Inf1000 (Uke 10)
Oppgaveløsning. Hashmap

Programmering med og uten objekter: hva er forskjellen?
Noen generelle råd vedrørende oppgaveløsing
HashMap

Are Magnus Bruaset og Arild Waaler
Institutt for informatikk
Universitetet i Oslo

To måter å programmere på

Programmering uten objekter

- Var fokus i starten av kurset
- Vi lager ikke objekter av klassene
- Alle variable og metoder er deklarert som static
- Begrepsmessig enkelt, men lite egnet for større programmer

Programmering med objekter:

- Er fokus for resten av kurset (og eksamen).
- Vi lager objekter av klassene (noen eller alle)
- Variable og metoder er vanligvis *ikke* deklarert som static
- Begrepsmessig noe mer komplisert, men mye bedre egnet for større programmer

Eksempel på programmering *uten* objekter

```
class VolumBeregning {
   static double pi = 3.14;
   static int maxAntall = 10;

   public static void main (String [] args) {
     ...
   }

   static double finnVolum(double radius) {
     ...
  }

  static int finnSum(int k) {
     ...
   }
}
```

Alle variable er deklarert som static

Alle metoder er deklarert som static

Eksempel på programmering *med* objekter

```
class StudentRegister {
   public static void main (String [] args) {
     ...
   }
}

class Student {
   String navn;
   String fnr;

   void init() {
     ...
   }

   String finnNavn() {
     ...
   }
}
```

Variable er ikke deklarert som static (vanligvis)

Metoder er ikke deklarert som static (vanligvis)

Organisering av veldig små programmer

- I veldig små programmer (noen få linjer) kan all programkode ofte ligge i main-metoden. Dette blir da naturlig nok *programmering uten objekter* (av egendefinerte klasser).

```
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. Da lager vi som hovedregel objekter av alle andre klasser enn den som inneholder main-metoden. Eksempel:

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

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


Variant

- 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() {
        ...
    }
}
```

- Da lager vi ett enkelt objekt av klassen med main-metoden, og bortsett fra oppstarten i main-metoden foregår programeksekveringen i objektet.
- En smakssak om man velger å gjøre det slik, eller slik som på forrige foil.

Initialisering av variable i et objekt

Anta at programmet vårt inneholder denne klassen:

```
class Person {
   String navn;
   String fnr;
}
```

Når vi har laget et objekt (med new) ønsker vi normalt å gi variablene i objektet fornuftige verdier med en gang. Noen muligheter:

- **Sett verdiene til objektvariablene direkte med prikk-notasjon:**

```
Person p = new Person();
p.navn = "Petter";
p.fnr = "15108559879";
```

- **Lag en init-metode i klassen:**

```
Person p = new Person();
p.init("Petter", "15108559879");
```

- **Benytt en konstruktør:**

```
Person p = new Person("Petter", "15108535738");
```

Konstruktører - repetisjon

En konstruktør er en spesiell type objektmetode som du kan bruke for å sikre at objektet starter sitt liv med fornuftige verdier i objekt-variablene. Konstruktører

- har alltid samme navn som klassen de ligger i
- utføres automatisk når et objekt opprettes med new
- har ingen returverdi, men skal ikke ha void foran seg
- overlastes ofte, dvs det er ofte flere konstruktører i en og samme klasse, hvor konstruktørene skiller seg fra hverandre ved antall parametre og/eller typen på parametrene.

Eksempel

```
class TestSirkel {
   public static void main (String [] args) {
      Sirkel s1 = new Sirkel();
      System.out.println("Radius: " + s1.radius);
      Sirkel s2 = new Sirkel(5.0);
      System.out.println("Radius: " + s2.radius);
   }
}

class Sirkel {
   double radius;

   Sirkel () {
      radius = 1.0;
   }

   Sirkel (double r) {
      radius = r;
   }
}
```

Konstruktør 1

Konstruktør 2

Når det ikke er noen konstruktør

- Når en klasse ikke inneholder noen konstruktør, vil Java selv føye på en "tom konstruktør" uten parametre når programmet kompileres. Dermed er følgende to klassedeklarasjoner ekvivalente:

```
class Sirkel {

   double radius;

   Sirkel () {

   }
}
```

⟺

```
class Sirkel {

   double radius;

}
```

- Merk: dersom klassen inneholder en eller flere konstruktører, vil Java ikke føye på en "tom konstruktør" uten parametre.

Oppgave: virker dette?

Lar dette programme seg kompilere og kjøre?

```
class Oppgave1 {
   public static void main (String [] args) {
      Bil b = new Bil();
   }
}

class Bil {
   String regnr;

   Bil (String regnr) {
      this.regnr = regnr;
   }
}
```

SVAR: nei!

Finn- og sett-metoder

Når man skal lage "virkelige" programmer er det vanlig å

- Deklarere alle objektvariable som **`private`**
- Bruke finn- og sett-metoder for å endre på objektvariable

```
class Eksempel {
  public static void main (String [] args) {
    Student stud = new Student();
    stud.settNavn("Petter");
    System.out.println(stud.finnNavn())
  }
}
class Student {
   private String navn;
   private String fnr;
   void settNavn(String navn) {this.navn = navn;}
   String finnNavn() {return this.navn;}
}
```


Rollefordeling i større programmer

En klasse som inneholder alt

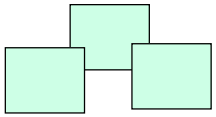
Klasse med main-metode
Inneholder main-metoden.

Gjør stort sett ikke annet enn å få programmet igang.

Klasse med ordreløkke
Holder på data-strukturen.

Inneholder evt ordreløkke, og metoder som kalles fra denne.

Øvrige klasser som inngår som del av datastrukturen



Noen tips

- 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 liker best og hold deg til den når du skriver dine egne programmer - så slipper du å kaste bort tid på å velge hver gang du skal lage et nytt program.

Klassevariable og objektvariable

Objektvariable:

Bare definert i objekter av en klasse.

Hvert objekt har sitt eget sett med objektvariable.

Klassevariable:

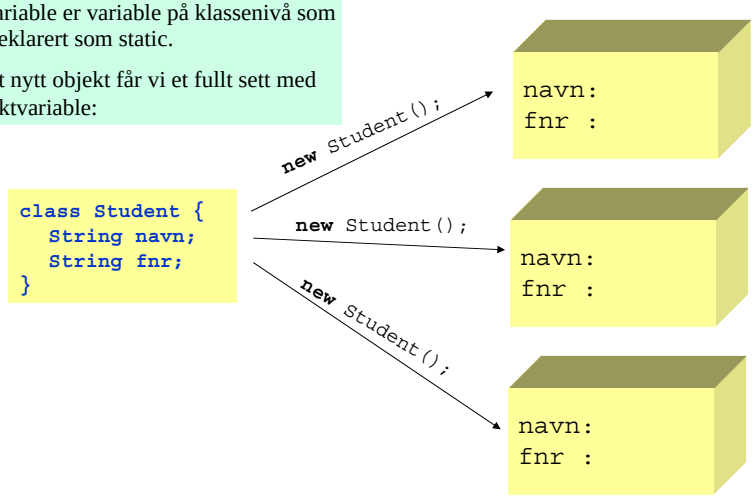
Definert selv om det ikke er laget objekter av klassen.

Alle objekter av klassen deler de samme klassevariablene.

Objektvariable

Objektvariable er variable på klassenivå som *ikke* er deklarert som static.

For hvert nytt objekt får vi et fullt sett med nye objektvariable:



Objektvariablenes levetid

```
class Student {
  String navn;
  String fnr;
}
```

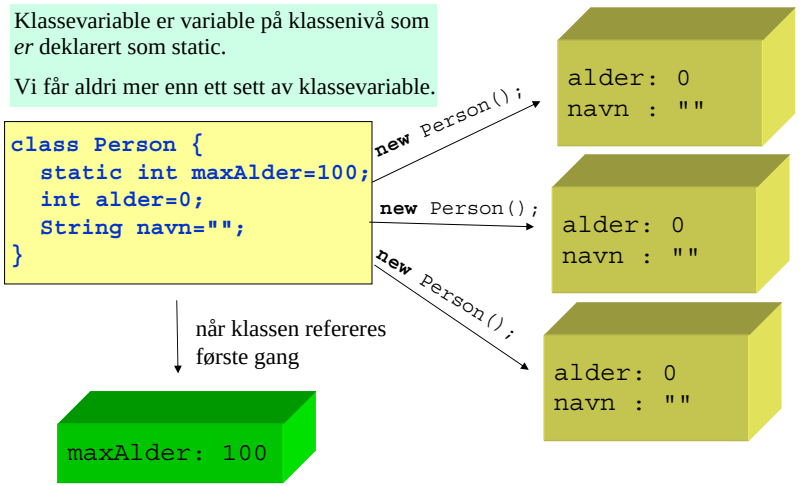
Disse variablene blir deklarert når vi lager et objekt av klassen ved å skrive **`new Student(),`** og de lever så lenge objektet lever.

- Objektvariablene blir til når objektet blir til (med new)
- Objektvariablene lever så lenge objektet finnes

Klassevariable

Klassevariable er variable på klassenivå som *er* deklarert som static.

Vi får aldri mer enn ett sett av klassevariable.



Klassevariablenes levetid

```
class Person {
  static int maxAlder = 100;
  int alder;

  void metode() {...}
}
```

Denne variabelen blir deklarert når klassen Person *blir referert til for første gang* under kjøringen av programmet.

Variabelen lever helt til programmet avsluttes.

Første gang klassen Person blir referert til = første gang programeksekveringen "møter på" Person-klassen, f.eks. :

```
.... new Person() ....
.... Person.maxAlder ....
.... Person.metode() .....
```

Klassemetoder og objektmetoder

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
- 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)

Oppgave

```
class StudentRegister {
  public static void main (String [] args) {
    Student s1, s2;

    s1 = new Student();
    s1.init("Torjus", "S25332");
    s2 = new Student();
    s2.init("Vilde", "S36336");

    s1.skrivUt();
    s2.skrivUt();
  }
}

class Student {
  static String navn;
  static String studId;

  static void init(String n, String s) {
    navn = n;
    studId = s;
  }

  static void skrivUt() {
     System.out.println("Navn: " + navn);
     System.out.println("StudId: " + studId);
  }
}
```

Ole lærer å programmere og har laget programmet til venstre.

Hvorfor skriver programmet følgende ut på skjermen? Hvordan skal vi endre det?

Navn: Vilde
StudId: S36336
Navn: Vilde
StudId: S36336

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)

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

Analogi: hjemme hos deg selv, plasser du verktøy, bestikk og CD-plater i samme skuff? Sannsynligheten er stor for at du allerede tenker objektorientert.

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 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!

Valg av datamodell: eksempel 2

Anta at du skal lese en fil med opplysninger om en rekke værstasjoner, der hver linje gir informasjon om en enkelt værstasjon:

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

En opplagt løsning vil da være å gruppere dataene stasjonsvis:

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

For her linje vi leser fra filen lager vi da et nytt objekt av klassen Stasjon, og fyller objektet med verdiene fra filen

Valg av datamodell: eksempel 2 *forts*.

Anta at det for hver værstasjon også er 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.

Noen muligheter

- **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;
}
```


Noen muligheter *forts.*

- **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];
}
```


Hvordan programmere med objekter?

Grunnregel:
Et objekt skal inneholde data og operasjoner *som naturlig hører sammen.*

Et objekt kan representere:

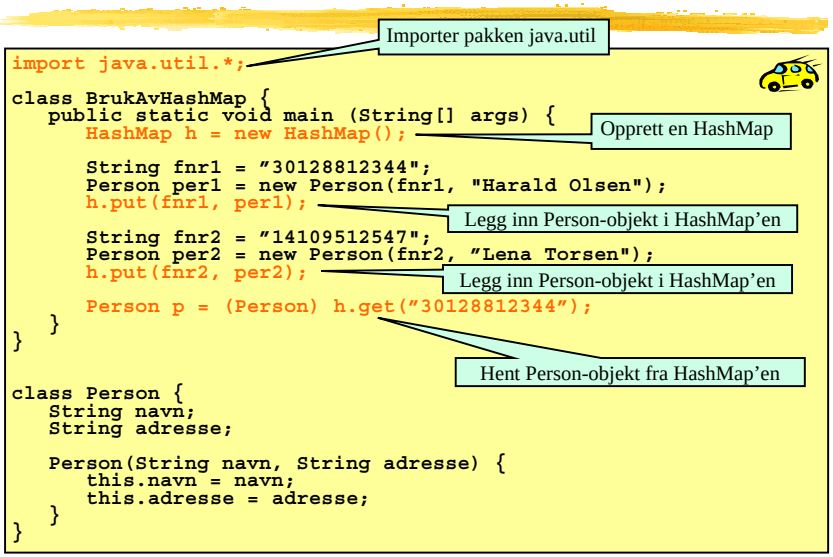
- et objekt i problemdomenet:
  En konto, en bil, en student, et lån, en eiendom
  (brukes til å definere en datamodell – en "modell av virkeligheten")

- et objekt av mer programteknisk art:
  Et skjermvindu, en fil, en tekststreng, en tabell
  (slike objekter er vanligvis ikke del av datamodellen – det er bare en effektiv måte å gruppere sammen programelementer som hører sammen)

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.

Eksempel på bruk av HashMap




```
import java.util.*;

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

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

      String fnr2 = "14109512547";
      Person per2 = new Person(fnr2, "Lena Torsen");
      h.put(fnr2, per2);

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

class Person {
   String navn;
   String adresse;

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


Opprette en HashMap

- 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:

```
HashMap h = new HashMap();
```

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

- 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

- For å hente et objekt med utgangspunkt i nøkkelen:

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

- Legg merke til at vi i starten må skrive i parentes navnet på klassen som objektet tilhører - i dette tilfellet klassen Person.
- Årsaken er at 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 INF1010).
- Merk: å hente et objekt fra en HashMap slik som over medfører *ikke* at objektet fjernes fra HashMap'en.

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`.