INF1000 – Uke 10

Hvordan gripe an et stort problem? 5 råd
Noen eksempler du kan overføre til Oblig 4
Oblig 4 – kort oversikt
Et større programeksempel

Råd 1: Programmer ”ovenfra ned”

- Hvilke klasser skal være med?
  - Les oppgaven
  - Se etter substantiver
  - Lag klassediagram
- Bestem datastruktur
  - Hvordan er input-dataene?
  - Fyll inn de mest sentrale variablene
  - Trengs nye klasser?
- Følg programflyten når du bestemmer metoder
  - Skriv først metodene på ”toppnivå” f.eks. en kommandoløkke
  - Kall på metoder ved behov, selv om disse ennå ikke er skrevet
  - Skriv metodene du kaller på, og fortsett til programmet er ferdig

Velg datastruktur etter hva som skal gjøres!

- I objekt-orientert programmering
  - *tenker vi i form av objekter*
  - *men programmer i form av klasser*
- Prøv å gruppere data etter objekter som “eier” dem
  - variable og metoder som logisk hører sammen bør ligge samlet
  - variable og metoder som ikke har noe med hverandre å gjøre bør holdes godt atskilt

Gruppering etter ”eier”

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

Info knyttet til person splittet opp i tre arrayer


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

Info samlet i ett objekt.

Data og metoder sammen

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



```
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

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

Valg av datamodell: nytt eksempel

Gitt fil med opplysninger om antall registrerte tilfeller det var av tre ulike sykdommer i Norge :

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

Hvordan er det naturlig å modellere dette?

Noen muligheter

Gruppere tellinger relatert til samme sykdom

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

Gruppere tellinger foretatt samtidig

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

Ingen gruppering – tre arrayer

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

Ingen gruppering – en 2D-array

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

Beste datastruktur avhenger av hva du skal bruke dataene til!

Råd 2: Metoder "utenfra og inn"

- Hva 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).

UNIVERSITETET I OSLO

Råd 3: Deleger oppgaver

- Stykk opp oppgavene og fordel dem
- Dermed blir hver enkelt del mer oversiktlig
  - faren for feil minker
  - lettere å finne feil senere
- Ofte lurt: Deleger operasjoner på data til objekter som er ”nærme” dataene
- Ikke overdriv delegering!
  - hvert objekt trenger ikke metoder for å lese fra terminal!
  - av og til bedre å gjøre ting sentralt og kalle på metoder i objektene for å oppdatere deres variable

UNIVERSITETET I OSLO

Ideen bak objektorientering

- Hvert objekt skal ha sin bestemte og naturlige oppgave
- Kollektivt samarbeid om å løse oppgavene
- Objektene opprettes som instanser av klasser
- Objektene samarbeider ved å sende meldinger:
  - kaller hverandres metoder
  - overfører informasjon i form av parametere og returverdier
- Metodekallene har en entydig mottager som overtar ansvaret for oppgaven

UNIVERSITETET I OSLO

Helhet og deloppgaver i OOP

- *Vi trenger ikke å ha oversikt over hele programmet eller hele datastrukturen* når vi skriver en metode
- Vi ”skifter hatt” og ser systemet gjennom øynene til hver av aktørene for seg
  - Når vi er kelner, beskriver vi kelneren ut fra kelnerens perspektiv, når vi er hovmesteren ser vi det ut fra hans perspektiv osv.
- Vi programmerer en klasse ut fra klassens perspektiv og glemmer da resten av helheten

UNIVERSITETET I OSLO

Objekter svarer til programmer

- Vi kan tenke oss at hvert objekt av en gitt klasse svarer til en ”kjøring av klassen”
- Et objekt kan *samarbeide* med alle det kjenner til ved metodekall
- Vi starter ett av ”programmene” ved å kalle klassens main-metode
- Deretter: objektene *opprettes* når programmet kjører

Råd 4: Formater koden

```
    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");
   }}}
```

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!

Råd 5: Ingen skam å snu!

- Programmer blir til ved at vi jobber litt her og der
  - Vi kan bruke mange runder før vi er fornøyd
- Vi finner ofte ut at vi trenger flere klasser
  - eller at en klasse bare er "i veien" og fjerner den
- Det endelige programmet kan ha andre klasser og metoder enn vi startet med
- Pass likevel på å holde programmet kompilerbart!
  - Lag tomme metoder som du kan fylle ut siden
  - Hold koden ryddig

Flyreservasjon

Klasser

Egenskaper

Prosedyrer

- 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
- Systemet skal lese inn beskrivelse av flyene, med antall seter, klasser på de forskjellig seteradene, osv
- Det skal kunne reservere seter, avbestille og skrive ut en oversikt over flyets seter, med klasse og om det er ledig eller ikke

Klasseinndeling

- 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
- class Seterad
  - Setene i raden
- class Sete
  - Klasse og om det er opptatt eller ikke

class 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();
  }
}
```

class 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: ordreløkken

- For hver kommando skal ordreløkken kalle på en metode i klassen Flyreservasjon.
- 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
- Eksempel: kaller ordreløkken på metoden visFlyvning(), kan du skrive en "dummy-metode":

```
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.

class 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();
  }
}
```

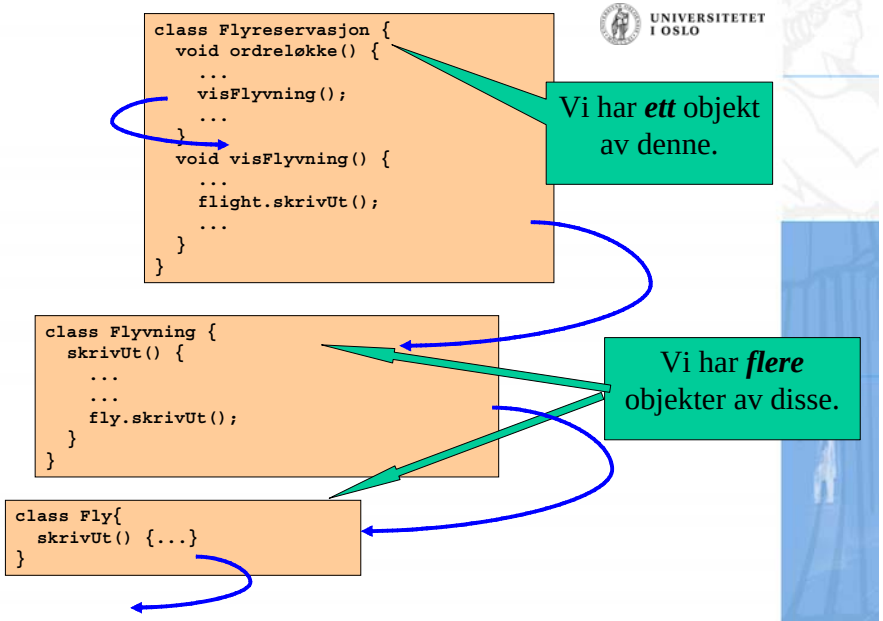
Oppdraget delegeres videre til en metode i Fly-objektet

class Fly

- Skriver ut informsjon on flyet
- Delegerer videre til seteradene
- 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



Oblig4: Komme i gang

Les hele oppgaven. Identifiser klasser
Lag et enkelt klassediagram
Skisser de viktigste variablene og metodene
Lag et skall til programmet

Råd 1: Programmer "ovenfra ned"
Råd 2: Metoder "utenfra og inn"
Råd 3: Deleger oppgaver
Råd 4: Formater alltid koden underveis
Råd 5: Ingen skam å snu!

Les teksten, finn 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,
  - fylket,
  - regionen hvor stasjonen ligger.

Stnr	Navn	Hoh	Kommune	Fylke	Region
180	TRYSIL_VEGSTASJON	360	TRYSIL	HEDMARK	ØSTLANDET
5590	KONGSVINGER	148	KONGSVINGER	HEDMARK	ØSTLANDET
10380	RØROS_LUFTHAVN	625	RØROS	SØR-TRØNDELAG	MIDT-NORGE
94500	FRUHOLMEN_FYR	13	MÅSØY	FINNMARK	NORD-NORGE
99950	JAN_MAYEN	10	JAN MAYEN	JAN MAYEN	ARKTIS

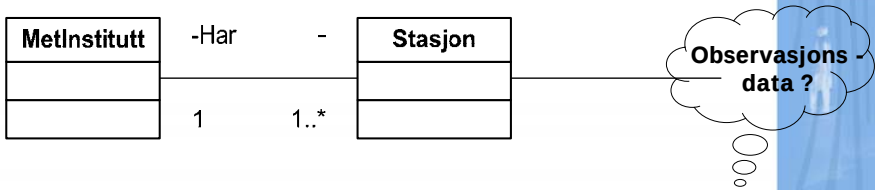
Se på substantivene!

- 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,
  - fylket,
  - regionen hvor stasjonen ligger.

Stnr	Navn	Hoh	Kommune	Fylke	Region
180	TRYSIL_VEGSTASJON	360	TRYSIL	HEDMARK	ØSTLANDET
5590	KONGSVINGER	148	KONGSVINGER	HEDMARK	ØSTLANDET
10380	RØROS_LUFTHAVN	625	RØROS	SØR-TRØNDELAG	MIDT-NORGE
94500	FRUHOLMEN_FYR	13	MÅSØY	FINNMARK	NORD-NORGE
99950	JAN_MAYEN	10	JAN MAYEN	JAN MAYEN	ARKTIS

Skisse klassediagram

- Hvilke substantiver passer som klasser?
  - De som ***er*** noe
- Hva er variable i klassene?
  - Egenskaper ved klassene
- Hva passer som metoder
  - Det som ***gjøres*** med eller av klassene



Observasjonsdata

- FFM Middel av vindhastigheter m/s
- FFX Høyeste vindhastighet m/s
- RR Nedbør mm
- SA Snødybde cm
- TAM Middeltemperatur ºC
- TAN Minimumstemperatur ºC
- TAX Maksimumstemperatur ºC

Stnr	Dato	FFM	FFX	RR	SA	TAM	TAN	TAX
180	01.01.2009	0,6	1,1	0,1	-999	-20,4	-24,5	-16,2

UNIVERSITETET I OSLO

Manglende data

- "-999" betegner at observasjon mangler
- Ved beregning av gjennomsnitt ol, må det sjekkes om verdiene er -999
  - Kan bruke en konstant (final int) !
- 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

UNIVERSITETET I OSLO

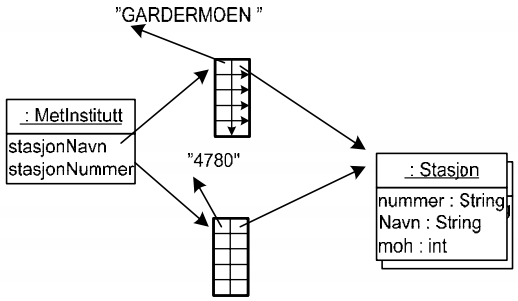
ÆØÅ-problemer

- Bruker du PC via Windows, blir æ, ø, å, Æ, Ø, Å blir skrevet ut som andre tegn på skjermen
- Hvis man f.eks taster inn æ, ø, å på en Windowspc, vil de ikke bli oppfattet som de æ-ene, ø-ene og å-ene du har lest inn fra fila.
  - Eksempler: NY-□LESUND (99910) MIDTL□GER (46510), VARD□ (98550))
- *Bruk stasjonsnumrene* til å identifisere stasjonene, både i brukerdialogen og når du leser data-filene.
  - Skriv derfor både ut navn og nummer på stasjonene
- Når brukeren skal velge stasjon:
  - List opp alle stasjoner med nummer og navn
  - Be brukeren skrive inn stasjonsnummer

UNIVERSITETET I OSLO

To HashMap-er?

- Trenger du å kunne søke ut fra både stasjonsnr og navn?
- Bruk *to* HashMap'er!
  - en hvor nøkkelen er navnet
  - 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

Unngå int som nøkkel i HashMap

- Trenger du en nøkkel til en HashMap av noe som egentlig er et heltall, si: **int num**?
- Lag en nøkkel slik:

```
String numSomTekst = "" + num;
```

- Dette lager en `String` med tallverdien i `num` ved å legge til den tomme tekststrengen

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
```

Javadoc

- Javadoc-generering av systemet:

```
>javadoc –package *.java
```

- **–package:** alle variable, klasser og metoder uten modifikator, samt de med public foran, blir med
  - Har du ikke med **–package,** kommer bare **public** med
- La kun ”.java”-filene på filområdet tilhøre Oblig4!

Korte Javadoc-kommentarer

- Skrives på en linje:

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

- Javadoc-kommentarer for en variabel skrives over deklarasjonen:

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