INF1000 – Uke 10

Mer om metoder og klasser

Oversikt

- En del repetisjon
- Noen generelle råd
- Gjennomgang av et eksempel
- Kunstig intelligens?

INF1000-seminar

- Lørdag 3 november kl. 11-17
  (Førstkommende lørdag!)
- Tema: Hjelp til oblig 4
- Billetter kan kjøpes i termstua Abel (30,-)

- 11:15 - Forelesning og tips til oblig 4 her på Sophus
  Lie av foreleser Fredrik eller Arne
- 12:15 - Orakeltjeneste på termstuer
- 14:00 - Pizzaspising
  (De 30,- som billetten koster dekker pizza)
- 15:00 - Tilbake til orakeltjeneste.
- 17:00 – Ferdig

Neste gang - Gjennomgang av eksamensoppgave

- PRØVEEKSAMEN i INF1000
  fra 23. november 2004

- Se på den til neste uke
  - Finn ut hva som ser vanskelig ut og forbered spørsmål

- Løs så mange gamle eksamensoppgaver dere rekker før eksamen!

Feil her?

```
int i = 4;
double pi = 3.14159;
pi = i;
```

- Nei, det går bra. Det kan alltids lages et flyttall av et heltall.
- Det er bare å ”legge på .000 · · · ” (Det er ikke akkurat slik flyttall representeres, men de lagres som heltalls- og flyttallsdel.)
- Altså 4 => 4.000

Men, dette er ikke lov

```
int k = Math.ceil(3.14);
```

- Math.ceil er en funksjon som returnerer en float (4.000 · · · i eksemplet).
- Vi kan ikke tilordne en float til en int for det vil generelt bety tap av presisjon
- Vi kan likevel skrive

```
int k = (int) Math.ceil(3.14);
```


Operasjoner utføres fra venstre

```
System.out.println("3 + 2 = " 3 + 2);
```

- Det blir **IKKE**:
  `3 + 2 = 5`
- Det regnes ut fra venstre mot høyre slik:
  `"3 + 2 = " 3 + 2`
  `"3 + 2 = 3"  + 2`
  `"3 + 2 = 32"`

Divisjon med 0

- Kompilatoren vil ikke forutse forsøk på å dele på 0.
- Det oppdages når maskinen utfører operasjonen og det er faktisk i hardware at det oppstår et såkalt unntak, som maskinen overlater til Java å løse. Java løser det ved å skrive ut en feilmelding og avslutte programmet. Det finnes mekanismer i Java for at programmereren kan håndtere det selv. Såkalt unntakshåndtering
- Dette vil altså kompilatoren ikke reagere på:
  ```
  int a = 7; int b=0;
  int c = a/b;
  ```
- Faktisk heller ikke dette:
  ```
  int b = 7;
  int c = b/0;
  ```

Skop

- Husk at dette ikke vil kompilere:

```
int b=0;
while(true){
  a++; b++;
  int a = b;
  System.out.println("a=" + a);
}
```

a er kun synlig herfra og til }

- Og heller ikke dette:

```
if(true){
  int z = 0;
}
System.out.println("z=" + z);
```

a er kun synlig herfra og til }

Lokale variable i metoder

- Vi har klassen:

```
class C {
  void skrivUt(int i) {
    i++;
    System.out.println ("i=" + i + " ");
  }
}
```

- Anta flg . kode i main:

```
int i = 7;
C cc = new C();
cc.skrivUt(i);
cc.skrivUt(i);
```


Resultat

```
$ java Lokale
i=8
i=8
$
```


Men, arrayer er referanser

- Vi har klassen:

```
class A{
  void skrivUt2(int [] a) {
    a[0]++;
    System.out.println("a[0]=" + a[0] + " ");
  }
}
```

- Anta flg . kode i main:

```
int [] b = {7,23};
A a = new A();
a.skrivUt2(b);
a.skrivUt2(b);
```

Resultat

```
$ java Lokale
a[0]=8
a[0]=9
$
```


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() {
    ...
  }
}
```


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 prikknotasjon:

```
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 konstruktører 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 parametere 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 parametere når programmet kompileres.
- Dermed er følgende to klassedeklarasjoner ekvivalente:

```
class Sirkel {
  double radius;
}
```

```
class Sirkel {
  double radius;

  Sirkel() {  }
}
```

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

Oppgave: virker dette?

- Lar dette programmet seg kompilere og kjøre?

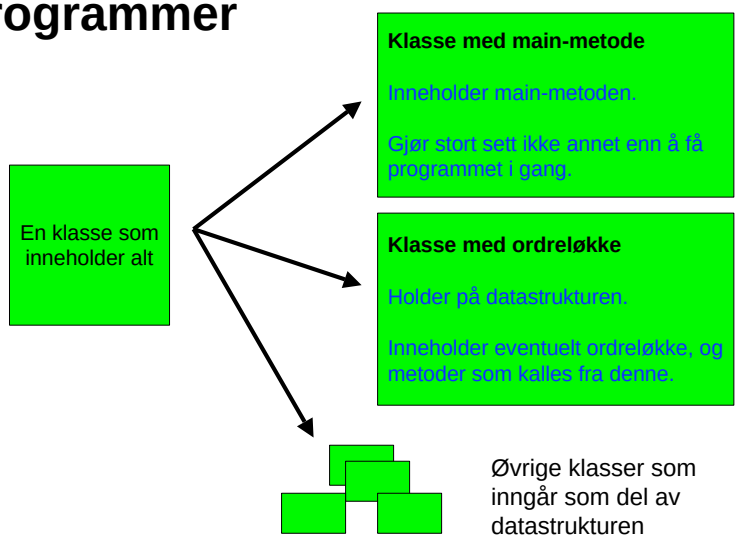
```
class OppgaveBil {
  public static void main (String [] args) {
    Bil b = new Bil();
  }
}

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


Resultat

```
$ javac OppgaveBil.java
OppgaveBil.java:3: cannot find symbol
symbol  : constructor Bil()
location: class Bil
        Bil b = new Bil();
                ^
1 error
$
```

Rollefordeling i større programmer



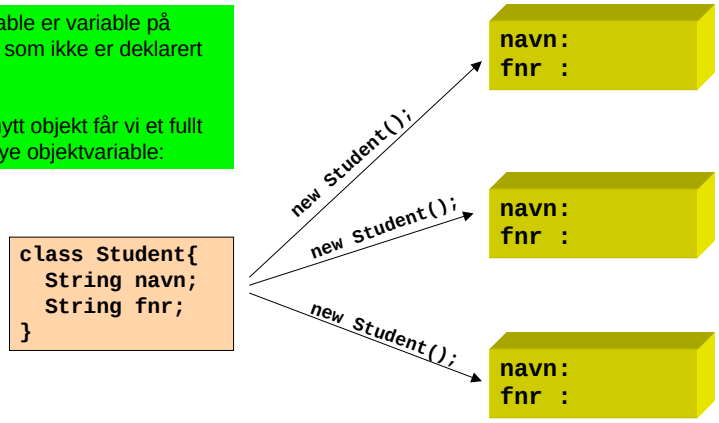
Klassevariable og objektvariable

- 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.

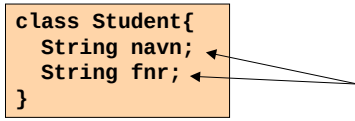
Objektvariable

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



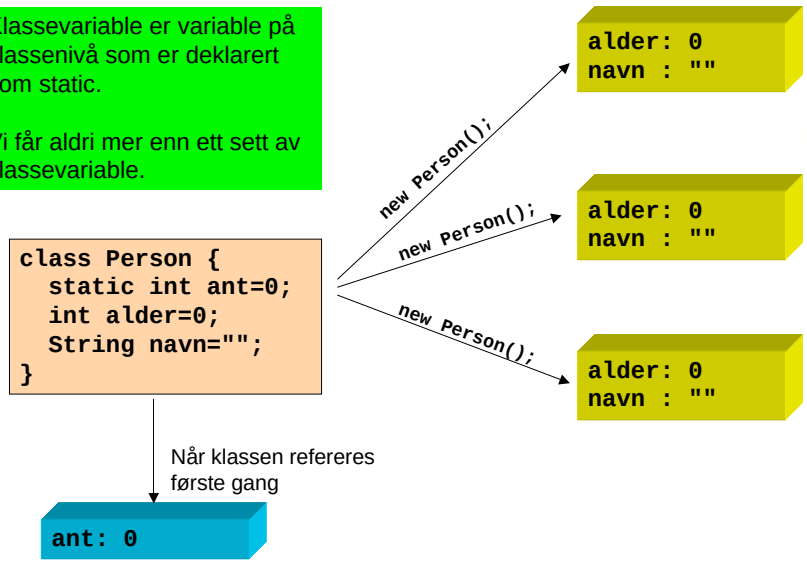
Disse variablene blir deklarert når vi lager et objekt av klassen ved å skrive **`new Student()`** og de lever så lenge objektet lever.

- 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 ant = 0;
  int alder;
  String navn;

  Person(){ ant ++; }
}
```

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.ant ....
```


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 - Hva skriver programmet ut på skjermen?

```
class Studentregister {
  public static void main (String [] args) {
    Student s1 = new Student();
    s1.init("Torjus", "S25332");
    Student 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);
  }
}
```

Restultat

```
$ javac Studentregister.java
$ java Studentregister
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

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!

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 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).

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: Det er alltid lov å gå tilbake å endre på noe!

- Programmer blir til ved at vi jobber litt her og der.
- Vi finner ofte ut at vi trenger flere klasser, eller at en klasse bare er "i veien" og fjerner den
- Det er ingen skam å snu. Det endelige programmet kan ha andre klasser og metoder enn vi startet med
- Pass likevel på å holde programmet kompilerbart og å heller ha "tomme skall" av alle metoder som kalles enn å ikke ha de der.

Eksempel: Flyreservasjon

Klasser

Egenskaper

Prosedyrer

- Vi skal lage et system for et flyselskap
- 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
- Oppgavene systemet skal løse er å lese inn en beskrivelse av alle flyene, med antall seter, klasser på de forskjellig seteradene, osv
- Så skal man kunne reservere seter, avbestille og skrive ut en oversikt over flyets seter, med klasse og om det er ledig eller ikke

Eksempel: Flyreservasjon

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

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();
  }
}
```


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

- Programmere ordreløkken
  - For hver kommando som skal utføres, skal ordreløkken kalle på en passende metode i klassen Flyreservasjon.
  - 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 visFlyvning(), så deklarerer du samtidig denne ”dummy-metoden” i klassen Flyreservasjon:

```
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.

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();
  }
}
```

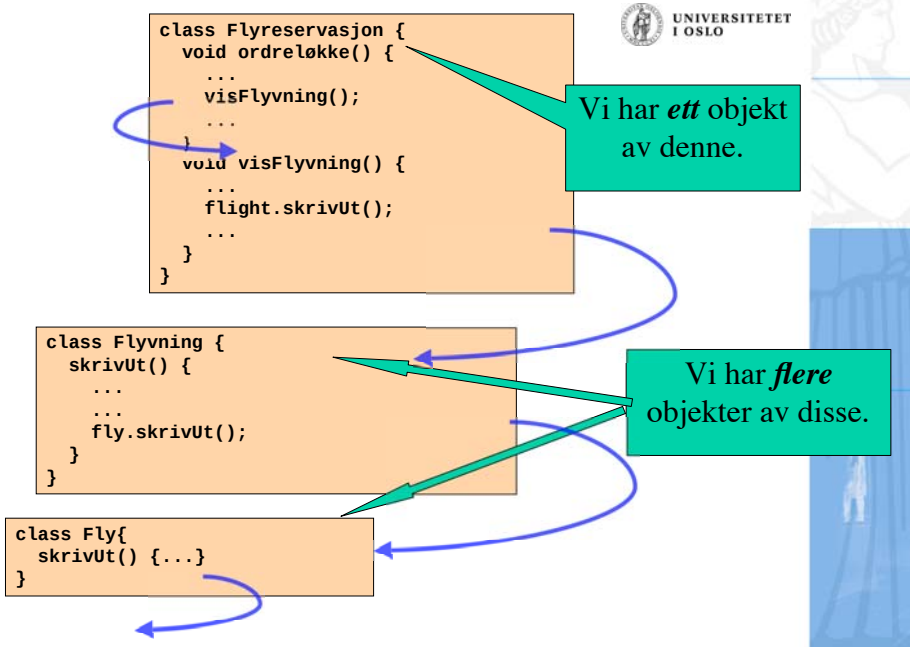
Oppdraget delegeres videre til en metode i Fly-objektet som er aktuelt.

Fly

- Skriver ut informsjon on flyet og delegerer videre til seteradene, som igjen 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



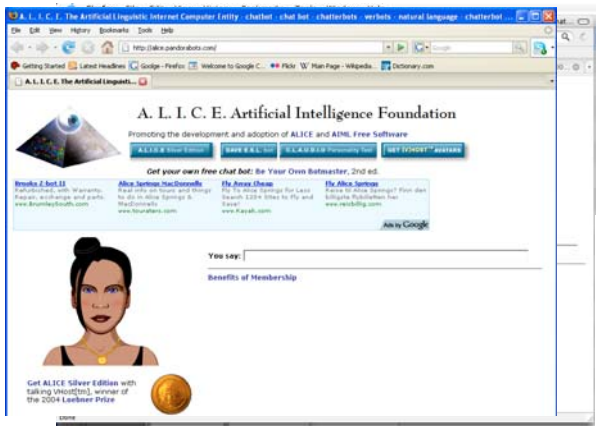
ELIZA – den elektroniske psykiater

- 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 I look like a calculator? 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 (eventuelt et standardsvar 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("Antallordlest: " +
                     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. Fortellmer.");
      }
      if (tast.hasNext()) {
        tast.readLine();  // Tømmer input-bufferet
      }
    }
  }
}
```


UNIVERSITETET I OSLO

Du finner hele programmet Eliza.java på nettsidene til kurset sammen med fila ord.txt.

Det kan hende det bare virker med forrige versjon av easyIO. Den som ligger på kurssiden for høsten 2006.