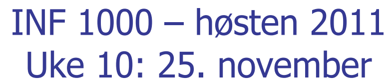


INF 1000 – høsten 2011
Uke 10: 25. november

Grunnkurs i Objektorientert Programmering
Institutt for Informatikk
Universitetet i Oslo

Kursansvarlige: Arne Maus og Siri Moe Jensen

Innhold – uke 10

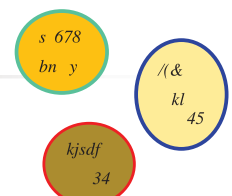
- Mer om objektorientering, klasser og metoder
- Repetisjon forrige uke: HashMaps
- Hvordan gripe an et "stort" problem? 5 råd
- Eksempel: Flyreservasjon

Mål for uke 10:
** Mer utfyllende om klasser*
** Hvordan designe og programmere objektorientert*
** Støtte til oppstart på Oblig 4: Jobbe frem eget forslag til datastruktur*

Objektorientering: Hvorfor?

- Et program (særlig i eldre og mer lavnivå språk) *kan* bestå av en lang rekke setninger der kontrollen hopper frem og tilbake, kanskje bare med direkte "hopp"-instruksjoner ("goto")
- Data *kan* deklareres som en "uendelig" lang rekke felles variable, som brukes når de trengs og er tilgjengelige over alt
- Dette er uproblematisk for datamaskinen som skal utføre programmet, MEN:
  - det blir fort vanskelig å holde oversikten for programmereren
  - svært tidkrevende å finne feil
  - nesten umulig å sette seg inn i for andre (eller huske hvordan det fungerte noen måneder etter at man skrev det)
  - vanskelig (farlig!) å utvide/ endre
  - nesten umulig å dele opp for å samarbeide om utvikling

Objektorientering: Hvordan?



Think globally, act locally!

Data:
- Tenk helhet, overordnet gruppering gjennom identifisering av viktige "ting" (substantiver) -> klasser
- Arbeid lokalt med bestanddeler, representasjon -> variable

Handlinger (metoder):
- Først hvilke oppgaver som skal løses, og hvor de best løses (hvor er dataene de skal arbeide med)
- ..deretter: Hvordan løse oppgaven

Hva er en klasse?

- En klasse er et mønster for/ 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
  - En egenskap i en klasse som viser til en annen klasse representeres som en peker-variabel til et objekt av den andre klassen. (Til "mange" objekter: Peker-array eller HashMap)

Skille mellom deklarasjon og bruk av en klasse

- Klassedeklarasjoner i programmet medfører 'lite' aktivitet:
  - De variable og metodene det står `static` foran er tilgjengelige (klassevariable og klassemetoder)
  - ..dvs static metoder kan kalles (eks. `main` ved start av programmet)
  - Mønsteret for nye objekter av klassen er tilgjengelig
  - Hvis det aldri blir sagt `new` på klassen, finnes det ingen objekter, og dermed ingen objektvariable eller objektmetoder
- Når utførelsen kommer til en setning med `new` på en klasse, får vi laget et objekt av klassen
  - Objektet inneholder alle objekt-variable og – metoder (variable og metoder som ikke har `static` foran deklarasjonen i klassen)
  - Det kalles automatisk en konstruktør-metode i klassen, og først når den er ferdig, returnerer `new` med det nye objektet. Konstruktøren sørger for at objektvariablene i det nye objektet har de initialverdiene vi ønsker. Disse verdiene kan for eksempel hentes fra parametre til konstruktøren eller leses fra fil eller terminal.

Klassemetoder(-variable) og objektmetoder(-variable)

- Klassemetoder (static-metoder)
  - Definert selv om det ikke er laget noen objekter av klassen
  - Kan "ses" av alle objekter av klassen
  - Kan brukes av andre gjennom dot-notasjon: <klassenavn>.metode(...)
  - Har ikke tilgang til objektvariable eller objektmetoder
- Objektmetoder
  - Bare definert i objekter av klassen (opprettet med "new")
  - Kan "ses" av objektet som metoden befinner seg i
  - Kan brukes av andre gjennom dot-notasjon: <peker>.metode(...)
  - Har tilgang til alle variable (både klassevariable og objektvariable) og alle metoder (både klassemetoder og objektmetoder)

Forskjeller mellom klasser og metoder

Begge lager objekter når de kalles, men:

- Et metode-objekt:
  - fjernes når metoden returnerer
  - inneholder 'bare' variable og parametere som alle er skjult for resten av programmet
- Et objekt laget med new fra en klasse:
  - 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 .)

Oppgave

*Vi skal lage en konstruktør til klassen Student. Konstruktøren skal ha studentens navn som parameter og skal initiere objektvariabelen **navn**.*

*Vi skal også skrive en objektmetode **void økPoeng(int poeng)** i klassen Student som øker antall studiepoeng for en student med parameterens verdi.*

```
class Student {
      String navn;
      int antallStudiepoeng = 0;
      // Her skal du skrive konstruktøren


      // Her skal du skrive objektmetoden økPoeng


}
```

Konstruktør og metode i klassen Student

```
class Student {
      String navn;
      int antallStudiepoeng = 0;
      // Her skal du skrive konstruktøren

      Student (String navn) {
           this.navn = navn;
      }


      // Her skal du skrive objektmetoden økPoeng
      void økPoeng (int poeng) {
           antallStudiepoeng += poeng;
      }
}
```

Oppgave

*Vi skal lage en konstruktør til klassen Student. Konstruktøren skal ha studentens navn som parameter og skal initiere objektvariabelen **navn**.*

*Vi skal også skrive en objektmetode **void økPoeng(int poeng)** i klassen Student som øker antall studiepoeng for en student med parameterens verdi.*

*Du skal deretter deklarere klassen StudentTest som kan holde rede på et antall Student-objekter i en array **uioStud** - hvor mange bestemmes når StudentTest-objektet opprettes.*

*Studentobjektene i arrayen **uioStud** skal initieres med hvert sitt studentnavn **Stud-1**, **Stud-2**, **Stud-3**, ... **Stud-32000**. Dermed skal f.eks. **uioStud[252]** peke på et Student-objekt hvor studentens navn er satt lik **Stud-253**.*

Opprettelse og initiering av Studentobjekter

```
class StudentTest {
    Student[] uioStud;
    int ant;

    StudentTest(ant) {
       this.ant = ant;
       uioStud = new Student[ant];
       for (int i = 0; i<= ant; i++) {
         uioStud[i] = new Student ("Stud-" + (i+1));
       }
    } // end konstruktør for StudentTest

} // end class StudentTest

<i main:>
       ...
       // oppretter register med 32000 studenter:
       StudentTest st = new StudentTest (32000);
       // 30 ekstra studiepoeng til de første 25000 studentene
       for (int i=0; i<25000; i++) {
         st.uioStud[i].økPoeng(30);
       }
       ...
```

Innkapsling

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

- 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 metodene `settInn()` og `taUt()`.
- Vi kan regulere tilgangen til variable og metoder ved å sette en av følgende modifikatorer foran en variabel- eller metodedeklarasjon:
  - `private`
  - `public`
  - `protected`

Hvis alle .java filene ligger på samme filområde

- ingenting foran en deklarasjon/metode
  - fullt tilgjengelig for all annen kode på samme filområde
  - usynlig /sperret for kode på andre filområder.
- **`private`**
  - kun synlig fra metoder deklarert i samme klasse, usynlig/sperret for all annen kode
- **`public:`**
  - Synlig for "all" annen kode (forutsetter katalogen referert i CLASSPATH)
  - Nødvendig for `main`
  - Ellers det samme som <ingenting> i vårt tilfelle

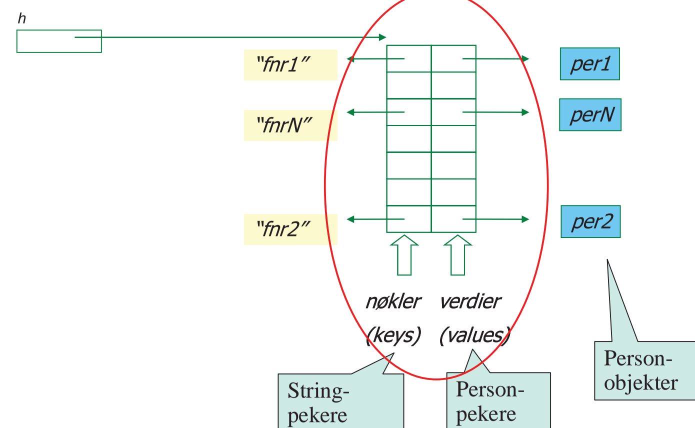
Slik delvis sperring av adgang sikrer oss at vi selv bestemmer hvordan en variabel i en klasse skal endres.

HashMap - repetisjon

- Hvorfor/ når?
  - ukjent antall objekter av en type (array har fast lengde)
  - direkte oppslag på andre (eller flere) egenskaper enn indeks 0-N
- Hva inneholder et HashMap?
  - "sekk" med objekter av samme type (i INF1000)
  - hvert objekt har en unik identifikator (av typen String i INF1000)
- Hva kan et HashMap?
  - Legge inn et nytt objekt
  - Slå opp et objekt
  - Fjerne et objekt
  - Lage en samling av alle verdier (pekere) (tilfeldig rekkefølge)
  - Lage en samling av alle identifikatorer (tilfeldig rekkefølge)
  - ++, se s190 i boken

Hva er et HashMap?

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



Hva må jeg kunne om HashMaps?

- Bruke det – stikkord: Ukjent antall objekter, oppslag
- Klargjøre og deklarere
  - `import java.util.*;`
  - `HashMap<String,Person> h = new HashMap <String,Person>();`
- Legge inn objekter
  - `h.put (s,k);`
- Hente ut (lese) objekter
  - `p=h.get(s);`
- Fjerne objekter
  - `h.remove (s);`
- Lese antall objekter

- Gjennomløpe alle objektene (NB usortert!) vha peker eller nøkler
  - `for (Person p: h.values()) {}`
  - `for (String s: h.keyset()) {}`
  - se RpJ s. 187-188 om iterator og videre behandling, eks sortering

Oppgave: Minimalistisk HashMap

*Filen **SpamOrd.txt** inneholder en del spam-ord (atskilt av blanke tegn).*

*Du skal lage et program som leser **SpamOrd.txt** og som deretter leser filen **Epost.txt** og sjekker om denne inneholder noen av spam-ordene. Programmet skriver til slutt ut på skjermen om filen **Epost.txt** inneholdt noen spam-ord eller ikke (du trenger ikke å skrive ut hvilke eller hvor mange spam-ord filen inneholdt).*

*Du kan anta at **SpamOrd.txt** maksimalt inneholder 200 ord. I denne oppgaven kan du godt legge all programkoden inn i main-metoden.*

Løsning

*

- Trenger en String-variabel for hvert ord (spam-kjennetegn) fra spam-filen

- Trenger å holde rede på ukjent antall ord

- Må kunne slå opp ett og ett ord fra mail-fil mot spam-ordene

- Lager et HashMap med spam-ord som nøkler

- Hva lagrer vi som *verdier* i HashMap'en??

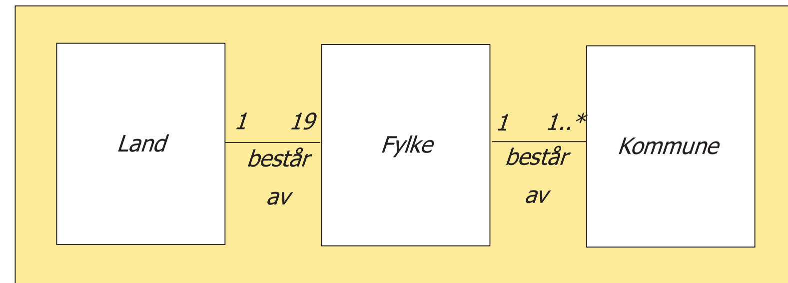

```
import easyIO.*;
import java.util.*;

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

      In spamfil = new In("SpamOrd.txt");
      while (!spamfil.lastItem()) {
         h.put(spamfil.inWord(), null);
      }

      In epostfil = new In("Epost.txt");
      boolean funnet = false;
      while (!epostfil.lastItem() && !funnet) {
         String s = epostfil.inWord();
         if (h.containsKey(s))
            funnet = true;
      }

      if (funnet) System.out.println("Spam-ord !");
      else System.out.println("Ingen spam-ord");
   }
}
```

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

Oppgave

Norge består av 19 fylker som hver består av et antall kommuner. Anta at vi modellerer dette ved hjelp av tre klasser **`Land`**, **`Fylke`** *og* **`Kommune`** *slik at Norge representeres ved et objekt av klassen* **`Land`**, *hvert fylke er representert ved et objekt av klassen* **`Fylke`** *og hver kommune er representert ved et objekt av klassen* **`Kommune`**. *Lag et UML klassediagram som viser forholdet mellom de tre klassene* **`Land`**, **`Fylke`** *og* **`Kommune`**, *og få på riktige antall i hver ende av forholdene.*



Velg datastruktur etter hva som skal gjøres!

- I objekt-orientert programmering
  - tenker vi i form av objekter
  - men programmer i form av klasser
- Spør: Hva kan objektene gjøre for meg?
  - Flytt funksjonalitet ut i objektene – deleger!
- 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];
```

Ikke objektorientert:
Info om person splittet opp i tre arrayer

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

Info om én person samlet i samme objekt, beskrevet ved klassen Person

NB, objektene må opprettes med **`new`**

Data og metoder hører 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 samles
** 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*

Råd 2: Metoder "utenfra og inn"

- Hva er input og output til metoden du skal skrive?
- Input:
  - Kan være data metoden finner i (klassevariable eller) objektvariable eller noe som skal leses fra bruker eller fil
  - Hvis det bestemmes når metoden kalles, og vi vet verdien på kallstedet, bruker vi parametere til metoden
- Output:
  - Kan være modifikasjoner av (klassevariable eller) objektvariable, eller noe som skrives ut til bruker eller på fil
  - Hvis noe skal tas med tilbake til kallstedet og brukes videre i programmet lager vi en returverdi fra metoden

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
- Delegering kan overdrives..

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 via pekere:
  - kaller hverandres metoder
  - overfører informasjon i form av parametere og returverdier
- Metodekallene har en entydig mottager som overtar ansvaret for oppgaven (metoden i det aktuelle objektet)

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

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

5 råd oppsummert

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

Flyreservasjon

Klasser
Egenskaper
Metoder

- Systemet skal brukes til flyreservasjon: 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 fra en fil med flyvninger
- 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 brukerdialog-metode.
- class Flyreservasjon
  - Inneholder brukerdialogen og andre metoder + HashMap-tabell for å holde orden på flyvningene.
- class Flyvning
  - Hvert objelt har info om kode, avreisested, destinasjon, fly, etc
- class Fly
  - Hvert objekt inneholder id.nr på flyet + alle seteradene og setene
- class Seterad
  - Setene i raden
- class Sete
  - Setenr, 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 = "Flyvninger.txt";
    Flyreservasjon f = new Flyreservasjon(s1);
    f.brukerdialog();
  }
}
```

class Flyreservasjon

Datastruktur

```
class Flyreservasjon {
  HashMap<String, Flyvning> flyvninger = new HashMap<String,Flyvning>();

  Flyreservasjon(String s1) {
    lesFlyvninger(s1);
  }
  void lesFlyvninger(String filnavn) {...}
  void brukerdialog() {...}

  ...
}
```

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 flere metoder som skal kalles fra brukerdialogen

Flyreservasjon: Brukerdialogen

- For hver kommando skal brukerdialogen kalle på en metode i klassen Flyreservasjon
- Sørg for å deklarere alle de metodene som du kaller på fra brukerdialog-metoden
- Du kan vente med å fylle inn innholdet i disse metodene
- Eksempel: kaller brukerdialogen på metoden visFlyvning(), kan du skrive en "dummy-metode"

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

Skrive ut flyvning

Lag kode for metodene som kalles fra brukerdialogen
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

Metoden skrivUt()

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
- … som delegerer til sete

```
class Fly {
  String flykode;
  Seterad[] seterader;
  int skrivUt() {
    System.out.println("Flykode: " + flykode);
    for(int i=0; i<seterader.length; i++){
      System.out.println("Seterad " + (i+1)  + ":");
      seterader[i].skrivUt();
    }
  }
}
```

Fly delegerer til Seterad