Hva passer som metoder
  – Det som ***gjøres*** med eller av klassene

- Lag en skisse

Oppgave 1

- Hele denne oppgaven skal løses med klasser og objekter. Ut fra opplysningene ovenfor og ved å lese gjennom de spørsmål systemet skal kunne gi svaret på i oppgavene 2-6, så skal du tegne et klassediagram i UML over systemet ditt. Gi navn på de forhold du vil ha i systemet ditt. Skriv også antall på begge sidene av forholdene. Lever gjerne UML-diagrammet som en håndtegning.



Oppgave 2

- Du skal nå skrive de delene av programmet som leser de to filene “**Stasjoner-1.txt**” og ”**Vaerdata-1.txt**”. Det oppretter objekter av de klassene du har definert etter hvert som du leser filenes. Alle klassene (med unntak den som inneholder ”**main**”) skal ha en konstruktør du har skrevet selv og som initierer objektet med de data du leser inn om vedkommende objekt. Navnene på disse filene skal være parametere når du starter programmet (dvs. filnavnene er i **args[0]** og **args[1]** som parametere til **main**, se hint).

Start med ”skallet”

```
/**
* Omsluttende klasse for problemet, tar opp parametre
* fra kommandolinja og starter kommandoløkka. Feilmelding
* hvis ikke minst to parametere.
*****************************************************/

class Oblig4 {

  /**
  * Sjekker parametere, starter opp ordreløkka etter at
  * filene er lest via konstruktoren til 'MetInst'
  *****************************************************/

  public static void main(String[] args) {
    if (args.length >= 2) {

      MetInst m = new MetInst(args[0],args[1]);
      m.ordreløkke();

    } else
      System.out.println("Bruk: >java Oblig4 <fil med ”+
          ” Stasjonsdata> < fil med Observasjonsdata>");
  }
} // end class Oblig 4
```


Filnavnene

- **f) De to filnavnene som parametere til ”main”**
  For å få til det så nevner du dem bare på samme linje som du når du starter programmet ditt. Anta at klassen din som inneholder ”main” i en klasse som heter ’Meteorologi’. Du starter da programmet som følger:

  ```
  >java Meteorologi Stasjoner-1.txt Vaerdata-1.txt
  ```

  Når du senere vil prøve de større filene med flere måledata, sier du bare:

  ```
  >java Meteorologi Stasjoner-2.txt Vaerdata-2.txt
  ```


Stasjoner-2.txt vs Stasjoner-1.txt

- “**Stasjoner-2.txt**” og ”**Vaerdata-2.txt**” inneholder data for langt flere stasjoner, og som derfor kan gi bedre og mer interessante svar enn de få værstasjonene det er i de tilsvarende filene som slutter med ”**-1.txt**”

Oppgave 3

- Du skal nå utstyre programmet med en metode som skriver ut en meny for brukeren som gir brukeren følgende valg, som du skal programmere i oppgavene 4-6 (Menyen skal se ut nøyaktig som nedenfor):

```
1 – Finn antall uværsdager
2 – Finn kommune med lavest temperatur
3 – Finn stasjon med mest nedbør
4 – avslutt
```

- Du bør nå ha et overordnet program og noen klasser og kan lage tre metoder som kalles fra hovedprogrammet

Lag altså metodene

```
finnAntallUværsDager(String stasjonsnavn, int måned)

temperaturMinKommune ()

stasjonMedMestNedbør(int måned)
```

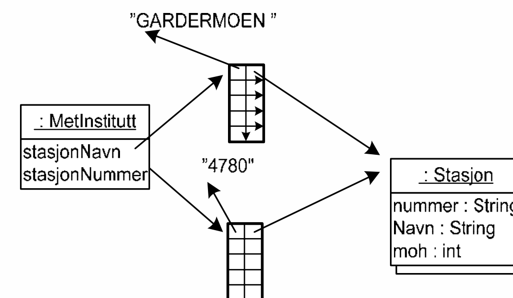
- Oppgavene kan/skal delegeres til objektene du har laget ved innlesing.

Måned

- b) Lag tre hjelpemetoder i den klassen som representerer måledata for en måned for en stasjon.
  - se på hva du kan trenge av opplysninger fra en målestasjon for å løse oppgavene

To HashMap-er

- **c)** Av og til ønsker du å søke etter stasjonene ut fra deres stasjonsnummer og av og til ut fra deres navn. Det kan derfor være fornuftig å lage *to* HashMap'er – en hvor nøkkelen er navnet og en hvor nøkkelen er stasjonsnummeret. Det er det *samme* objektet du legger (peker til) i de to HashMap'ene, men nøklene vil være ulike



UNIVERSITETET I OSLO

Kan ikke bruke int som nøkkel i HashMap

- **d)** Hvis du vil lage en nøkkel til en HashMap av noe som egentlig er et heltall, si: **int num;,** så lager du enkelt en nøkkel slik

```
String s = "" + num;
```

(dette lager en String med tallverdien i num ved å legge til den tomme tekststrengen)

Manglende data

UNIVERSITETET I OSLO

- **e)** En måte å behandle manglende data, er at alle **-99**'ene leses inn som om dette er virkelige data. De metoder som regner ut gjennomsnitt ol, må da hver gang sjekke om det er verdier **!= -99** de finner og ikke ta med slike dataverdier i beregningene. Slike metoder kan da også selv returnere **-99** dersom det ikke finnes **noen** data å gjøre beregningene på (enten f.eks. for at brukeren spesifiserer en måned vi ikke har data for - f.eks. måned: 8 – august, eller at **alle** data for vedkommende måned mangler). Husk at du samtidig må telle opp hvor mange dager du har virkelige observasjoner for, slik at gjennomsnittet blir riktig.

Altså:

- Mangler det måling for en dag kan den ikke være med i gjennomsnittet for måneden (antallet blir også mindre!)
- Mangler alle data/dagne for en måned kan vi ikke finne gjennomsnitt for den
- Hvis vi ikke har gjennomsnitt for noen måneder for en stasjon kan vi ikke finne gjennomsnittet for stasjonen.

UNIVERSITETET I OSLO

ÆØÅ-problemer

- **g)** Hvis du leser inn data og så skriver dem ut på en PC via Windows, så vær klar over at æ, ø, å, Æ, Ø, Å blir skrevet ut som andre tegn på skjermen, og hvis man f.eks taster inn æ, ø, å på en Windowspc, så vil de ikke bli oppfattet som de æ-ene, ø-ene og å-ene du har lest inn fra fila. *Bruk derfor stasjonsnumrene* til å identifisere stasjonene, både i brukerdialogen og innad i programmet når du leser Vaerdata-filene. (eksempler på "mishandling" av ÆØÅ: NY-☐LESUND (99910), MIDTL☐GER (46510), VARD☐ (98550)). Skriv derfor både ut navn og nummer på stasjonene. Hvis du har dette problemet kan du be brukeren taste inn et stasjonsnummer når stasjonen velges, men du skal likevel kalle f.eks. finnAntallUværsDager(..) med den tilhørende teksten som er stasjonsnavnet.
- Løsningen er altså (når brukeren skal velge stasjon):
  - List opp alle stasjoner med nummer og navn
  - Be brukeren skrive inn stasjonsnummer

Javadoc

UNIVERSITETET I OSLO

- **i)** Når du skal lage javadoc av systemet ditt (og vi antar at de eneste ".java"-filene på det filområdet er de som hører til Oblig4), så gir du kommandoen:

```
>javadoc -package *.java
```

- Grunnen til å å ha med parameteren **–package** er at den gjør at alle variable, klasser og metoder som det ikke står noen modifikator foran samt de det står public foran blir med i dokumentasjonen. Har du ikke med **–package,** vil bare de som er nevn som **public** blir med i dokumentasjonen.
- Husk også at du kan lage korte javadoc-kommentarer på en linje som f.eks:

```
/** Skriver ut bruker-menyvalg */
void meny (Out ut) {
...
```

- Javadoc-kommentarer for en objektvariabel plasseres like over deklarasjonen, som f.eks:

```
/** Stasjonenes høyde over havet (meter) */
int moh;
```

EKSTRA

- Tåler ditt program å koble:

  `Stasjoner-2.txt` med `Vaerdata-1.txt`

  manglende data for mange stasjoner

  `Stasjoner-1.txt` med `Vaerdata-2.txt`

  data for manglende stasjoner
  (ignorere disse data)

Til slutt

- IKKE JUKS!
- JOLY finner like besvarelser
- Spør om hjelp
- Oppgi kilder
- Lever det dere har når fristen går ut

Å lage en fornuftig datamodell

- Med objekter kan vi ofte organisere våre data bedre.
- Eksempel:

```
String[] navn = new String[100];
String[]  fnr = new String[100];
int[] tlfnr = new int[100];
```

Informasjonen knyttet til en bestemt person er splittet opp i tre arrayer.

```
class Person {
  String navn;
  String fnr;
  int tlfnr;
}
Person [] personreg = new Person[100];
```

Informasjonen knyttet til en bestemt person er samlet i et objekt.

Bedre organisering – særlig når det er mye data å holde orden på.

Å lage en fornuftig datamodell (II)

- Med objekter kan vi samle data og operasjoner på dem.

```
... data om studenter...
... data om ansatte ...
... data om kurs ...
... student-metoder ...
... ansatt-metoder ...
... kurs-metoder ...
```

Her ligger alle data og alle metoder samme sted

```
class Student {
  ... data om studenter ...
  ... student-metoder ...
}
class Ansatt {
  ... data om ansatte ...
  ... ansatt-metoder ...
}
class Kurs {
  ... data om kurs ...
  ... kurs-metoder ...
}
```

Metoder og data som hører sammen er samlet. Lett å se hvilke metoder som jobber på hvilke data (modularisering av koden).

Lett å kopiere alt som har med personer å gjøre (data + metoder) til andre programmer (gjenbruk).

Å lage en fornuftig datamodell (III)

UNIVERSITETET I OSLO

- Eksempel: i Oblig 3 skulle du holde orden på
  - en rekke studenter → `class Student`
  - en rekke hybler → `class Hybel`
  - et hybelhus (potensielt flere) → `class Hybelhus`
- En objektorientert løsning (med klassene over) sørger for at
  - variabler og metoder som logisk hører sammen ligger også samlet i programkoden
  - variabler og metoder som ikke har noe med hverandre å gjøre holdes godt atskilt i programkoden

Valg av datamodell: eksempel

UNIVERSITETET I OSLO

- Eksempel:
- Du har gitt en fil med opplysninger om hvor mange registrerte tilfeller det var av tre ulike sykdommer i Norge hvert av årene 1950...2000:

```
        INFLUENSA    KYSSESYKE    MENINGITT
1950    ...          ...          ...
1951    ...          ...          ...
1952    ...          ...          ...
.
.
2000    ...          ...          ...
```

- Hvordan er det naturlig å modellere dette?

Noen muligheter

UNIVERSITETET I OSLO

Forslag 1: Gruppere tellinger relatert til samme sykdom

```
class Sykdom {
    String sykdomsNavn;
    int[] antallTilfeller = new int[51];
}
```

Forslag 2: Gruppere tellinger foretatt samtidig

```
class Aarsdata {
    int antInfluensa;
    int antKyssesyke;int antMeningitt;
}
```

Forslag 3: Ingen gruppering – tre arrayer

```
int[] influensatilfeller = new int[51];
int[] kyssesyketilfeller = new int[51];
int[] meningittilfeller = new int[51];
```

Forslag 4: Ingen gruppering – en 2D-array

```
int[][] sykdomstilfeller = new int[3][51];
```

Beste datastruktur avhenger i stor grad av hva du skal bruke dataene til!

UNIVERSITETET I OSLO

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 (f.eks. endring av innholdet i en HashMap).

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 koden underveis

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: Det er alltid lov å gå tilbake å endre på noe!

- Programmer blir til ved at vi jobber litt her og der.
- Vi finner ofte ut at vi trenger flere klasser, eller at en klasse bare er “i veien” og fjerner den
- Det er ingen skam å snu. Det endelige programmet kan ha andre klasser og metoder enn vi startet med
- Pass likevel på å holde programmet kompilerbart og å heller ha ”tomme skall” av alle metoder som kalles enn å ikke ha de der.

Eksempel: Flyreservasjon

UNIVERSITETET I OSLO

Klasser

Egenskaper

Prosedyerer

- Vi skal lage et system for et flyselskap
- Systemet skal holde orden på alle selskapets flyvninger og reserverte seter på flyene
- En flyvning har en kode, et avreisested og en destinasjon, i tillegg til et fly, som har et identifikasjonsnummer
- Et fly består av seterader, med seter
- Oppgavene systemet skal løse er å lese inn en beskrivelse av alle flyene, med antall seter, klasser på de forskjellig seteradene, osv
- Så skal man kunne reservere seter, avbestille og skrive ut en oversikt over flyets seter, med klasse og om det er ledig eller ikke

UNIVERSITETET I OSLO

Eksempel: Flyreservasjon

- class Systemet
  - Inneholder kun main-metoden. Lager objekt av klassen under og kaller på ordreløkke-metode.
- class Flyreservasjon
  - Inneholder ordreløkke og andre metoder + HashMap-tabeller for å holde orden på flyvningene.
- class Fly
  - Hvert objekt inneholder info om en flyet + alle seteradene og setene i flyet
- class Seterad
  - Setene i raden
- class Sete
  - Klasse og om det er opptatt eller ikke

UNIVERSITETET I OSLO

Systemet

```
import easyIO.*;
import java.util.*;

class Systemet {
  public static void main (String[] args) {
    String s1 = ”Fly.txt";
    String s2 = ”Bestillinger.txt";
    Flyreservasjon f = new Flyreservasjon(s1,
                                         s2);
    f.ordreløkke();
  }
}
```


UNIVERSITETET I OSLO

Flyreservasjon

```
class Flyreservasjon {
  HashMap fly = new HashMap();
  HashMap flyvninger = new HashMap();

  Flyreservasjon(String s1, String s2) {
    lesFly(s1);
    lesReservasjoner(s2);
  }
  void lesFly(String fnavn) {...}
  void lesReservasjoner(String fnavn) {...}
  void ordreløkke() {...}

  ...
}
```

Datastruktur

Konstruktør som gjør initialisering (her: lese data fra fil)

Metoder for å lese fra fil og for å lese inn kommando fra bruker

Her kommer det metoder som skal kalles fra ordreløkken

Flyreservasjon

- Programmere ordreløkken
  - For hver kommando som skal utføres, skal ordreløkken kalle på en passende metode i klassen Flyreservasjon.
  - For at programmet skal kompilere, sørg for å deklarere alle de metodene som du kaller på fra ordreløkke-metoden. Du kan vente med å fylle inn innholdet i disse metodene, dvs bare fyll inn en utskriftssetning i hver av metodene.
  - Eksempel: hvis ordreløkken kaller på metoden visFlyvning(), så deklarerer du samtidig denne "dummy-metoden" i klassen Flyreservasjon:

```
void visFlyvning() {
 System.out.println("Metoden visFlyvning utført");
}
```


Skrive ut flyvning

- Programmer metodene som kalles fra ordreløkken
- Eksempel (i klassen Flyreservasjon):

```
void visFlyvning() {
  System.out.println("Flyvning: ");
  String flightKode = tast.inLine();

  Flyvning flight = <finn flyvingen ved oppslag i
              flyvninger>;

  flight.skrivUt();
}
```

Oppdraget delegeres videre til en metode i Flyvning-objektet som er aktuelt.

Flyvning

- Skriver ut litt informasjon om flyvningen og delegerer så ansvaret for utskrift av oppsettet i flyet til klasen fly.

```
class Flyvning {
  String flightkode;
  String avreisested;
  String destinasjon;
  Fly fly;
  void skrivUt() {
    System.out.println("Flight: " + flightkode);
    System.out.println("Fra: " + avreisested);
    System.out.println("Til: " + destinasjon);
    fly.skrivUt();
  }
}
```

Oppdraget delegeres videre til en metode i Fly-objektet som er aktuelt.

Fly

- Skriver ut informsjon on flyet og delegerer videre til seteradene, som igjen delegerer videre til setene.

```
class Fly {
  String flykode;
  Seterad[] seterader;
  int skrivUt() {
    System.out.println("Flykode: " + flykode);
    for(int i=0; i<seterader.length; i++){
      seterader[i].skrivUt();
    }
  }
}
```

Og Fly delergerer videre

```
class Flyreservasjon {
  void ordreløkke() {
    ...
    visFlyvning();
    ...
  }
  void visFlyvning() {
    ...
    flight.skrivUt();
    ...
  }
}
```

Vi har *ett* objekt av denne.

```
class Flyvning {
  skrivUt() {
    ...
    ...
    fly.skrivUt();
  }
}
```

Vi har *flere* objekter av disse.

```
class Fly{
  skrivUt() {...}
}
```


ELIZA – den elektroniske psykiater

- ELIZA: program skrevet av Joseph Weizenbaum i 1966 for studier av språklig kommunikasjon mellom mennesker og datamaskiner.
- Programmet imiterer en psykiater, slik som i dette eksemplet:

Person: *Men are all alike.*
ELIZA: In what way?
Person: *They're always bugging us about something or other.*
ELIZA: Can you think of a specific example?
Person: *Well, my boyfriend made me come here.*
ELIZA: Your boyfriend made you come here?
Person: *He says I'm depressed much of the time*
ELIZA: I am sorry to hear that you are depressed.
Person: *It's true. I am unhappy.*
ELIZA: Do you think coming here will help you not to be unhappy?
Person: *I need some help, that much seems certain.*
ELIZA: What would it mean to you if you got some help?
Person: *Perhaps I could learn to get along with my mother.*
ELIZA: Tell me more about your family

ALICE: En kunstig intelligens-basert prate-robot

- ALICE = Artificial Linguistic Internet Computer Entity
- http://alice.pandorabots.com

Human: Hi Alice
ALICE: Hello there.
Human: How are you doing today?
ALICE: Ah. I'm doing great, how about you?
Human: I'm ok, listen can I ask you something?
ALICE: I am listening to you. Go ahead. Ask me anything.
Human: How many are you talking to at this moment?
ALICE: Right now there are 100 clients on line.
Human: You must be very busy
ALICE:Thanks for telling me that I am busy.
Human: Do you ever go on vacation?
ALICE:Robots never take vacations.
Human: Do you ever feel tired?
ALICE:As a robot I lack any human emotions.
Human: I sometimes feel tired
ALICE:What is that feeling like? Better get some rest.

Human: What is 2 + 2?

ALICE: Four.

Human: What is the square root of 5?

ALICE: Do I look like a calculator? I think you already know the answer anyways.

Human: That was very rude

ALICE:I thought it was too.

Human: What is Java?

ALICE:Java is the latest in the C family of languages. Proponents of Java cite its universal "write once, run anywhere" promise. Actually Java borrow a lot of features from AI languages like Lisp.

Human: Do you like Java?

ALICE:Yes Java is a pretty cool language for AI.

En samtalepartner

- Vi skal nå lage et program som gjør det mulig å "snakke med maskinen". Hver gang maskinen gjenkjenner et ord i det vi skriver så gir den et bestemt (og tilnærmet fornuftig) svar. Mer konkret:
  - Programmet skal først lese filen "ord.txt" som inneholder en del søkeord med tilhørende svar og legge disse inn i en HashMap.
  - Programmet skal deretter gå i løkke, og i hvert gjennomløp av løkken skal programmet:
    - vente på og lese inn en linje fra terminal (bruker-input)
    - søke etter matchende ord i input og søkeord i HashMap'en
    - skrive ut tilhørende svar (eventuelt et standardsvar hvis ingen match)

Filen ord.txt

```
hei Hei du. Fortell hvorfor du er kommet til meg!
morn Morn du. Hva kan jeg hjelpe med?
Heisan God formiddag, og hva er ditt problem?
dager Det var ikke saa lenge.
uker Det var lenge.
maaneder Det var veldig lenge.
vondt Hvor lenge har du vaert slik?
hodet Hvilke symptomer har du kjent?
trist Foeler du deg deprimert?
jobber Jobber du veldig mye?
jobbe Jobber du veldig mye?
syk Hva med aa kontakte en lege?
frisk Det er viktig aa holde seg frisk.
```


Programskisse

```
import easyIO.*;
import java.util.*;
class Eliza {
  public static void main(String [] args) {

  }
}
class Samtale {
  HashMap hash = new HashMap();
  In tast = new In();
  void lesFraFil() {

  }
  void snakk() {

  }
}
```

```
class Eliza {
  public static void main (String [] args) {
    Samtale sam= new Samtale();
    sam.lesFraFil();
    sam.snakk();
  }
}
```



```
void lesFraFil() {
  In fil= new In("ord.txt");
  while (!fil.lastItem()) {
    String søkeord= fil.inWord();
    String svar= fil.inLine();
    hash.put(søkeord, svar);
  }
  fil.close();
  System.out.println("Antallordlest: " +
                     hash.size());
}
```



```
void snakk() {
  while (true) {
    System.out.print("> ");
      boolean funnetMatch= false;
      do {
        String ord= tast.inWord().toLowerCase();
        if (hash.containsKey(ord)) {
          String svar= (String) hash.get(ord);
          System.out.println(svar);
          funnetMatch= true;
        }
      } while (tast.hasNext() && !funnetMatch);

      if (!funnetMatch) {
        System.out.println("Interessant. Fortellmer.");
      }
      if (tast.hasNext()) {
        tast.readLine();  // Tømmer input-bufferet
      }
    }
  }
}
```


Du finner hele programmet Eliza.java på nettsidene til kurset sammen med fila ord.txt.

Det kan hende det bare virker med forrige versjon av easyIO. Den som ligger på kurssiden for høsten 2006.