Uke 11 – noen tips og råd + eksempel

Noen generelle råd vedrørende oppgaveløsing

Noen råd i forbindelse med oblig 4

Vi utvider eksempelet i avsnitt 9.9 side 186 i læreboka

1. november 2005

Valg av datamodell: eksempel

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

- **Forslag 1: Gruppere tellinger relatert til samme sykdom**

```
class Sykdom {
   String navn;
   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 av hva du skal bruke dataene til.

Valg av datamodell: oblig 4

Anta at du har gitt en fil med opplysninger om en rekke værstasjoner:

4780	GARDERMOEN	202	ULLENSAKER	AKERSHUS
10400	RØROS	628	RØROS	SØR-TRØNDELAG
18700	OSLO-BLINDERN	94	OSLO	OSLO
25590	GEILO-GEILOSTØLEN	810	HOL	BUSKERUD
.....				

Her virker det naturlig å gruppere dataene stasjonsvis:

```
class Stasjon {
    String nummer;
    String navn;
    double høyde;
    String kommune;
    String fylke;
}
```

Valg av datamodell: oblig 4

For hver værstasjon har du gitt daglige værmålinger for et halvt år:

stasjon	dag	mnd	maxvind	nedbør	mintemp	maxtemp
4780	01	01	1.5	0.0	-23.6	-13.3
4780	02	01	2.6	0.8	-13.7	-10.0
4780	03	01	4.6	1.3	-17.9	-11.7
4780	04	01	4.6	0.1	-23.3	-16.7
4780	05	01	5.7	0.0	-22.9	-15.7
4780	06	01	3.1	1.0	-22.9	-16.0
.....						

Hvordan modellerer vi disse dataene? Svaret er mindre opplagt.

Valg av datamodell: oblig 4

- **Forslag 1: Ingen gruppering**

```
class Stasjon {
  double[] maxvind;
  double[] nedbør;
  double[] mintemp;
  double[] maxtemp;
  ...
}
```

- **Forslag 2: Gruppere målinger utført samtidig (dvs gruppere etter dag)**

```
class Dagdata {
   double maxvind;
   double nedbør;
   double mintemp;
   double maxtemp;
}

class Stasjon {
   Dagdata[] dagdata;
}
```


Valg av datamodell: oblig 4

- **Forslag 3: gruppere etter både måned og dag**

```
class Dagdata {
   double maxvind;
   double nedbør;
   double mintemp;
   double maxtemp;
}

class Maanedsdata {
   Dagdata[] dagdata = new Dagdata[31];
   int antDager;  // Antall dager i måneden
}

class Stasjon {
   Maanedsdata[] mdata = new Maanedsdata[6];
}
```

Dette er den varianten som blir anbefalt i hjelpenotatet for oblig 4.

Organisering av veldig små programmer

- I *veldig små programmer* (noen få linjer) med en enkelt klasse og ingen objekter av egne klasser, kan all programkode ligge i main-metoden:

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

class VeldigLiteProgram {
    public static void main (String [] args) {
        In tast = new In();
        System.out.print("Gi et filnavn: ");
        String fnavn = tast.inLine();
        if (new File(fnavn).exists()) {
            System.out.println("Filen fins");
        } else {
            System.out.println("Filen fins ikke");
        }
    }
}
```

Organisering av alle andre programmer

- I *alle andre programmer* bør main-metoden kun brukes til å få igang programmet. Eksempel:

```
class MittProgram {
    public static void main (String [] args) {
        Flyreservasjon fr = new Flyreservasjon();
        fr.ordreløkke();
    }
}

class Flyreservasjon {
    void ordreløkke() {
        ...
    }
}
```


Variant 1

- Det er mulig å slå sammen de to klassene på forrige foil til en klasse:

```
class MittProgram {
    public static void main (String [] args) {
        MittProgram mp = new MittProgram();
        mp.ordreløkke();
    }
    void ordreløkke() {
        ...
    }
}
```

- Det blir en smakssak om man velger å gjøre det slik, eller slik som på forrige foil. I undervisningen vil du hovedsakelig se eksempler på den varianten som bruker to klasser.

Variant 2

- Istedet for å lage en egen ordreløkke-metode, kan koden for ordreløkken ligge i konstruktøren:

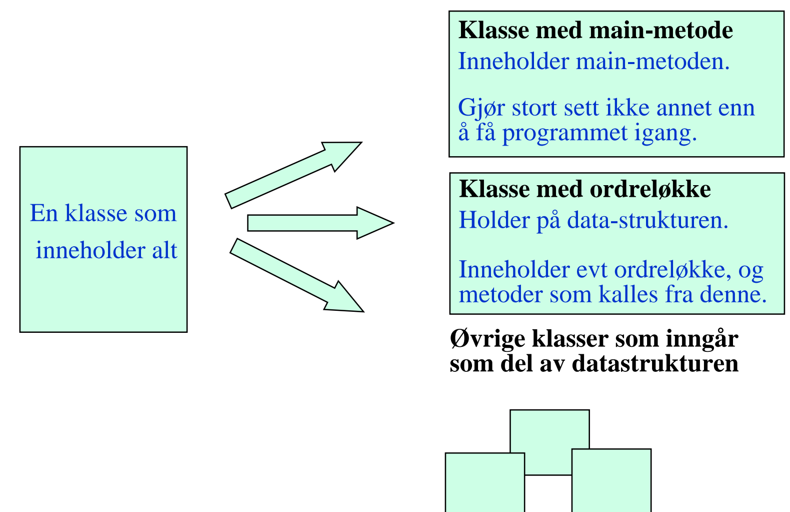
```
class MittProgram {
    public static void main (String [] args) {
        Flyreservasjon fr = new Flyreservasjon();
    }
}

class Flyreservasjon {
    Flyreservasjon() {
        ... ordreløkken ...
    }

}
```

- Dette blir også en smakssak, men varianten ovenfor bruker konstruktøren til noe annet enn initialisering av objektvariable og er mindre gjennomsiktig/selvforklarende enn å bruke en egen ordreløkke-metode.

Rollefordeling i større programmer



Oppsummering

- Legg ikke annen programkode i main enn det som skal til for å få igang programmet (typisk: lage et objekt og kalle på en metode i dette), unntatt i bittesmå programmer.
- I main kan du velge å lage et objekt av klassen som main ligger i - eller et objekt av en annen klasse. I kurset bruker vi stort sett siste variant.
- Metoden som kalles fra main er typisk den som er ansvarlig for programkontrollen (f.eks. en ordreløkke), eller en del av den.
- Programkontrollen kan ligge i en konstruktør eller i en annen metode (f.eks. `void ordreløkke()`). I kurset gjør vi stort sett det siste.
- **Råd:** bestem deg for hvilken organisering du foretrekker (f.eks. den på foil 3) og hold deg til den når du skriver dine egne programmer - så slipper du å bruke tid på å velge hver gang du skal lage et nytt program.

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.

Eksempel: oblig 4

`class Oblig4`

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

`class VaerAnalyse`

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 Maanedsdata`

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

`class Dagdata`

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 = "Stasjoner-1.txt";
       String s2 = "Vaerdata-1.txt";
       if (args.length >= 2) {
          s1 = args[0];
          s2 = args[1];
       }
       VaerAnalyse va = new VaerAnalyse(s1, s2);
       va.ordreløkke();
   }
}
```

Programmet kan startes som
`java Oblig4`
eller
`java Oblig4 filnavn1 filnavn2`

Oblig 4: trinn 2

```
class VaerAnalyse {
   HashMap<String,Stasjon> stasjonFraNavn = new HashMap<String,Stasjon> ();
   HashMap<String,Stasjon> stasjonFraNr   = new HashMap<String,Stasjon> ();

   VaerAnalyse(String s1, String s2) {
       lesStasjonerFraFil(s1);
       lesVaerDataFraFil(s2);
   }

   void lesStasjonerFraFil(String fnavn) {...}
   void lesVaerDataFraFil(String fnavn) {...}
   void ordreløkke() {...}

   ...
}
```

} 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

NB: Dette er kode for Java 1.5!

De to HashMap'ene

```
HashMap<String,Stasjon> stasjonFraNavn = new
   HashMap<String,Stasjon> ();
```

Brukes for å holde orden på værstasjoner, med stasjonsnavn som nøkkel.

```
HashMap<String,Stasjon> stasjonFraNr = new
   HashMap<String,Stasjon> ();
```

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);
stasjonFraNr.put(stNr, st);
stasjonFraNavn.put(stNavn, st);
```

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

Oblig 4: trinn 3

`void lesStasjonerFraFil(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 lesVaerDataFraFil(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 Maanedsdata-objekt som skal oppdateres, og delegerer oppdraget videre til dette objektet ved å kalle på en passende metode i dette Maanedsdata-objektet (f.eks. nyMåling(...)).

Mulig løsning 2: i stedet for at fila leses sentralt (i VaerAnalyse-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 VaerAnalyse):

```
void finnAntUværsdager() {
   <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.inWord("\n");
   System.out.println("Måned (1-6): ");
   int mnd = tast.inInt();

   Stasjon st = <finn stasjonen ved oppslag i stasjonFraNr>;
   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) {
   Maanedsdata md = mdata[mnd-1];
   int antDager = md.finnAntUværsdager();
   return antDager;
}
```

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

Oblig 4: trinn 7

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

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

   <returner med antallet uværsdager>
}
```


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 programkoden 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 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.