INF1000: Forelesning 11

Mer om HashMap
Noen råd i forbindelse med oblig 4

16. april 2007

HashMap

- Brukes til å holde orden på en samling objekter
- Alternativ til arrayer
- Akkurat som for arrayer kan man:
  - legge inn nye objekter
  - finne tilbake til et objekt som er lagt inn
  - fjerne et objekt som er lagt inn
  - løpe gjennom alle objektene i tabellen
- Viktig forskjell mellom arrayer og HashMap:
  - I en array legger vi inn objekter i en bestemt posisjon, og vi må gå tilbake til denne posisjonen/indeksen når vi senere skal se på objektet. Indeksen er et heltall mellom 0 og length-1.
  - I en HashMap oppgir vi en bestemt *nøkkel* (vanligvis en tekststreng) når vi legger vi inn et nytt objekt, og vi oppgir denne nøkkelen når vi senere skal se på objektet. Dvs. indeksen er en tekststreng.

Ulike versjoner i Java 1.4 (gammel) og Java 1.5

- Vi gjennomgår begge måtene, men anbefaler klart at 1.5-måten nyttes, da den hjelper deg mot visse feil (som ellers er lett å gjøre).
- 1.4 måten er viktig fordi mange gamle programmer inneholder slik kode.

Eksempel på bruk av HashMap (1.5)

Importer pakken java.util

Opprett en HashMap og forteller hvilke klasser nøkkelen og verdier har.

```
import java.util.*;

class BrukAvHashMap {
   public static void main (String[] args) {
      HashMap<String,Person> h = new HashMap <String,Person>();

      String fnr1 = "30128812344";
      Person per1 = new Person(fnr1, "Harald Olsen");
      h.put(fnr1, per1);

      String fnr2 = "14109522547";
      Person per2 = new Person(fnr2, "Lena Torsen");
      h.put(fnr2, per2);

      Person p = h.get("30128812344");
   }
}

class Person {
   String fnr;
   String navn;

   Person(String fnr, String navn) {
      this.fnr  = fnr;
      this.navn = navn;
}
}
```

Legg inn Person-objekt i HashMap'en

Legg inn Person-objekt i HashMap'en

Hent Person-objekt fra HashMap'en

Opprette en HashMap, Java1.4 (gammel) og Java 1.5

- I starten av programmet:
  ```
  import java.util.*;
  ```
  Dette importerer pakken java.util hvor bl.a. klassen HashMap ligger.
- I klassen eller metoden som skal bruke HashMap'en **Java 1.4**:
  ```
  HashMap h = new HashMap();
  ```
- I klassen eller metoden som skal bruke HashMap'en **Java 1.5** (best):

```
HashMap <String,Person> h = new HashMap <String,Person>();
```

I Java 1.5 forteller vi hvilke klasser nøkkel- og verdi-objektene kommer fra. Vi sier at vi da låser objektene til både nøkkelen og verdi-objektene til å være av disse typene.

NB: Hvis tabellen skal brukes av flere metoder i en klasse, deklareres variabelen ovenfor i starten av klassen (som en objektvariabel).

Hvis tabellen kun skal brukes av en enkelt metode, er det naturlig å deklarere variabelen ovenfor inni den aktuelle metoden.

Legge inn objekt i HashMap (samme i 1.4 og 1.5)

- Et hvilket som helst objekt i Java kan legges inn i en HashMap
- Når vi legger et objekt inn i HashMap'en, må vi samtidig oppgi en nøkkel, dvs en tekststreng som entydig identifiserer objektet.
- Vi trenger denne nøkkelen dersom vi senere skal finne eller fjerne objektet i HashMap'en.
- Eksempel:

```
String fnr = "30126512345";
Person p = new Person(fnr, "Kari Olsen");
h.put(fnr, p);
```

Her lager vi først et Person-objekt (med passende argumenter) og legger det deretter inn i tabellen med fødselsnummeret som nøkkel.

- Dersom vi legger inn flere objekter med samme nøkkel, er det bare det sist innlagte objektet som blir liggende i tabellen (de andre overskrives):

```
Person p1 = new Person(...);
Person p2 = new Person(...);
Person p3 = new Person(...);
String navn = "Jens";
h.put(navn, p1); // p1 legges inn
h.put(navn, p2); // p2 legges inn og p1 overskrives
h.put(navn, p3); // p3 legges inn og p2 overskrives
```

- Noen ganger må vi konstruere en nøkkel ut fra flere variable for å få entydighet:

```
String lengdegrad = "67.3";
String breddegrad = "53.3";
String posisjon = lengdegrad + ";" + breddegrad;
Fjelltopp fjell = new Fjelltopp(posisjon, "Bjørnefjell");
h.put(posisjon, fjell);
```


Hente objekt fra HashMap – Java 1.4 og 1.5

Java 1.4: For å hente et objekt med utgangspunkt i nøkkelen:

```
// 1.4: Vi vil finne en person ut fra fødselsnummeret:
Person p = (Person) h.get(fnr);
```

- Legg merke til at vi i 1.4 i starten må skrive i parentes navnet på klassen som objektet tilhører - i dette tilfellet klassen Person.
  - Årsaken er at i 1.4 HashMap'en ikke holder rede på hvilken klasse objektene som legges inn har - bare at det er objekter. Når objektene hentes ut må vi derfor "minne Java på" hvilken klasse objektet var av (dette er egentlig et møte med en avansert og svært nyttig mekanisme i objektorienterte språk som kalles *arv* og som blir tatt opp i vårens INF1010).

Java 1.5: For å hente et objekt med utgangspunkt i nøkkelen , nå trenger vi ikke si hvilken klasse objektet har (det har vi jo sagt i deklarasjonen av HashMapen h):

```
// 1.5: Vi vil finne en person ut fra fødselsnummeret:
Person p = h.get(fnr)
```

- **Merk**: å hente et objekt fra en HashMap slik som over medfører *ikke* at objektet fjernes fra HashMap'en (vi får bare en kopi av peker til objektet).

Fjerne objekt fra HashMap

- For å fjerne et objekt med gitt fødselsnummer som nøkkel:

```
h.remove(fnr);
```

- Dersom det ligger et objekt i HashMap'en med den gitte nøkkelen, blir objektet fjernet og setningen ovenfor returnerer med en peker til objektet som fjernes.
- Dersom det ikke ligger et objekt i HashMap'en med den gitte nøkkelen, returnerer setningen ovenfor verdien `null`.

Løp gjennom alle objekter i HashMap **Java 1.4**

- For å løpe gjennom alle objektene i en HashMap, lager vi en *oppramsing*:

```
Iterator it = h.values().iterator();
```

- Deretter kan vi se på hvert enkelt objekt i HashMap'en ved å gå i løkke:

```
while (it.hasNext()) {
     Person p = (Person) it.next();
     System.out.println("Navn: " + p.fåNavn());
}
```


Løp gjennom alle objekter i HashMap **Java 1.5**

- For å løpe gjennom alle objektene i en HashMap, lager vi en *oppramsing og låser samtidig det vi skal hente til en bestemt klasse*:

```
Iterator <Person> it = h.values().iterator();
```

- Deretter kan vi se på hvert enkelt objekt i HashMap'en ved å gå i løkke:

```
while (it.hasNext()) {
     Person p = it.next();
     System.out.println("Navn: " + p.fåNavn());
}
```

- Vi kan også i 1.5 nytte den nye for-løkka som automatisk lager en iterator

```
for (Person p: h.values()) {
     System.out.println("Navn: " + p.fåNavn());
}
```


To måter å løpe gjennom en HashMap – 1.5

- Løpe gjennom objektene (som på forrige foil):

```
Iterator <Person>it = h.values().iterator();

while (it.hasNext()) {
  Person p = it.next();
  <gjør noe med objektet>
}
```

- Løpe gjennom nøklene:

```
Iterator <String> it = h.keySet().iterator();

while (it.hasNext()) {
  String nøkkel =  it.next();
  <gjør noe med nøkkelen>
}
```

Metoder i HashMap

Metode	Eksempel	Beskrivelse
put	`h.put(nøkkel, objekt);`	Legg inn objekt med gitt nøkkel
get -1.4 get -1.5	`Person p = (Person) h.get(nøkkel);` `Person p = h.get(nøkkel);`	Finn objekt
remove	`h.remove(nøkkel);`	Fjern objekt

containsKey	`if (h.containsKey(nøkkel)) {`
	`  // gjør et eller annet`
	`}`
		Sjekk om nøkkel finnes i tabell

values	`Iterator it = h.values().iterator();`	Lag oppramsing av objektene
keySet	`Iterator it = h.keySet().iterator();`	Lag oppramsing av nøklene

Metoder i Iterator (oppramsing)

Metode	Eksempel	Beskrivelse

hasNext()	`while (it.hasNext()) {`
	`  < les neste og gjør noe>;`
	`}`
		returnerer true hvis flere objekter i oppramsingen

next() 1.5 next() 1.4	`Person p = it.next();` `Person p = (Person) it.next();`	Finn neste objekt

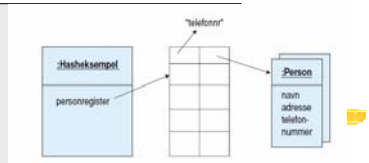
remove()	`Person p = it.next();`
	`if (p.navn.equals("Arne"))`
	`  it.remove();`
		Fjern siste objekt som ble returnet med next()


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

class Hasheksempel {
   public static void main(String[] argv) {
      In tastatur = new In();
      HashMap <String,Person> personregister =   new HashMap <String,Person>();
      System.out.print("Antall personer som registreres : ");
      int ant = tastatur.inInt();

      for (int i = 0; i < ant; i++) {
         System.out.println("Antall gjenværende personer " +(ant - i));
         Person p = new Person(tastatur);
         personregister.put(p.telefonnr, p);
      }
      // Skriv ut alle personobjektene
      System.out.println("Viser alle personer" +
      "(ukjent rekkefølge):");

      for (Person p: personregister.values()){
         p.skrivData();
      }
   }
}
```




```
class Person {
   String navn, adresse, telefonnr;

   Person (In tastatur) {
      System.out.print("Oppgi navn : ");
      navn = tastatur.inLine();
      System.out.print("Oppgi adresse : ");
      adresse = tastatur.inLine();
      System.out.print("Oppgi telefonnummer : ");
      telefonnr = tastatur.inLine();
   }
   void skrivData() {
      System.out.println("Navn : " + navn);
      System.out.println("Adresse : " + adresse);
      System.out.println("Telefonnummer : " + telefonnr);
   }}
```

Eksempel fra boka s.186

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 parametre til metoden

  Kan også være klassevariable/objektvariable (eksempel: de to HashMap'ene skal brukes i mange av metodene i klassen VaerAnalyse, selv om de ikke er parametre til metodene).

- Output:

  Eventuell returverdi fra metoden

  Kan også være modifikasjoner av klassevariable/objektvariable (f.eks. endring av innholdet i de to HashMap'ene).

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: ikke endre på forutsetningene

- 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.

Eksempel: oblig 4

`class Oblig4`

Inneholder kun main-metoden.
Lager objekt av klassen under og kaller på ordreløkke-metode.

`class Analyse`

Inneholder ordreløkke og andre metoder + HashMap-tabeller for å holde orden på værstasjonene.

`class Stasjon`

Hvert objekt inneholder info om en værstasjon + alle data fra stasjonen

`class Maaned`

Hvert objekt inneholder info om en måned med værdata fra en stasjon

`class Dag`

Hvert objekt inneholder info om dagsmålinger foretatt ved en stasjon

Oblig 4: trinn 1

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

class Oblig4 {
   public static void main (String[] args) {
      String s1 = "Stasjonsdata-1.txt";
      String s2 = "Meteorologidata-1.txt";
      Analyse a = new Analyse(s1, s2);
      a.ordreløkke();
   }
}
```


Oblig 4: trinn 2

```
class Analyse {
   HashMap hashNavnStasjon = new HashMap();
   HashMap hashNummStasjon = new HashMap();

   Analyse(String s1, String s2) {
      lesStasjonsdata(s1);
      lesMeteorologidata(s2);
   }

   void lesStasjonsdata(String fnavn) {...}
   void lesMeteorologidata(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

De to HashMap'ene

`HashMap hashNavnStasjon = new HashMap();`
Brukes for å holde orden på værstasjoner, med stasjonsnavn som nøkkel.

`HashMap hashNummStasjon = new HashMap();`
Brukes for å holde orden på værstasjoner, med stasjonsnummer som nøkkel.

Eksempel på å sette inn i de to HashMap'ene:

```
// Anta at opplysninger om en stasjon er lest fra fil
Stasjon st = new Stasjon(stNr, stNavn, stHøyde, stKommune, stFylke);
hashNummStasjon.put(stNr, st);
hashNavnStasjon.put(stNavn, st);
```

Viktig: det er samme stasjonsobjekt som skal settes inn i begge HashMap'er!

Eksempel på å hente ut fra de to HashMap'ene:

```
Stasjon st = (Stasjon) hashNummStasjon.get(stasjonsNummer);

Stasjon st = (Stasjon) hashNavnStasjon.get(stasjonsNavn);
```

Oblig 4: trinn 3

`void lesStasjonsdata(String filnavn)`

Skal lese fil med data om værstasjoner. Gå i løkke, hvor hvert gjennomløp leser en linje fra fila, lager et objekt av klassen Stasjon og fyller med innleste verdier, og legger objektet inn i de to HashMap'ene.

`void lesMeteorologidata(String filnavn)`

Skal lese fil med værdata.

Mulig løsning 1: gå i løkke, hvor hvert gjennomløp leser en linje fra fila, får tak i riktig Stasjons-objekt (ikke lag nytt Stasjons-objekt, bruk HashMap'ene) og oppdaterer dette objektet med de nye værdataene ved å kalle på en passende metode i Stasjonsobjektet (f.eks. registrerMåling(....)).

Stasjonsobjektet må så finne ut hvilket Maaned-objekt som skal oppdateres, og delegerer oppdraget videre til dette objektet ved å kalle på en passende metode i dette Maaned-objektet (f.eks. nyMåling(...)).

Mulig løsning 2: i stedet for at fila leses sentralt (i Analyse-objektet) kan selve jobben med å lese data om en gitt stasjon delegeres til Stasjons-objektet. Verken mer eller mindre objektorientert, men noen liker dette bedre.

Oblig 4: trinn 4

- Programmer ordreløkken
  - For hver kommando som skal utføres, skal ordreløkken kalle på en passende metode i klassen VaerAnalyse. Du skal altså ikke legge programkode for å utføre kommandoer i løkka, bare delegere oppdraget videre.
  - 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 `lagStasjonsliste()`, så deklarerer du samtidig denne "dummy-metoden" i klassen VaerAnalyse:

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


Oblig 4: trinn 5

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

```
void finnAntallUværsdager() {
  <Løp gjennom en av HashMap'ene og skriv ut liste over alle
  stasjoner, med stasjonsnr og stasjonsnavn for hver av dem>

  System.out.println("Stasjonsnummer: ");
  String stasjonsNr = tast.inLine();
  System.out.println("Måned (1-6): ");
  int mnd = tast.inInt();

  Stasjon st = <finn stasjonen ved oppslag i hashNummStasjon>;
  int antDager = st.finnAntUværsdager(mnd);

  <Skriv ut resultatet>
}
```

Oppdraget delegeres videre til en metode i Stasjons-objektet som er berørt.

Oblig 4: trinn 6

- Skriv de metodene som kalles fra metodene i trinn 5.
- Eksempel (i klassen Stasjon):

```
int finnAntUværsdager(int mnd) {
  Maaned md = mdata[mnd-1];
  int antDager = md.finnAntUværsdager();
  return antDager;
}
```

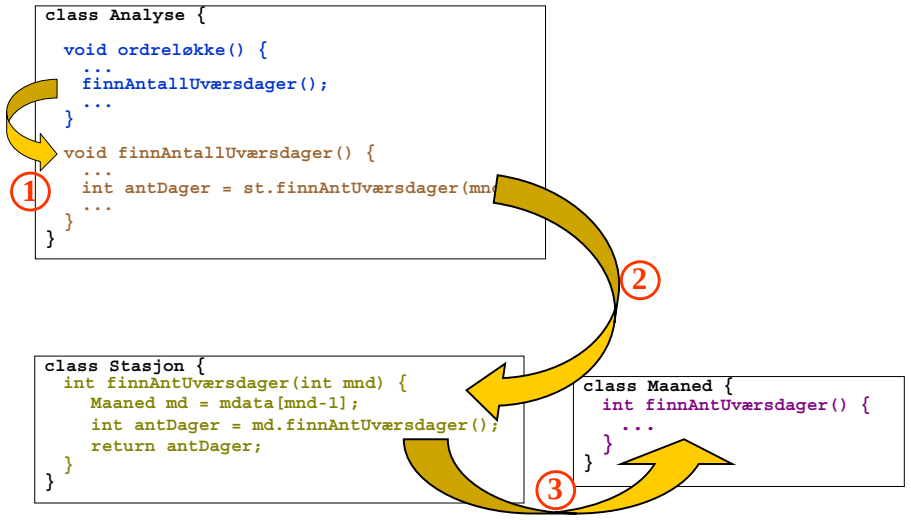
Oppdraget delegeres videre til en metode i Maaned-objektet som er berørt.

Oblig 4: trinn 7

- Skriv de metodene som kalles fra metodene i trinn 6.
- Eksempel (i klassen Maaned):

```
int finnAntUværsdager() {
  <gå gjennom data for alle dager i måneden og tell opp
  hvor mange som oppfyller kravet til uværsdag>

  <returner med antallet uværsdager>
}
```

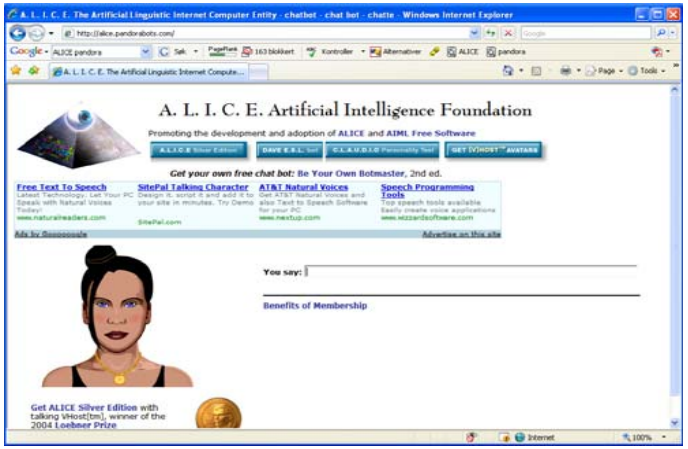
ELIZA - den elektroniske psykiater

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 you have a life? 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 (evt et standard-svar 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("Antall ord lest: " + 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. Fortell mer.");
        }

        if (tast.hasNext()) {
            tast.readLine();  // Tømmer inputbufferet
        }
     }
    }
}
```

Du finner hele programmet Eliza.java på nettsidene til kurset for høsten 2006 (se detaljert undervisningsplan h2006), sammen med fila ord.txt.