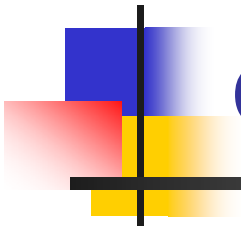
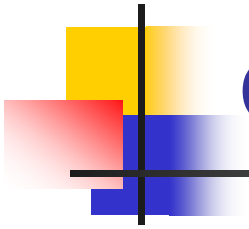


Inf1000 (Uke 7)
Objekter, klasser og pekere

Grunnkurs i programmering
Institutt for Informatikk
Universitetet i Oslo

Anja Bråthen Kristoffersen og Are Magnus Bruaset



Oversikt

- Repetisjon
  - Variable deklarasjoner, tilordnings-setningen
  - Løkker (for, while, do)
  - valgsetninger (if .. else, switch case)
  - arrayer (array-peker og array-objekt)
  - metoder
- Objekter og klasser
  - i verden
  - i programmet, hvorfor og hvordan
- Pekere, og hvordan lage objekter fra klasser
- Programeksempler med Objekt Orientert programmering
- Static

Variable, deklarasjon og tilordning

- En variabel er en navngitt plass i lageret som inneholder en verdi av en viss type.
- Variable deklareres og får da sitt navn og type. (+ mulig startverdi)

```
double rente;
int i = 4, j = i+1;

rente = 2.5;
rente = rente - 0.5;
```

- Tilordningssetningen gir en 'ny' verdi til en variabel – den gamle verdien overskrives

Løkker – gjør setninger flere ganger

a)
```
int[] primTall = {2,3,5,7};
int sum1 = 0;
for (int i = 0; i < primTall.length; i++) {
    sum1 = sum1 + primTall[i];
}
```

b)
```
int sum2 = 0;
for (int tall : primTall) {
    sum2 = sum2 + tall;
}
```

c)
```
int sum3 = 0, i = -1;
while (++i < primTall.length) {
    sum3 += primTall[i];
}
```

d)
```
int sum4 = 0, i = 0;
do {
    sum4 += primTall[i];
} while (++i < primTall.length );
```

for-løkke med lokal løkkevariabel 'i'

for-løkke med lokal løkkevariabel 'tall'

while-løkke

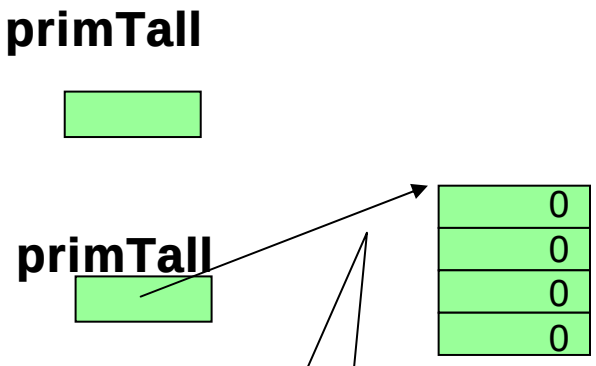
do-while løkke går minst en gang (feiler hvis length = 0)

array

- En array betår av to deler
  - arraypekeren
  - arrayobjektet

```
int[] primTall ;

primTall= new int[4];

primTall[0] = 2;
primTall[1] = 3;
primTall[2] = 5;
primTall[3] = 7;
```

tilordningen primTall = new.. plasserer adressen til arrayobjektet i primTall

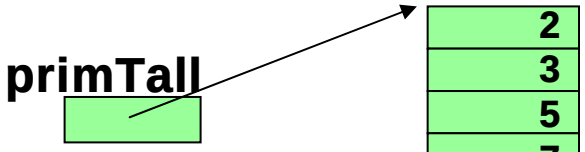


new lager array-objektet og returnerer adressen

array

Her lager vi en peker som kan peke på et objekt av typen int[]

```
int[] primTall;

primTall= new int[4];

primTall[0] = 2;
primTall[1] = 3;
primTall[2] = 5;
primTall[3] = 7;
```



Her lager vi arrayobjektet primTall skal peke på, og definerer størrelsen på arrayobjektet

primTall[1] = 3;
betyr at plassen hvor primTall 'peker' + 1 får verdien '3'

Programmet kan lese og skrive alle heltalls variablene i arrayobjektet ved å starte der primTall peker, og så evt. legge til 0,1,2 eller 3 (= length -1)

```
import easyIO.*;

class ArrayTest{
  public static void main(String args[]) {
    In tastatur = new In();
    Out skjerm = new Out();
    double s1 =0.0,s2=0.0;

    // registrer temeraturen 2 ganger om dagen i en uke
    double[][] temp = new double[7][2];
    String[] ukedag ={"mandag", "tirsdag", "onsdag", "torsdag",
                      "fredag", "lørdag", "søndag"};

    for(int i = 0; i < 7; i++) {
        skjerm.out("Gi morgen-temp for " +ukedag[i] +" :");
        temp[i][0] = tastatur.inDouble();
        skjerm.out("Gi kvelds-temp for " + ukedag[i] +" :");
        temp[i][1] = tastatur.inDouble();
    }
    for(int i = 0;i<7;i++) {
        s1 += temp[i][0];
        s2 += temp[i][1];
    }
    skjerm.outln(" Gjennomsnitt morgen-temp = " + s1/7);
    skjerm.outln(" Gjennomsnitt kvelds-temp = " + s2/7);
   }
}
```

Valgsetninger – if .. else og switch/case

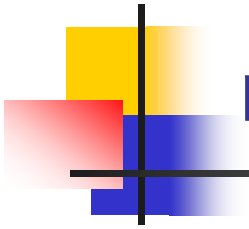
```
Out skjerm = new Out();
In tastatur = new In();
int alder = tastatur.inInt();

if (alder > 15)
        skjerm.out("Voksenbillett”);
else
        skjerm.out("Barnebillett");
```

Hadde vi hatt flere setninger, måtte vi brukt {..} rundt disse

```
import easyIO.*;

class Utskrift{
  static void  meny (Out ut) {
         ut.outln(" Velg:");
         ut.outln("   1 - les data");
         ut.outln("   2 - skriv ut");
         ut.outln("   3 - avslutt");
  }

  public static void  main ( String[] args)  {
    int alder = 0, valg;
    Out skjerm = new Out();
    In tastatur = new In()

    do{
      meny(skjerm);
      valg = tastatur.inInt();
      switch(valg) {
         case 1:   // les data
              skjerm.outln("Skriv inn din alder: ");
              alder = tastatur.inInt();
              break;
         case 2:   // skriv data
              skjerm.outln("Alder = " + alder);
              break;
         case 3:   // avslutt
              skjerm.outln("Systemet avslutter");
              break;
         default:  //  feil
              skjerm.outln("Du valgte " + valg + " gyldige valg er: 1 - 3");
      }
    } while (valg != 3);
  }
}
```

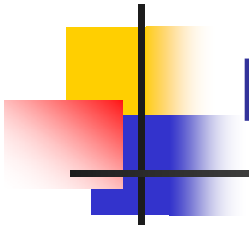
kall på metoden ’meny(..)’ som skriver ut valgmulighetene

switch /case – setning. velger mellom 1,2,3 og ’gal verdi’



metoder

- Brukes til å dele opp programmet
  - i så små deler at de er ’lette’ å programmere riktig
  - vi velger selv hvilke setninger vi legger i en metode
- En metode gis et navn vi selv velger
- Noen metoder skal returnere et svar
  - Der setter vi typen til svaret (int, double, int [],..) foran metodenavnet
- Noen metoder ’bare gjør noe’
  - da setter vi ’void’ foran metodenavnet



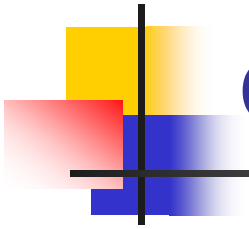
kall av metoder

- Kall av en metode er:
  - At vi skriver navnet på en metode som (del av) en setning i en annen metode (f.eks i 'main')
  - Når vi gjør det, 'hopper' programutførelsen til starten på den kalte metoden.
  - Når den kalte metoden er ferdig (returnerer), 'hopper' programutførelsen tilbake til rett etter der metoden ble kallet.
- Metoder kan ha parametere (navngis i parentes bak metodenavnet)
  - disse oppfører seg som lokale variable i metoden
  - når vi kaller metoden, får vi ***kopier*** av verdiene som ble brukte i kallet på parameterplassene før man 'hopper' til den kalte metoden (se neste foil).
- Når vi lærer om klasser, skal vi se at vi ofte ikke behøver å ha **`static`** foran metoden (static= klassemetode)

```
import easyIO.*;
class MetodeTest {
  static int leggSammen(int a, int b) {
          int svar = a + b;
          return svar;
  }

  static void skrivTegn(Out ut, int ant, char c) {
          for (int i = 0; i < ant; i++){
            ut.out(c);
          }
          ut.outln();
   }

  static void skrivAdvarsel(Out u, String s) {
           skrivTegn(u, s.length() + 12, '*');
           u.outln("Advarsel:" + s + "!!!!");
           skrivTegn(u, s.length() + 12, '*');
  }

  public static void  main ( String[] args)  {
   int i=14, abc = 22, c;
   Out skjerm = new Out();

   c = leggSammen(i,abc);
   skjerm.outln("svaret er:" + c);
   skjerm.outln("nytt svar:" + leggSammen(c,abc));
   skjerm.outln("enda ett svar:" + leggSammen(i+c,i));
   String x = "" + c;
   skrivAdvarsel(skjerm, x);
 }
}
```

3 kall på leggSammen(..)

kall på skrivAdvarsel(..) som igjen kaller skrivTegn(..)

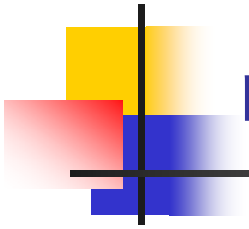


Objekter

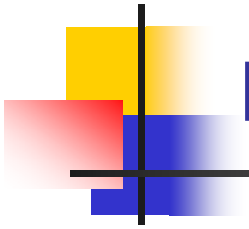
- Vi deler verden inn i enheter for å bedre kunne tenke på , snakke om og forstå verden.

- Eksempler på slike enheter
  - Levende organismer
  - Planter
  - Blomst
  - Trær
  - Mennesker
  - Studenter
  - Jenter
  - Gutter
  - Voksne
  - Barn

Verden består av mange objekter, noen ganske like, noen ulike



- Ser vi rundt oss i auditoriet, ser vi
  - Studenter, stoler, murstein, lysarmatur, blyanter,...
- Vi ser at når vi betrakter verden, deler vi den opp i et passe antall enheter/deler som vi har egne navn for.
- De enheter vi deler programmeringsverden inn i, kalles ***objekter***
- Mange av objektene i verden er av samme type. Vi kan skille dem fra hverandre med f.eks. ulike navn
  - Studentene Kai og Espen
  - to bøker med ulik tittel/forfatter
- Objekter som kan ses på som like sier vi at tilhører samme ***klasse***
  - De beskrives av samme sett variable, men har *ulike* verdier på noen av variablene (to biler av samme bilmerke og farge men med ulike registreringsnummer)

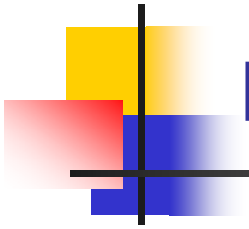


Klasser og objekter i verden

- To objekter kan også være helt forskjellige (f.eks et tre og en lastebil)
  - De er da to objekter av hver sin klasse (klassen Tre og klassen Lastbil)
- Hva vi velger å betrakte som et objekt, og hvilke klasser vi bruker for å beskrive en problemstilling, er ikke bestemt på forhånd. Innenfor vide rammer bestemmer vi det selv.

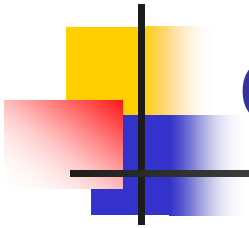
Som Objekt orientert programmerer, hevder jeg :

- Klasser er generelle begreper som egentlig ikke eksisterer i verden – det som eksisterer er objektene.
- Klasser er menneskenes måte å beskrive og strukturere verden, ikke gitt en gang for alle og vi kan godt lage oss nye begreper.



Hvor mange klasser (og objekter) er det ?

- Hvilke klasser vi bruker til å beskrive et problem, varierer ofte etter hvor detaljert vi betrakter en problemstilling og hvilke spørsmål vi ønsker å kunne gi svar på:
  - Beskriver vi problemet med veitrafikk og køer på veiene, er vi neppe interessert i mer enn å telle antall biler og kanskje skille mellom lastebiler, busser og personbiler.
  - Beskriver vi problemet til en bilfabrikk, trenger vi en meget detaljert og komplisert beskrivelse av hver bil (bestående av motor, hjul, karosseri, lys,..., hver beskrevet med sin klasse) og mange ulike typer av biler. Vi se da en rekke klasser som hver beskriver sine objekter.



Objektorientert Programmering

Når vi betrakter et problem vi skal lage et datasystem for, gjør vi to avgrensninger:

1. Vi ser bare på ***en del av verden*** (vårt problemområde)
2. Innenfor problemområdet betrakter og beskriver vi bare det som er der med ***en viss detaljeringsgrad*** - bare så mange detaljer vi trenger for å svare på de spørsmål datasystemet skal kunne gi svar på.

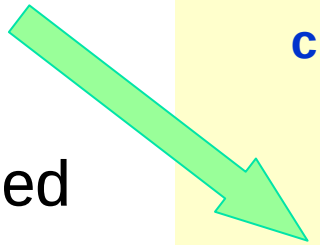
Eks: Hvordan beskrive en student ? Skal vi lage:

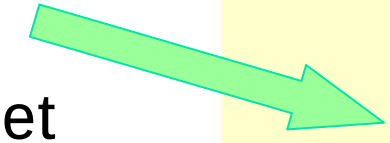
a) Et Studentregister, registrerer vi bare navn, personnummer, adresse, tidligere utdannelse og kurs (avlagte og kurs vedkommende tar nå)

b) Et legesystem for studenter, ville vi ta med svært mange opplysninger om hver student (medisiner, sykdommer, resultat fra blodprøver, vekt…) som vi ikke ville drømme om å ha i et vanlig studentregister

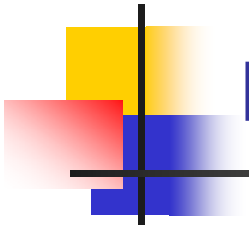
Hvordan lage klasser og objekter i et program.

- Klasser deklarerer vi med **`class`**
- Vi lager pekere til objekter ved å deklarere dem med klassenavnet
- Vi lager et objekt med å si **`new`** foran et klassenavn
- Forholdet mellom et objekt og en peker er som en array-peker og et array-objekt





```
class Student {
   String navn, adresse;
}

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

    Student s1, s2;

    s1 = new Student();
    s2 = new Student();

   }
}
```



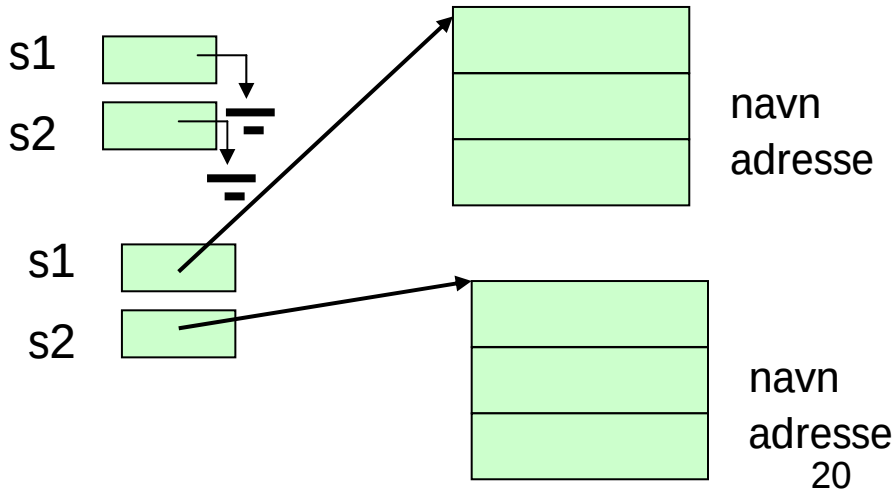
Hva er et objekt i programmet?

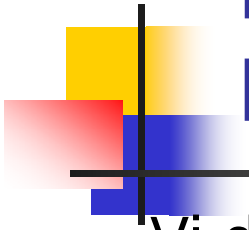
- Et objekt er et område i lageret som inneholder *en kopi* av alle de metodene og variable i en klasse det ikke står **`static`** foran
- Klassene er en slags mal/form/oppskrift som vi kan lage objekter med.
- Vi kan lage så mange objekter av en klasse som vi vil.
- De variable og metode det ikke står **`static`** foran, kalles *objekt-variable og objektmetoder*
- De variable og metoder det står **`static`** foran, kalles *klasse-metoder og klassevariable*, og blir ikke med i objektene (men ligger lagret i bare ett eksemplar et annet sted)

Peker og objekter

```
class Student {
   String navn, adresse;
}

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

    Student s1,
            s2;

    s1 = new Student();
    s2 = new Student();

   }
}
```

- En peker inneholder adressen til hvor et objekt ligger i lageret
- eller den inneholder 'null' (= ikke-objekt)
- Vi tegner adressen som en 'pil' (peker)





Programmer i Java består av en eller flere klasser

- Vi deler opp programmet vårt i flere klasser
  - fordi hver programdel (klasse) skal være mulig å holde oversikt over – ikke for stor.
  - fordi en klasse skal være en god modell på en del av problemet vi lager program for.
    - Anta at vi hadde et datasystem som omhandlet kurs og studenter. Da ville vi ha en klasse Student og en klasse Kurs i programmet.
- En klasse inneholder
  - deklarasjon av null eller flere variable
    - som beskriver *ett eksemplar* av det klassen er modell av
  - null eller flere metoder
- En klasse representer et generelt begrep som:
    - Eksempler kan være: Student, Kurs, ...

Objekter og pekere, og hvordan få adgang til innmaten av et objekt (.)

- Når vi har laget et objekt med **new**, har vi altså fått en kopi av objekt-variablene og objekt-metodene, men hvordan får tak i dem?
- Vi bruker operatoren . (punktum).
  - Foran punktumet har vi navnet på en peker til et objekt.
  - Etter punktum har vi navnet på en variabel eller metode inne i objektet –
  - Punktumet leses som 'sin' eller 'sitt'
    - eks: La s1 peke på et Student-objekt.

```
s1.navn = "Ola N";
```

```
class Student {
   String navn, adresse;

   void skrivUt() {
     System.out.println("Student med navn:" + navn +
                        ", adr:" + adresse);
   }
}

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

    Student s1, s2;

    s1 = new Student();
    s1.navn = "Ola N";
    s1.adresse = "Storgt. 12, 1415 Nordby";
    s2 = new Student();
    s2.navn = "Åsne S";
    s2.adresse ="bokhandelen i Kabul";
    s1.skrivUt();
    s2.skrivUt();
   }
}
```

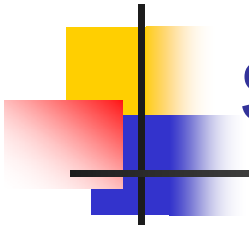
```
class Kurs {
   String kurskode;
   int studiepoeng;

   void skrivUt() {
     System.out.println("Kurs med kode:"
         + kurskode+ ", og stp:" + studiepoeng);
   }
}

class KursRegister {
  public static void main(String args []) {

    Kurs inf, mat;

    inf = new Kurs();
    inf.kurskode = "INF1000";
    inf.studiepoeng = 10;
    inf.skrivUt();

    mat = new Kurs();
    mat.kurskode = "MAT1000";
    mat.skrivUt();

   }
}
```

objektvariable

Objekt -metode

Klasse - metode

Lager to objekter av klassen Kurs



Stringer er objekter

- String er en klasse i Java-biblioteket, men har en egen spesiell syntaks (skrivemåte) så det ser ut som den er en av de basale typene (som int, double,..).
- Når vi har en string, har vi både en peker (den vi deklarerer navnet på) og et string-objekt.
- Egen skrivemåte for stringkonstanter:

```
String s = ”En fin dag i mai”;
```

Er det samme som:

```
String s = new String(”En fin dag i mai”);
```

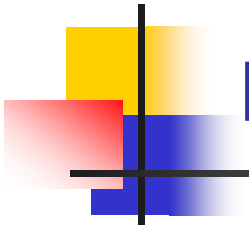
- Klassen String inneholder mer enn 50 metoder for konvertering mellom ulike datatyper og tekst, samt tekstsøking.

Klassen String

```
String tekst = ”kake”;
String t = ”ake”;


int lengde = tekst.length();
boolean b = tekst.equals(t);
char c = tekst.charAt(1);
String t = tekst.replace(’k’, ’r’);
String delstreng = tekst.substring(1);
String store = tekst.toUpperCase();
String smaa = tekst.toLowerCase();
int k = tekst.compareTo(t);
int k = tekst.indexOf(t);
boolean b = tekst.startsWith(”K”);
boolean b = tekst.endsWith(”e”);
```

Alle metodene vi har sett på for String begynte med `NavnPåString.`

Dette fordi String er en klasse. Når vi skriver String s = ”Hei”; lager vi en peker s som peker på et objekt av klassen String. Der pekeren peker i minne blir teksten ”Hei” lagret.



Klasse-variabel (=statisk variabel)

- Setter vi static foran en variabel, er det er bare **én** felles variabel med det navnet for alle objektene.
- Setter vi **`static`** foran en metode, har den bare utsikt til :
  - sine egne lokale variable og parametere
  - andre statiske variable og metoder
  - klassenavnene
- Statiske metoder og variable kan man få adgang til både
  - via klassenavnet og punktum
  - via peker til et objekt av klassen og punktum

```
class B {
        static int i = 0;
        double x = 0.0;
}
class A {
        int k;

    public static void  main ( String[] args)  {
        B b1 = new B();
        B b2 = new B();
        System.out.println("b1.i :"+ b1.i+", b2.i:" + b2.i);
        b1.i = 4; //her endres b1 sin klassevariabelen (felles).
        System.out.println("b1.i :"+ b1.i+", b2.i:" + b2.i);
        System.out.println("b1.x :"+ b1.x+", b2.x:" + b2.x);
        b1.x = 2; //her endres b1 sin objektvariabel (en hver).
        System.out.println("b1.x :"+ b1.x+", b2.x:" + b2.x);
   }
}
```

```
class A2
{        int k;  // objektvariabel 'k'
    public static void  main ( String[] args)  {
              k = 1;
 }}
```

```
>javac A2.java
a2.java:6: non-static variable k cannot be referenced from a static context
          k = 1;
          ^
1 error
```

```
class A2
{      int k;

    public static void  main ( String[] args)  {
              A2 aa = new A2();
              aa.k = 1;
       }
}
```

```
>javac A2.java


>
```

Arrayer av pekere til objekter

- Vi kan lage arrayer av pekere til objekter (men ikke av objektene direkte) omlag på samme måte som vi lager arrayer av int, double og String
- Har vi deklarert klassen **`Kurs{...}`** kan vi lage array som følger:

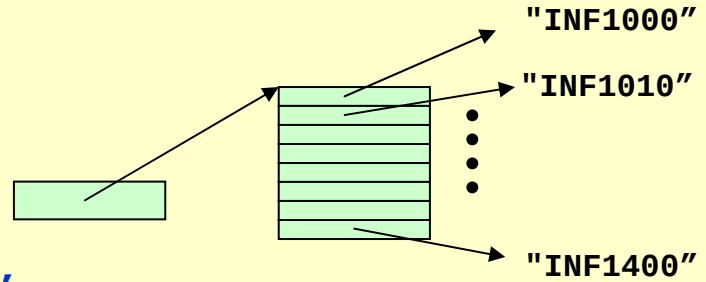
```
Kurs[] ifiKurs;

ifiKurs = new Kurs[120];

ifiKurs[0] = new Kurs();
```

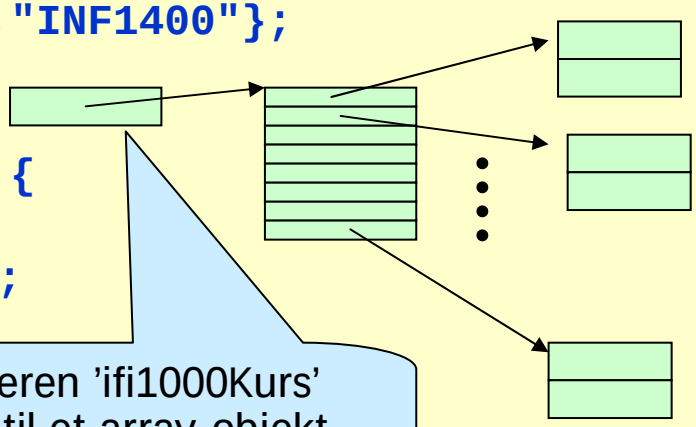
Her deklareres 'bare array-pekeren

Her lages array-objektet med 120 'tomme' pekere

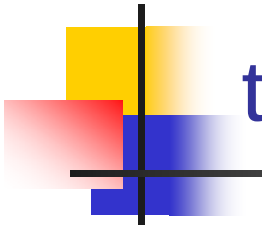
Her settes det første pekeren i array-objektet til å peke på et nytt Kurs-objekt

```
class Kurs {
   String kurskode;
   int studiepoeng = 10;

   void skrivUt() {
     System.out.println("Kurs med kode:"
         + kurskode + ", og stp:"
          + studiepoeng);
   }
}

class KursRegister2 {
  public static void main(String args []) {

    String[] kursKoder= {"INF1000","INF1010",
                         "INF1020","INF1040","INF1050",
                          "INF1060","INF1070","INF1400"};
    Kurs[] ifi1000Kurs = new Kurs[8];

    for(int i = 0; i < kursKoder.length; i++) {
       ifi1000Kurs[i] = new Kurs();
       ifi1000Kurs[i].kurskode = kursKoder[i];
       ifi1000Kurs[i].skrivUt();
    }
 }
}
```





arraypekeren ’ifi1000Kurs’
peker til et array-objekt
med 8 Kurs-pekere



this

- Ethvert objekt inneholder en peker til seg selv: this
- Noen ganger trenger vi å bruke this for å referere til objektvariablene:

```
class Person {
     String navn = ”Andre Bjerke”;
     String adr = ”Oslo”;
     void byttAdresse (String adr) {
       this.adr = adr;
     }
  }
```

parameteren adr

objektvariabelen adr

- Merk: vi kunne sluppet unna bruk av this ovenfor ved å kalle parameteren i metoden **`byttAdresse`** for noe annet (f.eks. **`String adresse`**) - men ofte er det mer oversiktlig å gjøre som over.

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:**

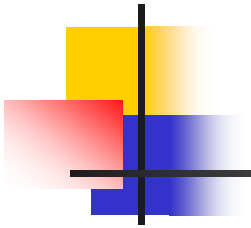
```
Person p = new Person();
p.navn = ”Petter”;
p.fnr = ”15108559879”;
```

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

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

- **Benytt en konstruktør:**

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



Konstruktører

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 overlasting av konstruktør

```
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;
   }
}
```

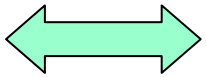
Konstruktør 1

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 parametere.

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;
   }
}
```

Pekerlikhet og innholdslikhet

```
class Likhet {
  public static void main (String [] args) {
    Student stud1 = new Student("Petter", "01018926324");
    Student stud2 = new Student("Petter", "01018926324");

    if (stud1 == stud2) {
       System.out.println("Samme objekt");
    }

    if (stud1.navn.equals(stud2.navn) && stud1.fnr.equals(stud2.fnr)) {
       System.out.println("Samme innhold");
    }
  }
}

class Student {
   String navn;
   String fnr;

   Student (String navn, String fnr) {
      this.navn = navn;
      this.fnr = fnr;
   }
}
```

enum: oppramsingstyper

- I java 1.5 er det kommet en ny type variabel som bare skiller mellom ulike navngitte verdier.
- Eksempel:

```
enum Hverdag {mandag, tirsdag, onsdag, torsdag, fredag}
Class EnumEks{
  public static void main(String[] args){
       Hverdag d = Hverdag.mandag
       System.out.println(”I dag er det” + d);
  }
}
```

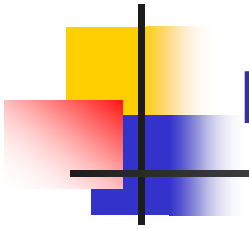

Eksempel: iterer over elementene i enum

```
enum Hverdag {mandag, tirsdag, onsdag, torsdag, fredag}
Class EnumEks2{
  public static void main(String[] args){
      for (Hverdag d: Hverdag.values()){
            System.out.println(”dag: ” + d +
                  ” ukedagnummer: ” + d.ordinal());
      }
  }
}
```

Metodene **`values()`** og **`ordinal()`** er forhåndslagde metoder i Java 1.5.
Metoden **`values()`** itererer over alle verdiene i oppramsningstypen.
Plasseringen det elementet vi ser på har i oppramsingen finner vi med metoden **`ordinal()`**.


Eksempel: verdi på elementene i enum

```
enum Hverdag {
  mandag(4), tirsdag(3), onsdag(2), torsdag(1), fredag(0);
  final int dagerIgjenTilHelgen;
    Hverdag (int v){
       dagerIgjenTilHelgen = v;
  }
}

class EnumEks3{
  public static void main(String[] args){
       for (Hverdag d: Hverdag.values()){
              System.out.println(”dag: ” + d +
              ”, antall dager igjen til helgen er: ”
              + d.dagerIgjenTilHelgen);
       }
  }
}
```



Hvorfor programmere med objekter?

- Med objekter kan vi ofte organisere dataene våre bedre.

Eksempel Personreigster:

- Uten Objekt Orientert programmering:

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

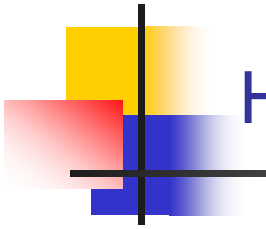
Informasjonen knyttet til en bestemt person er splittet opp i tre arrayer.

- Med Objekt Orientert programmering:

```
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.



Hvorfor programmere med objekter?

- Med objekter kan vi samle data og operasjoner på dem.

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

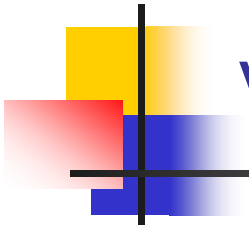
```
class Student {
  ... data om studenter ...
  ... student-metoder ...
}

class Kurs {
   ... data om kurs ...
   ... kurs-metoder ...
}
```

Her ligger alle data og alle metoder samlet ett sted

Metoder og data som hører sammen er samlet.

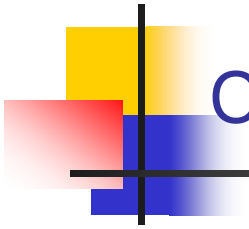
Lett å se hvilke metoder som jobber på hvilke data (strukturering av koden).

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



Viktige prinsipper

- En metode skal bare bruke objektvariable deklarert i egen klasse.
  - selv om man lett kan få tak i variable fra andre klasser, men bruk av variable fra andre klasser vil føre til rotete programmering hvor det senere blir vanskelig å endre på programmet. For eksempel slette en variabel (som viser seg å være brukt av en annen klasse).
- Hvis en metode viser seg vanskelig eventuelt lang å programmere del den i mindre metoder.
  - små enkle metoder som hovedsakelig kun gjør en oppgave (høyden noen få som klart hører sammen) gjør programmet lett lesbart for andre i ettertid.



Oppsummering om klasser, objekter, pekere og .

- Vi lager OO-programmer ved å lage en modell av problemområdet
  - ett objekt i verden gir ett tilsvarende Java-objekt i programmet
  - Objekter kan være av ulik type, og for hver slik type deklarerer vi en klasse
- Et Javaprogram består av en eller flere klasser
- Vi lager objekter fra klassen med new

```
s1 = new Student();
```

- Et objekt inneholder en kopi av alle ikke-statiske variable og metoder i klassen
- Vi får adgang (lese, skrive og kalle metoder) til det som er inni et objekt ved . operatoren:

```
s2.adresse ="bokhandelen i Kabul";
s1.skrivUt();
```