UNIVERSITETET I OSLO

Institutt for informatikk

IN1010 våren 2023
31. januar

Objektorientering i Java

Om innkapsling og enhetstesting
Mer om arrayer og noen klasser som kan ta vare på objekter
Om parameteroverføring

Stein Gjessing

Agenda

- Mer om hvordan representeres objekter i minnet
- Referansevariabler (pekere)
- Mer om objektorientert programmering
- Mer om objekter og innkapsling
  - Enhetstesting
- Mer om arrayer
- Om to beholdere i Javas bibliotek:
  - HashMap
  - ArrayList

Objektorientert programmering:

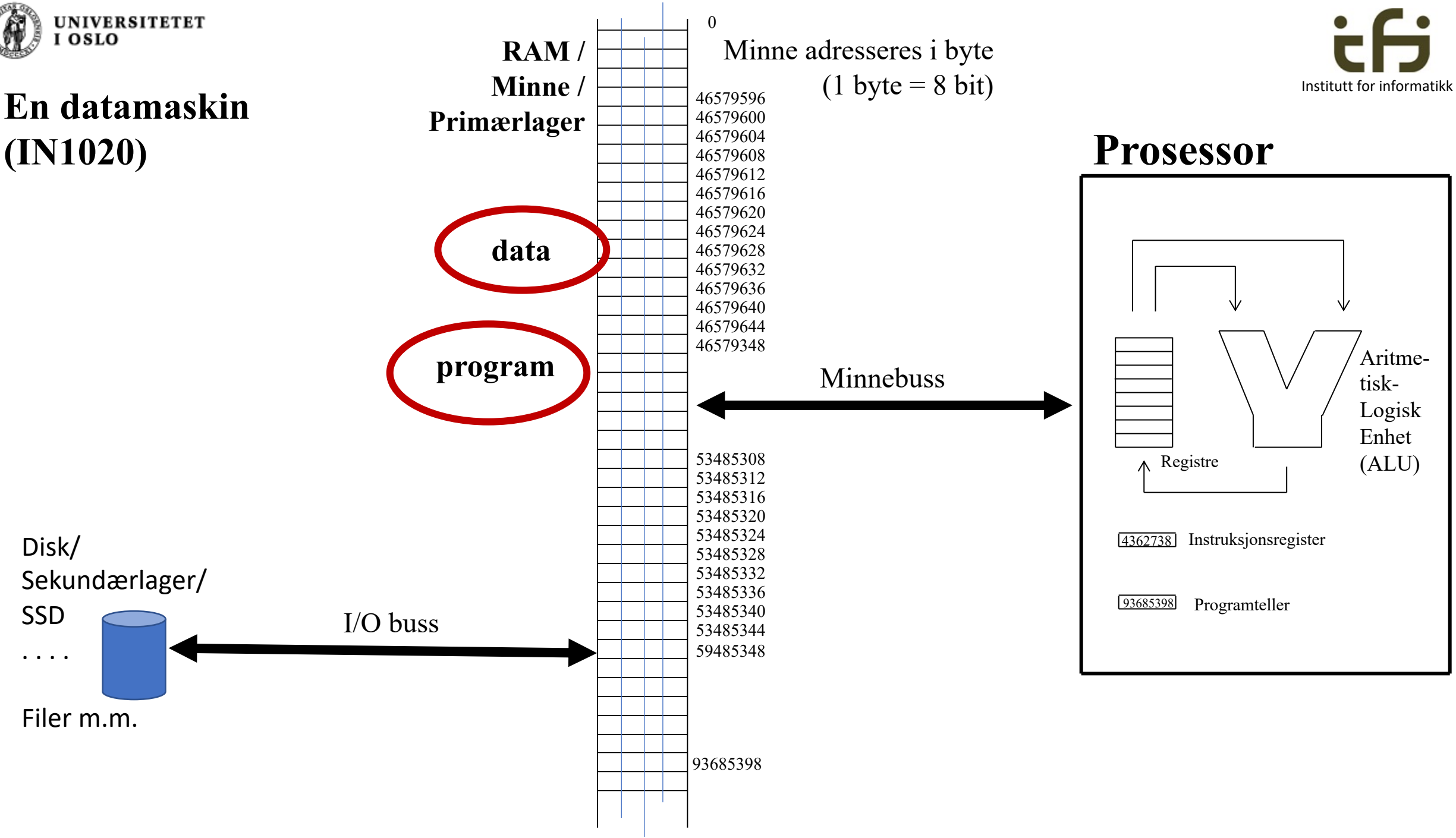
Å representere ting i det virkelige liv inne i datamaskinen

Hvordan ser en katt ut
inne i datamaskinen ?

UNIVERSITETET I OSLO

Institutt for informatikk

En datamaskin (IN1020)



Objektorientert programmering:

Å representere ting i det virkelige liv inne i datamaskinen

Hvordan ser en katt ut
inne i datamaskinen ?

Data: vekt, alder
Handlinger: spise, sove

UNIVERSITETET I OSLO

Institutt for informatikk

data: vekt
fødselsår

program: spise
sove

0
46579596
46579600
46579604
46579608
46579612
46579616
46579620
46579624
46579628
46579632
46579636
46579640
46579644
46579348
53485308
53485312
53485316
53485320
53485324
53485328
53485332
53485336
53485340
53485344
59485348
93685398

Minnebuss

CPU

Registre

Aritme-
tisk-
Logisk
Enhet
(ALU)

4362738 Instruksjonsregister

93685398 Programteller

Disk/
Sekundærlager/
SSD
. . . .

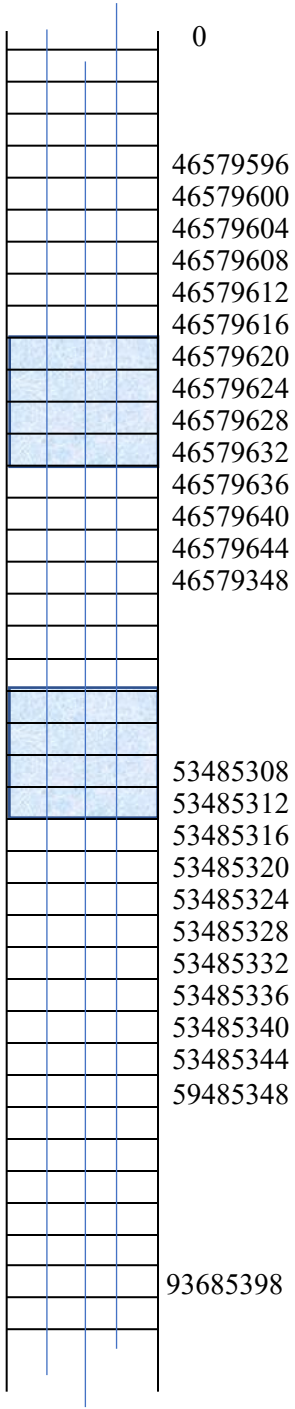
Filer m.m.

I/O buss

UNIVERSITETET I OSLO

Et objekt er en samling av data og handlinger

Institutt for informatikk



data: vekt, fødselsår

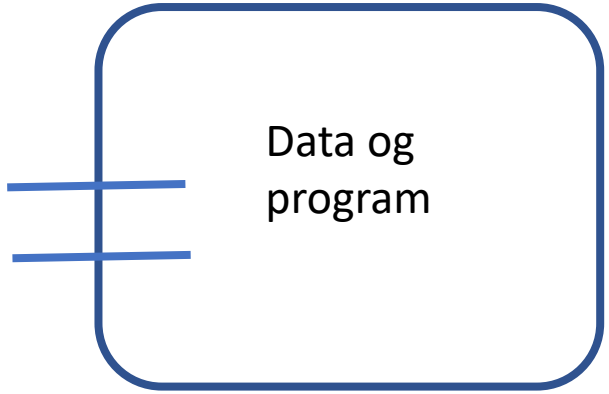
program: spise, mosjonere

Katt inne i datamaskinen

Katt i virkeligheten

Slik tenker vi oss et objekt:

Eller slik:

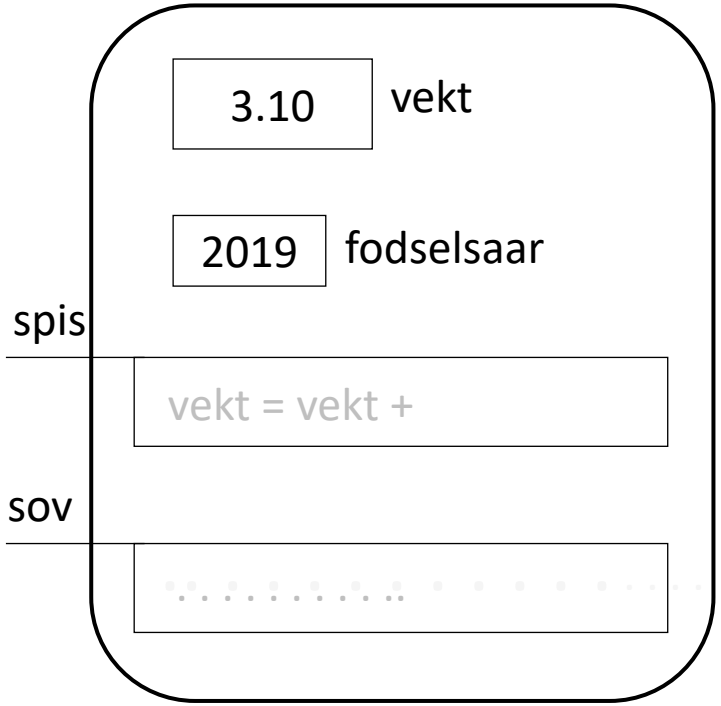


UNIVERSITETET I OSLO

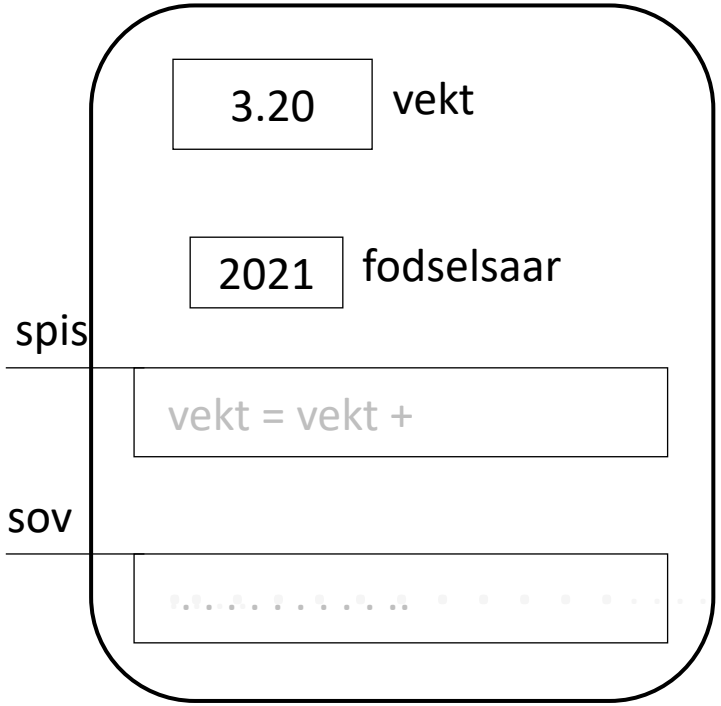
Ole-Johan Dahl og Kristen Nygaard fant på OBJEKTER inne i datamaskinen

Institutt for informatikk

Objekt som beskriver Mons



Objekt som beskriver Gråpus



Sist snakket vi om Javas primitive **verdier**

boolean	~~1~~ 00000001	8 bit	(Det er for upraktisk å lagre (sammen med andre data), pakke ut og inn og teste ett bit)
byte	01001110	8 bit	
char	0100111010110110	16 bit	
short	0100100101101110	16 bit	
int	01010010110110101000100101101110	32 bit	
long	0101001011010011010110100010101001010110100110101000100101101110	64 bit	
float	01010010110110101000100101101110	32 bit	
double	0101001011010011010110100010101001010110100110101000100101101110	64 bit	

Type: int Navn: tall

00000000000000000000000000001010

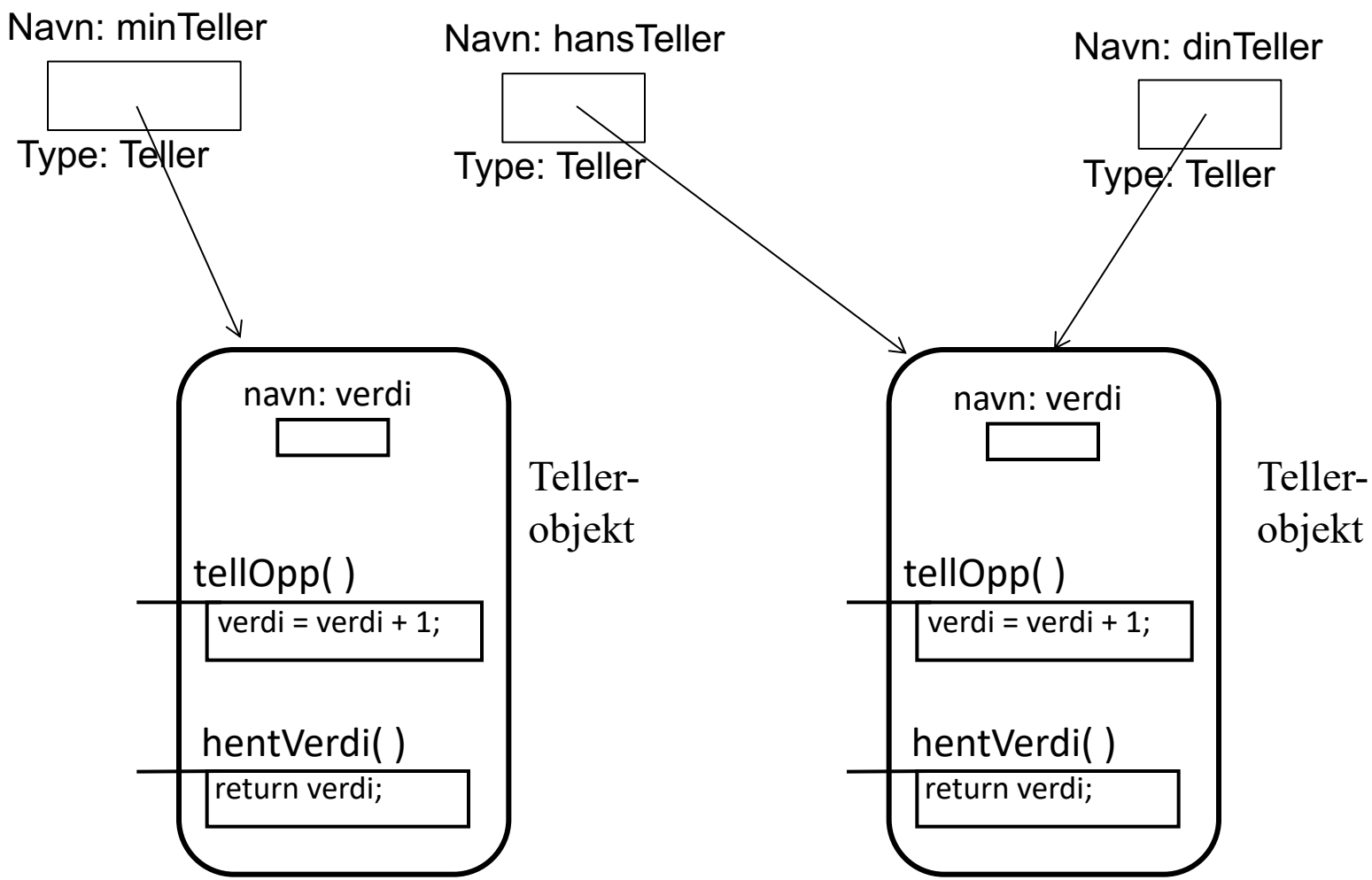
En **variabel** er et sted i minnet der det er lagret en slik **verdi**.

F.eks. deklarasjonen int tall = 10;

Java er **et sterkt typet språk:**

En variabel **har bare lov å inneholde en verdi av den deklarerte typen** ~~tall = 3.14~~

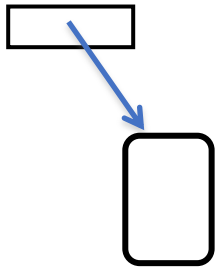
Nå skal vi snakke mer om referanseverdier og referansevariabler



Hva er en referanse / peker

Python:

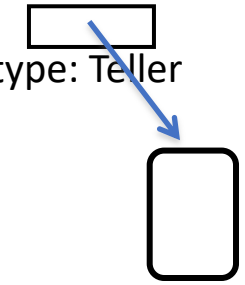
navn: minTeller



Java:

navn: minTeller

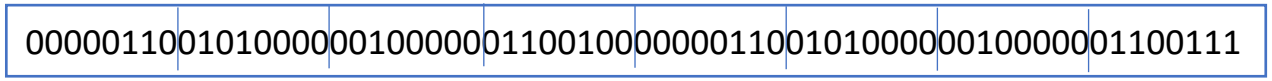
type: Teller



F.eks. deklarasjonen: `Teller minTeller = new Teller();`

En variabel: som innholder verdien:

navn: minTeller type: Teller



En variabel er et sted i minnet

00000110010100000010000001100100000011001010000001000000 1100111

En referanse(verdi) er en adresse til et sted i minnet der det ligger et objekt (eller en array)
I Java bruker vi *referanse* og *peker* synonymt.

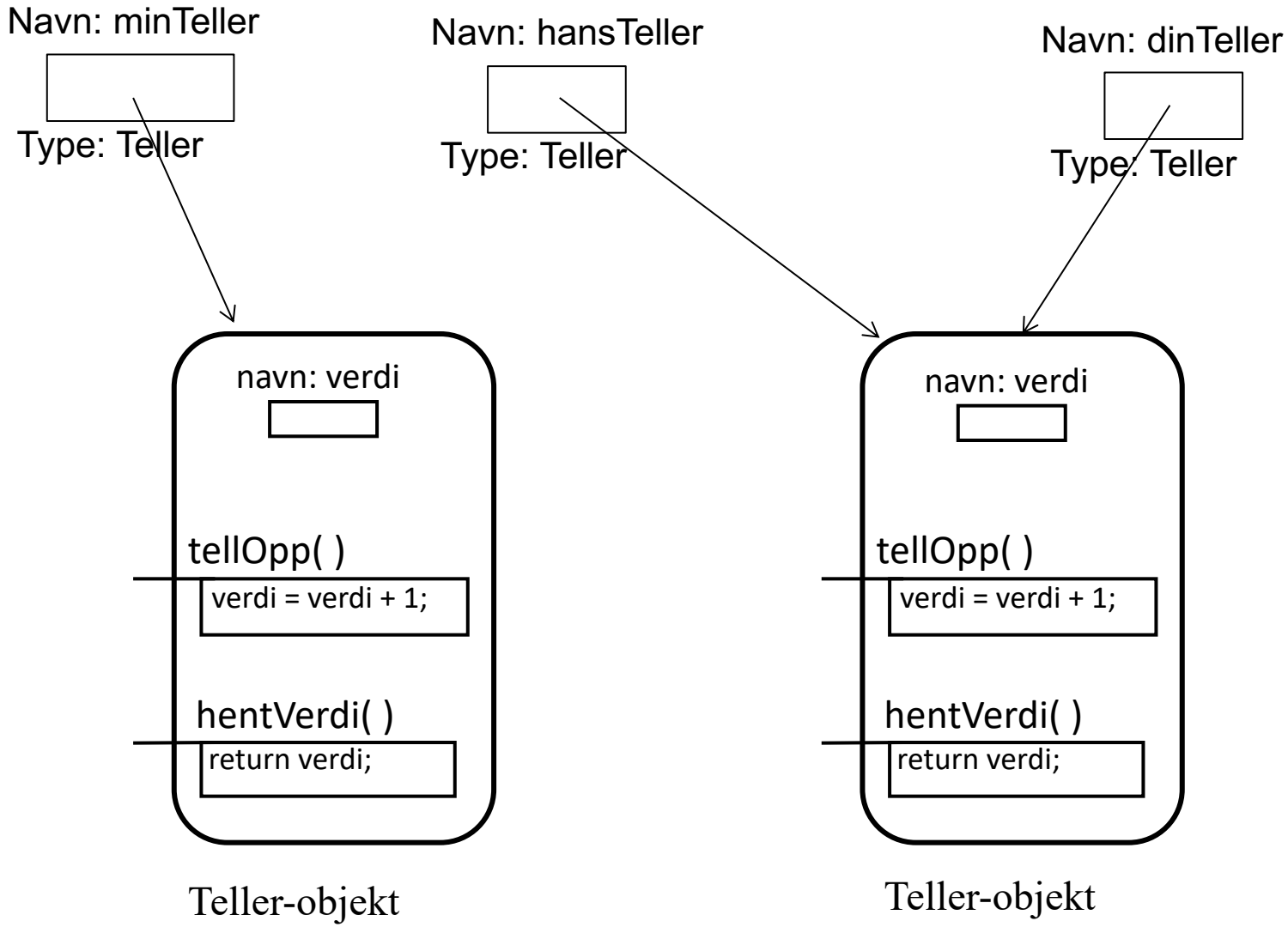
Java er **et sterkt typet språk:**
Referanse-variablen i dette eksemplet **har bare lov å inneholde en peker til et** Teller-objekt

Mer om referanser og tilordninger

Hva skjer om vi nå skriver

dinTeller = minTeller;

?

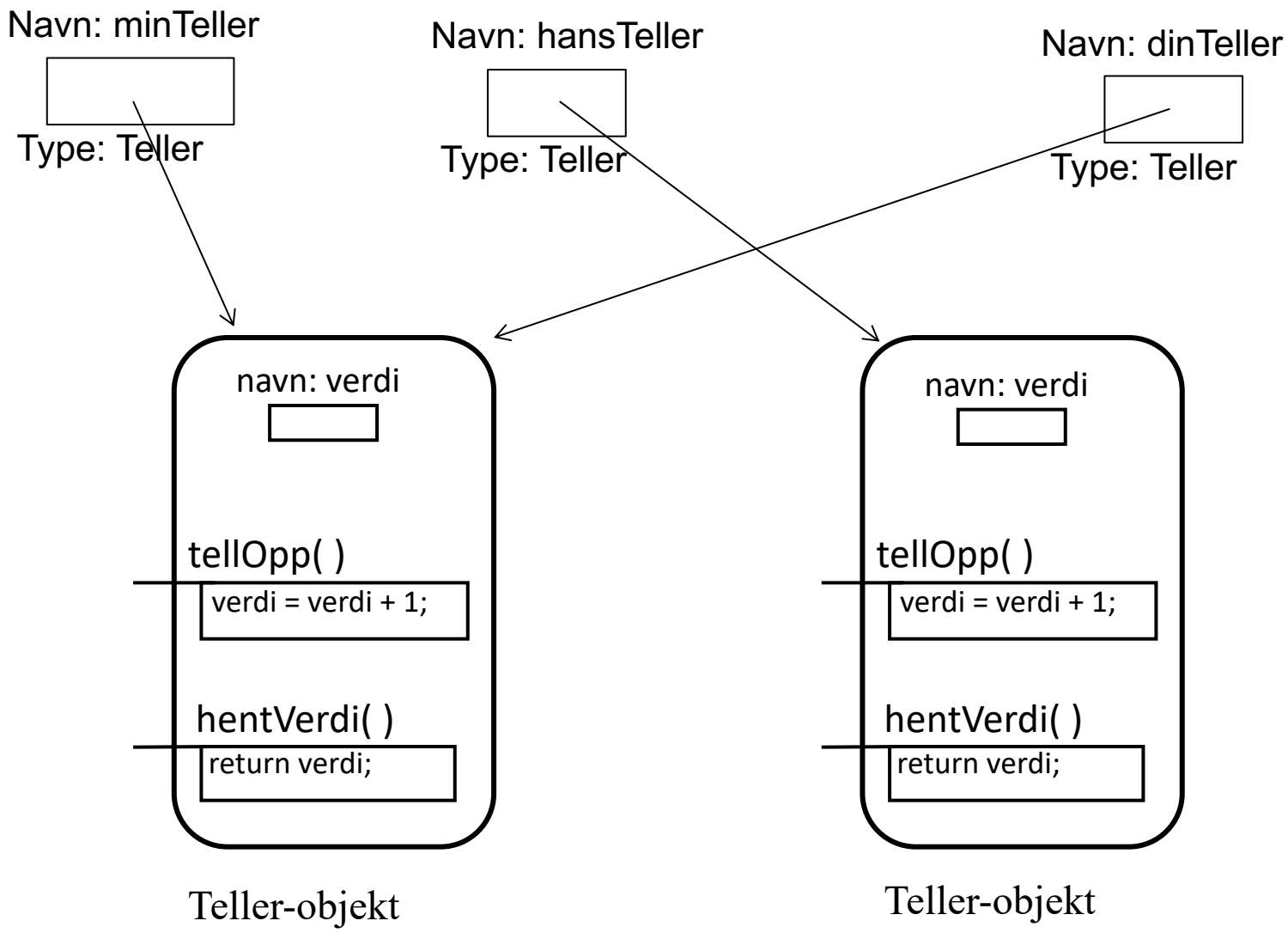


Mer om referanser og tilordninger

Etter

dinTeller = minTeller;

er tlstande slik:



Referanser

Navn: minTeller

53485320

Type: Teller

Navn: hansTeller

9483567

Type: Teller

Navn: dinTeller

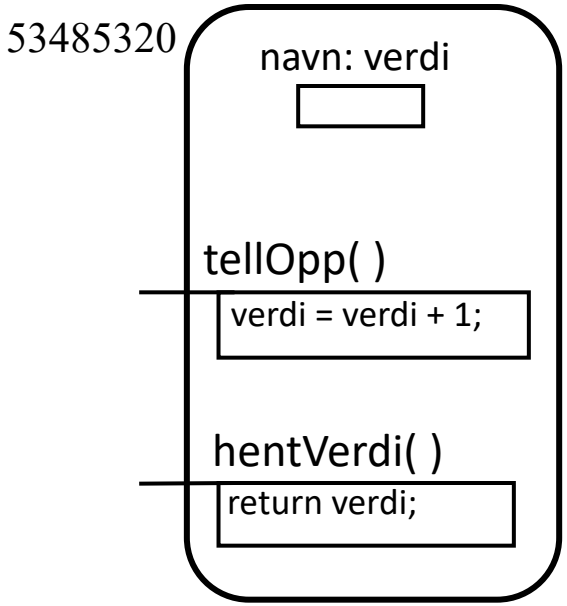
9483567

Type: Teller

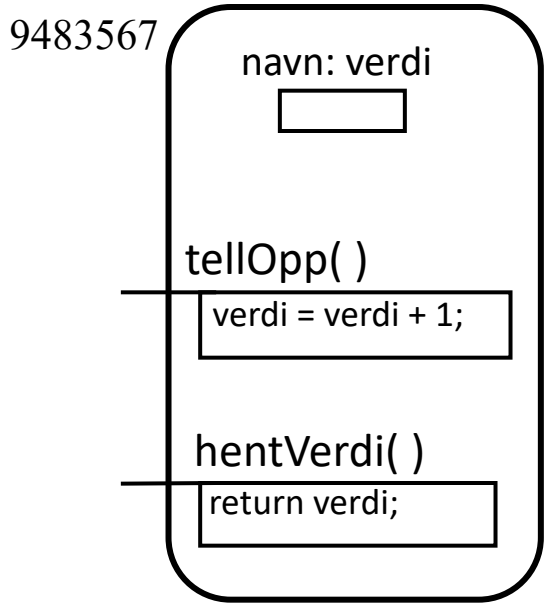
Hva skjer om vi nå skriver

dinTeller = minTeller;

?



Teller-objekt



Teller-objekt

Referanser

Navn: minTeller

53485320

Type: Teller

Navn: hansTeller

9483567

Type: Teller

Navn: dinTeller

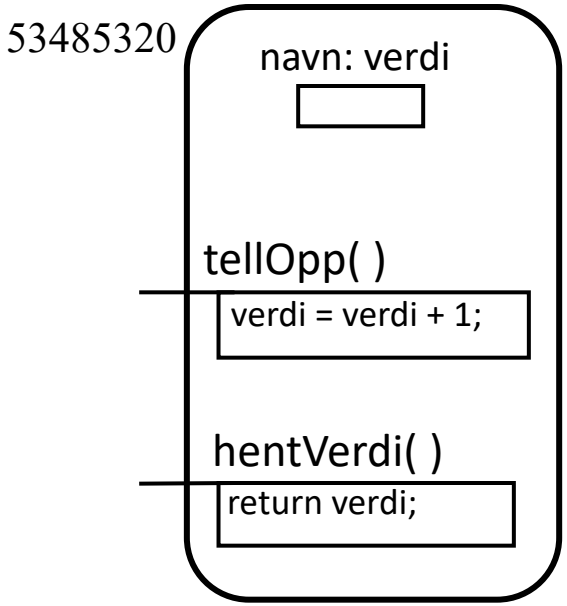
53485320

Type: Teller

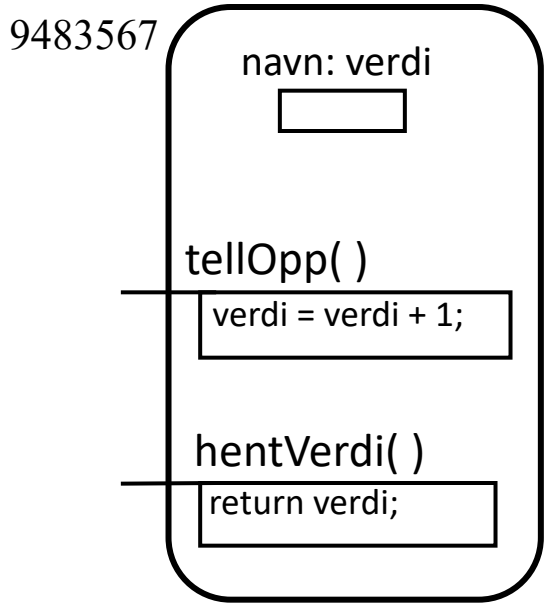
Etter

dinTeller = minTeller;

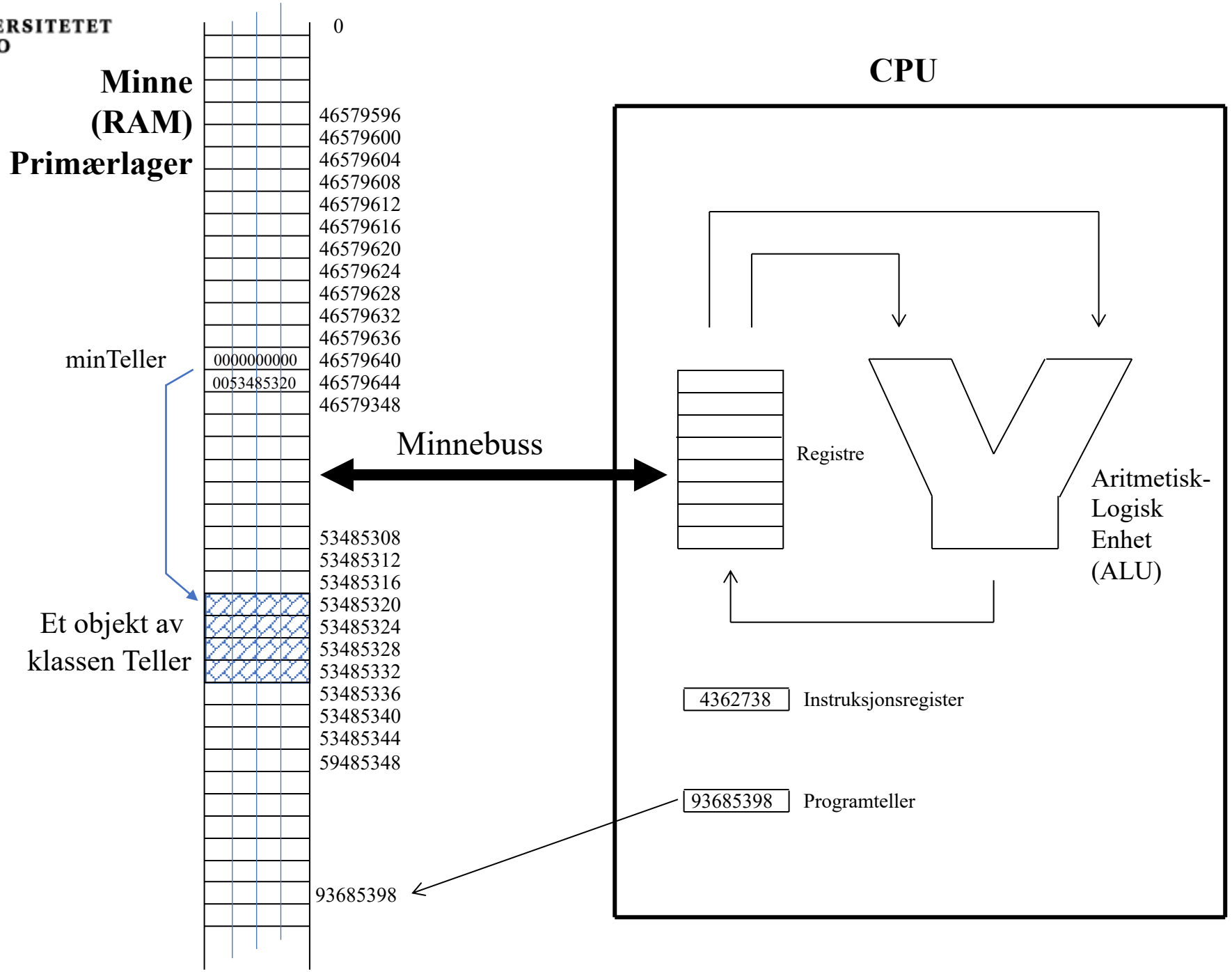
er tlstande slik:



Teller-objekt



Teller-objekt


Minne
(RAM)
Primærlager
0
46579596
46579600
46579604
46579608
46579612
46579616
46579620
46579624
46579628
46579632
46579636
minTeller
0000000000
46579640
0053485320
46579644
46579348
Minnebuss
53485308
53485312
53485316
53485320
53485324
53485328
53485332
53485336
53485340
53485344
59485348
Et objekt av
klassen Teller
93685398
CPU
Registre
Aritmetisk-
Logisk
Enhet
(ALU)
4362738
Instruksjonsregister
93685398
Programteller

Altså: En referanse er . . .

- **En referanse er** adressen til et objekt. Har en type som er et klassenavn
  - Det samme for arrayer (typen er da primitivtypen/referansetypen + [ ])

Referanse-verdi: 0101001011010011010110100010101001010110100110101000100101101110

64 bit / 8 byte

navn: . . . . type: <Klassenavn>

Referanse-variabel: 0000011001010000001000000110010000000110010100000010000001100111

Antall byte som brukes av referanser i Java er ikke definert i språket men avhengig av implementasjonen.
For tiden vanligvis 64 bit (64-bitsarkitektur). I gamle arkitekturer bare 32 bit

Javas uttrykk (expression)

- **Uttrykk** evalueres til en **verdi** av enten en
  - Primitiv verdi eller en
  - Referanseverdi
- De fleste uttrykk ser vi på høyresiden i en tilordning: tall = start + nyAntall;
- Husk at en aktuell parameter (argument) er et uttrykk: pus.spis(fisk+salat);
  - som evalueres til en verdi av en gitt type
  - Denne verdien blir startverdien til den formelle parameteren (en lokal variabel)
  - Se til slutt i dag
- Andre steder vi finner uttrykk
  - Boolske / sannhetsverdi – uttrykk
  - Indeks-uttrykk (int-indekser i en array)
  - ???
- Husk at uttrykk også kan være referanse-uttrykk (som beregner en referanse til et objekt eller en array)

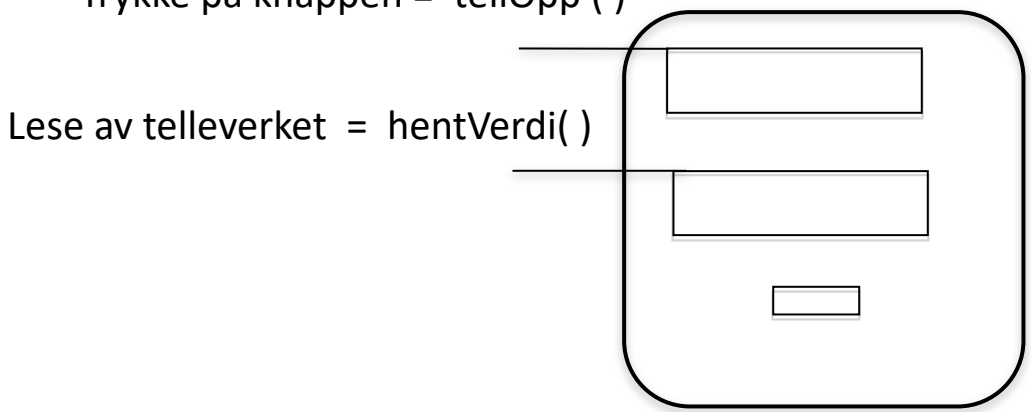
For en uke siden programmerte vi en teller

- Hvilke tjenester ønsker brukeren seg
- Har vi laget de tjenestene brukerene trenger?
- Hvem bestemmer hva grensesnittet skal være?
- Har vi programmert riktig?
- ??????????

Grenessnitt = tjeneste

Trykke på knappen = tellOpp ( )

Lese av telleverket = hentVerdi( )



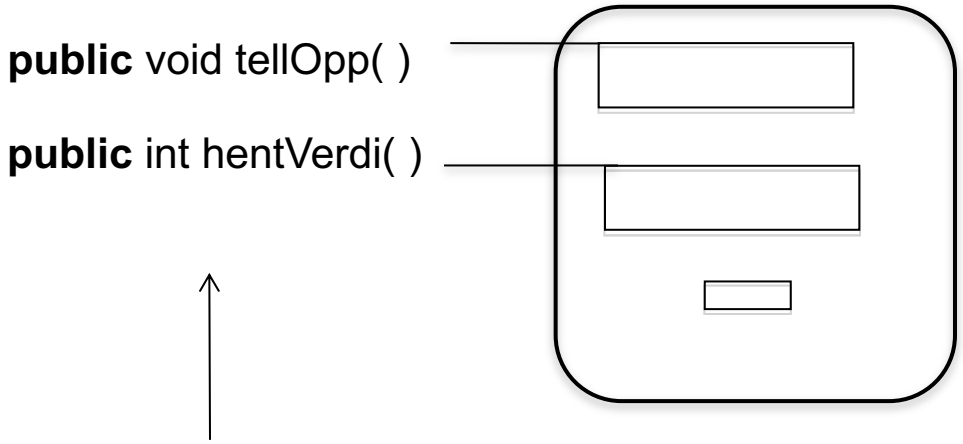
Ikke alltid lett å vite hva vi skal programmere

Når dere får obligatoriske oppgaver synes dere at oppgaveteksten er alt for lang og komplisert.

Men det er for at oppgaven skal være så presis at dere programmerer den “riktig”

Når vi lager tjenester:

En metodes *signatur*



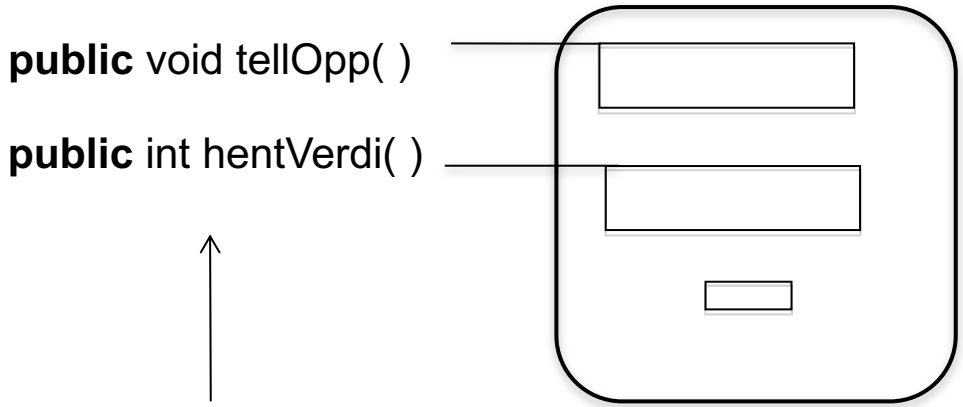
Dette kaller vi metodenes **signaturer** (skrivemåte, syntaks)

Signaturen til en metode er

- navnet på metoden
- typene, rekkefølgen og navnene til parametrene
- retur-typen (ikke i Java)

Når vi lager tjenester:

En metodes *semantikk*



Semantikk betyr
virkemåte

Hva gjør disse metodene? Hvordan virker de? Hvilke **tjenester** tilbyr de?
Hvordan beskrives disse tjenestene? Hva er sematikken til objektet / klassen?

Forslag til semantikk for et Teller-objekt:
- Når objektet opprettes er verdien 0
- Metoden hentVerdi returnerer objektets verdi
- Metoden tellOpp øker objektets verdi med 1

Men her snakker vi litt om
(en tenkt) implementasjon

Hva er en referanseimplementasjon? (slå opp på nettet)
(reference implementation)

Semantikk uten å snakke om implementasjon

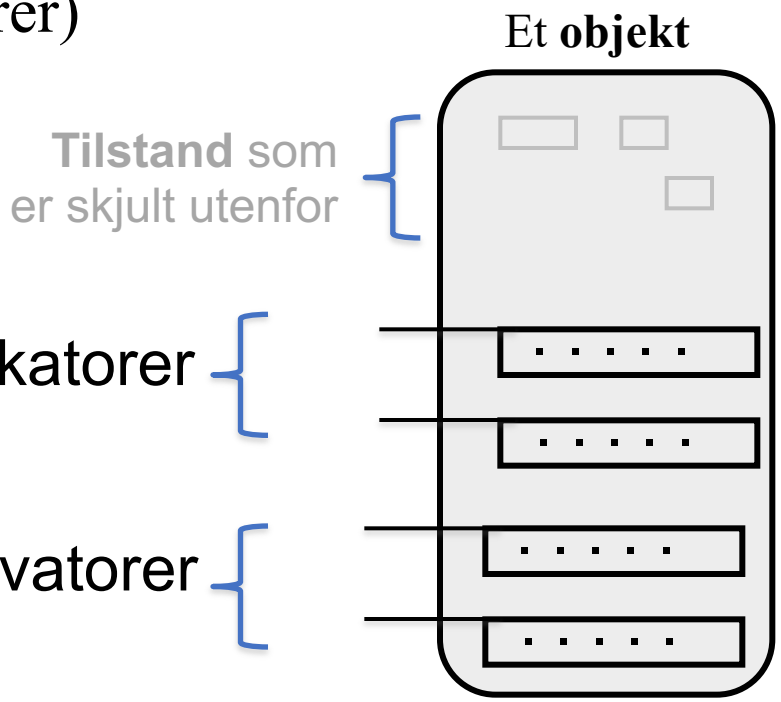
- Forslag til semantikk:
- Rett etter at objektet er opprettet returnerer hentVerdi 0
- Etter at metoden tellOpp er kalt vil hentVerdi retunere en verdi som er én større enn før tellOpp ble kalt siste gang

- Vi snakker altså bare om hva som skjer (hva som observeres) når metodene (som forandrer objektet) blir kalt

Modifikatorer og Observatorer

Accessors and mutators ?

- En modifikator-metode forandrer (modifiserer) tilstanden til et objekt
- En observator-metode leser av (observerer) tilstanden uten å forandre den



Modifikator, f.eks. tellOpp(), settVerdi()

Observator, f.eks. hentVerdi()

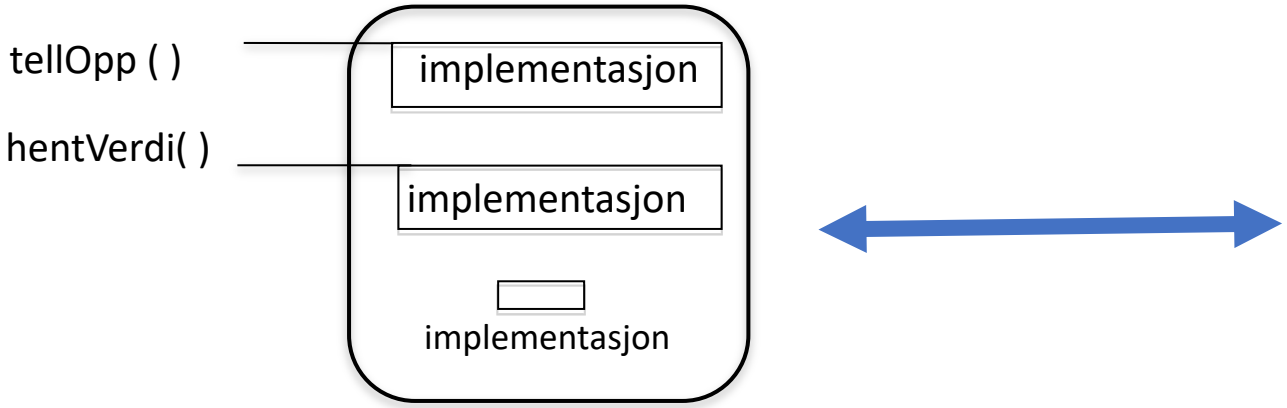
- # Informatikkens 3. lov: ☺

1. Først betemmer vi semantikken og signaturene til metodene
2. Deretter implementerer vi metodene
   samtidig som vi bestemmer hva de **private** dataene skal være

Dette gjelder for alle programmeringsspråk - dette er ikke Java-spesifikt.

☺ Dette er en spøk. Informatikken har ikke nummererte lover

Bare du, som er et menneske*, kan sjekke at implementasjonen overholder de SEMANTISKE KRAVENE til metodene (?)



Semantikk:

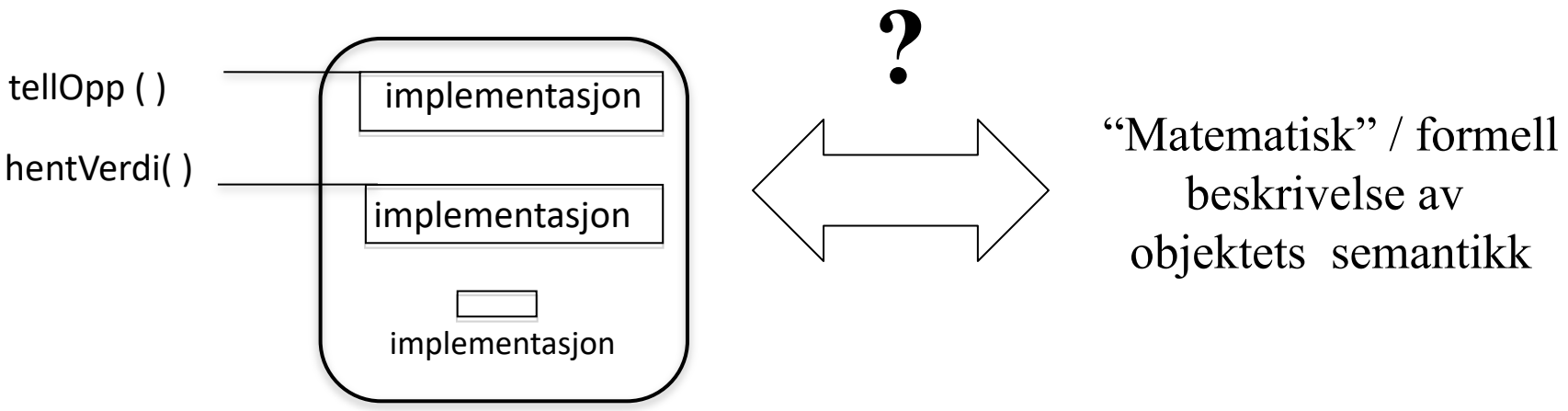
- Når objektet opprettes er verdien 0
- Metoden hentVerdi returnerer objektets verdi
- Metoden tellOpp øker objektets verdi med 1

* Se neste side

Institutt for informatikk har 16 forskningsgrupper.

En av disse heter "Pålitelige systemer" (PSY).

Her arbeider de bl.a. med å formalisere disse sematiske kravene, slik at du kan få hjelp av datamaskinen til å sjekke at implementasjonen overholder de semantiske kravene. Litt mer på en senere forelesning.



De sematiske kravene kalles også en "kontrakt"
(mellom brukerene av objektet og objektet selv)

Dokumentasjon og kommentarer

- Skriv en kommentar om hva hensikten / ansvarsområdet til ethvert objekt (klasse) er
  - Semantikken til klassen / objektet
- Skriv en kommentar om hva hensikten med hver metode er
  - Semantikken
- Hvis koden ikke er opplagt, f.eks.
  - Skriv en kommentar om hva en løkke gjør
    - Skriv en kommentar om hva ett gjennomløp av en løkke gjør

- Neste side: Dokumentasjon ved hjelp av Java-Doc

UNIVERSITETET I OSLO

Dokumentasjon med Java-doc

```java
/** Objekter av denne klassen er en teller
 *  Telleren starter på null
 *
 * Etter læreboka til
 * Cray Horstmann, SJSU
 * @author Stein Gjessing, UiO
 * @version 6. januar 2023
 */
public class Teller {
      private int verdi = 0;
      /**
      * Gjør at objektet teller opp.
      * Kan telle opp med mer enn en
      *
      *@param ant så mye telleren øker
      */
      public void tell(int ant ) {
           verdi+= ant;
      }
     /**
     * Henter ut av objektet dets verdi.
     * Denne verdien er så mye telleren totalt er talt opp med
     *
     @return  verdien som hentes ut
     */
      public int hentVerdi( ) {
           return verdi;
      }
}
```

```
steing@Steins-MBP-M1 programmer % javadoc Teller.java
Loading source file Teller.java...
Constructing Javadoc information...
Building index for all the packages and classes...
Standard Doclet version 17.0.1+12
Building tree for all the packages and classes...
Generating ./Teller.html...
Generating ./package-summary.html...
Generating ./package-tree.html...
Generating ./overview-tree.html...
Building index for all classes...
Generating ./allclasses-index.html...
Generating ./allpackages-index.html...
Generating ./index-all.html...
Generating ./index.html...
Generating ./help-doc.html...
steing@Steins-MBP-M1 programmer %
```

PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

SEARCH: Search

Class Teller

java.lang.Object
Teller

```
public class Teller
extends Object
```

Objekter av denne klassen er en teller Telleren starter på null Etter læreboka til Cray Horstmann, SJSU

Constructor Summary

Constructors

Constructor	Description
`Teller()`	

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
int	`hentVerdi()`	Henter ut av objektet dets verdi.
void	`tell(int ant)`	Gjør at objektet teller opp.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Details

Teller

```
public Teller()
```

Method Details

tell

```
public void tell(int ant)
```

Gjør at objektet teller opp. Kan telle opp med mer enn en

Parameters:
ant - så mye telleren øker

hentVerdi

```
public int hentVerdi()
```

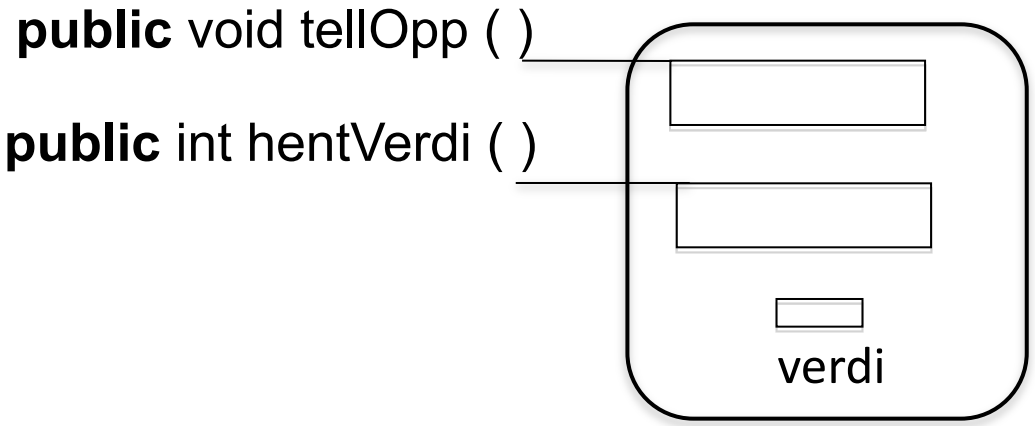
Henter ut av objektet dets verdi. Denne verdien er så mye telleren totalt er talt opp med

Returns:
verdien som hentes ut

r informatikk

Hvordan vet vi at programmet gjør det det skal?
Hvordan vet vi at objektet gjør det det skal?


```
class Teller {
      private int verdi = 0;
      public void tellOpp( ) {
            verdi++;
      }
      public int hentVerdi( ) {
            return verdi;
      }
}
```



Et Teller-objekt

Vi må gjøre 2 ting:

- *Vi må tenke og resonere nøye når vi designer og skriver programmet*
- *Vi må **teste** programmet*

Enhetstesting av objekter

Nå skal du lære litt om OO-programmering samtidig som du lærer Java

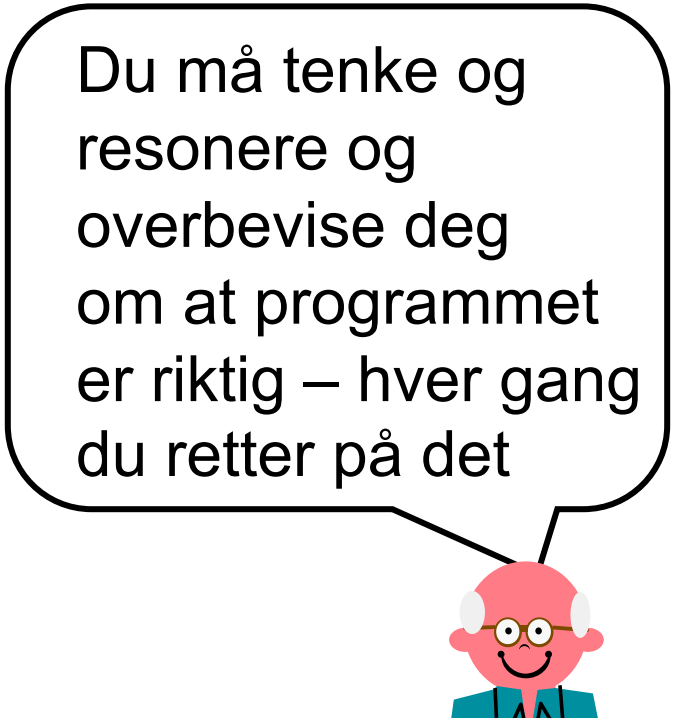
Testing -- Enhetstesting

- Når vi planlegger og skriver programmer prøver vi å overbevise oss selv (og dem vi skriver sammen med) at den koden vi skriver kommer til å utføre det vi ønsker (oppfylle semantikken)
- Men vi kommer alltid til å tenke og skrive feil
- Derfor må vi **teste** programmene våre
- Objektorientering / modularisering:
  - Teste et objekt eller en modul om gangen - **enhetstesting**
    - Sørg for at den er så riktig som mulig
  - Deretter kan vi test sammensettingen av objektene / modulene - **integrasjonstesting**

Mer om testing etter påske

Det er IKKE lov å

- Skrive et program til du tror at det kanskje virker
- Teste det, men det feiler
- Gjette på hva som er feil og rette én ting
- Teste på ny, men det feiler
- Gjette på hva som er feil og rette én ting
- Teste på ny, men det feiler
- Gjette på hva som er feil og rette én ting
- Teste på ny, . . .
- . . .



Fordi testing ikke viser fravær av feil og . . .

Når du skriver et større program SKAL du

- Skrive en liten del og kompilere og kjøre den
- Når den lille delen virker utvider du den og kompilerer og kjører
- Når dette litt større programmet virker utvider du det, komilerer og kjører det
- Når dette litt større programmet virker utvider du det, kompilerer og kjører det
- o.s.v . . .
- . . .
- Helt til hele programmet kompilerer og kjører og er testet med mange forskjellige inn-data

Obligatoriske innleveringer

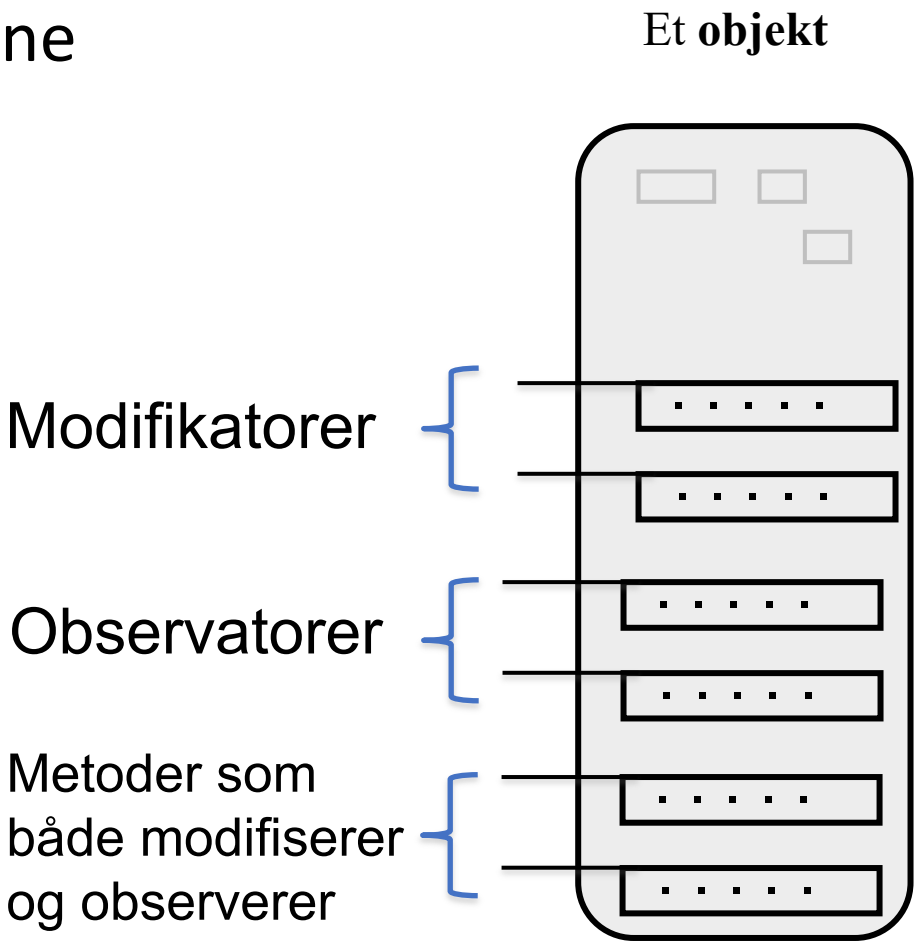
- **Når du leverer en oblig skal den alltid kompiler og kjøre**
- Hvis du ikke blir helt ferdig skal det du leverer **kompilere og kjøre**
- Om du leverer noe som **kompilerer og kjører** og som er et hederlig forsøk vil du få et nytt forsøk
- Selv om du ikke har levert en perfekt besvarelse av en oblig vil den kunne bli godkjent bare den **kompiler og kjører**

- Hvordan er dette mulig?
- Jo, fordi du har gjort som på forrige side:

Programmet ditt programmerer og kjører, ALLTID

Tilbake til testing - - Enhetstesting

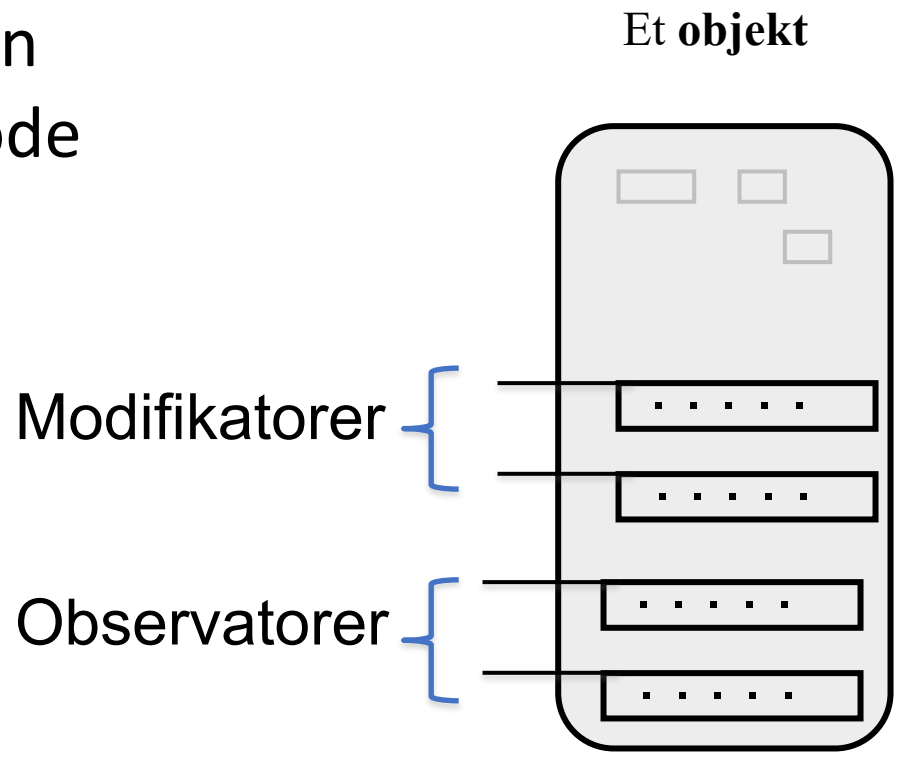
- Lag et objekt og se hva som skjer når du kaller metodene
- Du må selv sette opp rekkefølgene av kallene og vite hva resultatet skal bli
- Hva er de mulige feilene?
  - Ekstern testing vs. intern testing.
- Prøv å ta med så mange forskjellige kall-rekkefølger som mulig



Testing - - Enhetstesting

Hvis mulig:

- Når vi skal teste et objekt kan vi først kalle en observator-metode, så en modifikator-metode og så igjen observator-metoden og se om vi observerer det ønskede resultatet - den ønskede forandringen i **tilstanden**.



For de som er spesielt interessert:

Ole-Johan Dahl

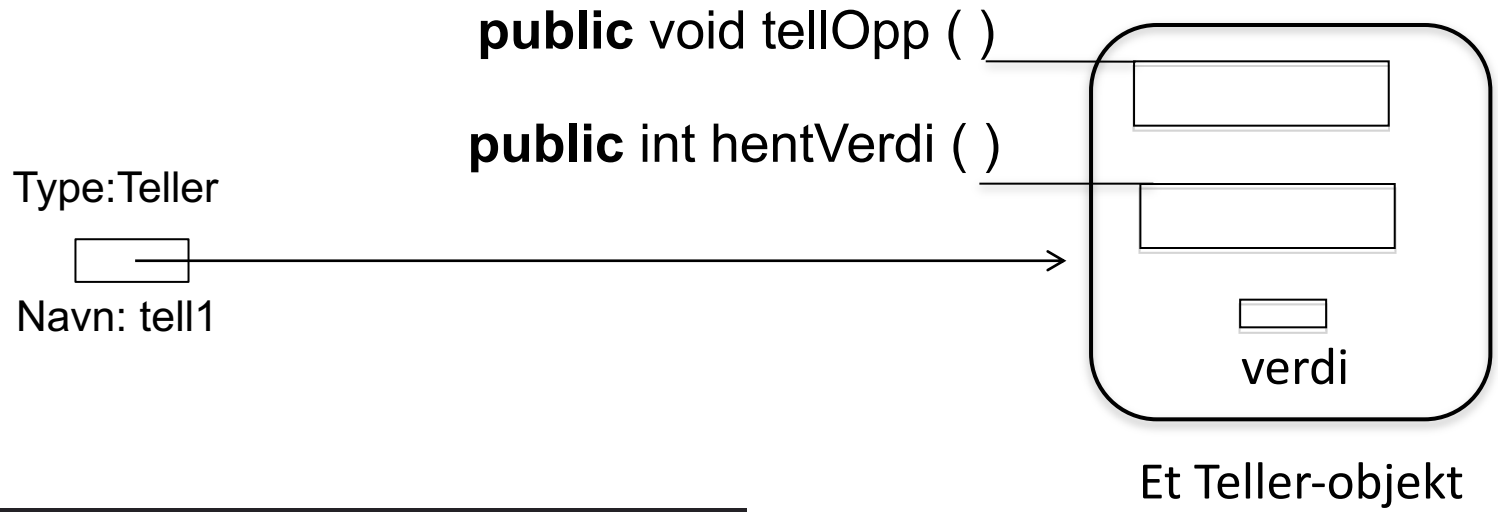
Objektets semantikk kan beskrives av den historiske sekvensen av operasjoner på objektet

UNIVERSITETET I OSLO

Om å teste et objekt / klasse

Institutt for informatikk

```
class Teller {
      private int verdi = 0;
      public void tellOpp( ) {
            verdi++;
      }
      public int hentVerdi( ) {
            return verdi;
      }
}
```



Et Teller-objekt

```
class BrukTellerTest1 {
     public static void main (String [ ] arg) {
           Teller tell1 = new Teller( );
           tell1.tellOpp();
           tell1.tellOpp();
           tell1.tellOpp();
           // test:
           System.out.println("Telleren er nå " + tell1.hentVerdi());
           System.out.println("Telleren skal være 3");
           if (tell1.hentVerdi() == 3) System.out.println("Svaret er riktig");
              else System.out.println("Svaret er feil");
      }
}
```

Når skal vi skrive testprogrammet?

Om du er alene?
Om dere er flere?

Samme – men med egen (statisk) metode for å teste


```
class Teller {
      private int verdi = 0;
      public void tellOpp( ) {
            verdi++;
      }
      public int hentVerdi( ) {
            return verdi;
      }
}
```


```
class BrukTellerTest2 {
     public static void main (String [ ] arg) {
           Teller tell1 = new Teller( );
           tell1.tellOpp();
           tell1.tellOpp();
           tell1.tellOpp();
           // test:
           test(tell1.hentVerdi(),3);
      }
      private static void test (int resultat, int fasit) {
           if (resultat == fasit) System.out.println("Svaret er riktig");
              else System.out.println("Svaret er feil");
      }
}
```

Flere eksempler på enhetstesting: Om å lage et kaninbur

```
class Kanin{
    private String navn;
    public Kanin(String nv) {
        navn = nv;
    }
    public String hentNavn() {
        return navn;
    }
}
```

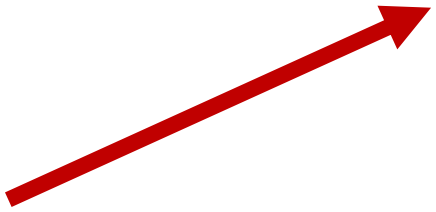
```
class Kaninbur {
    private . . . ;
    public boolean settInn(Kanin k) {
        . . .
    }
    public Kanin taUt( ) {
        . . .
    }
}
```

Kaninburets tjenester / grensesnitt – Signaturene og semantikken til metodene i kaninburet

Signaturer (syntaks)

```
class Kaninbur {
      public boolean settInn(Kanin k) {. . .}
      public Kanin taUt( ) { . . . }
}
```

Kan det beskrives enklere ?



Semantikk (virkemåte):

- Hvis objektet (buret) er tomt vil metoden "settInn" gjøre at objektet tar vare på kaninen som er parameter til metoden, og metoden returnerer sann.
  Hvis objektet allerede inneholder en kanin gjør metoden ingen ting med objektet, og metoden returnerer usann.

- Metoden "taUt" tar ut kaninen som er i objektet og returnerer en peker til denne kaninen. Metoden returnerer null hvis objektet var tomt.

Kaniner og kaninbur: Full kode


```
class Kanin{
    private String navn;
    public Kanin(String nv) {
        navn = nv;
    }
    public String hentNavn() {
        return navn;
    }
}
```

Men enda ikke et fullt program (mangler klasse med main).

```
class Kaninbur {
    private Kanin denne = null;
    public boolean settInn(Kanin k) {
        if (denne == null)  {
            denne = k;
            return true;
        }
        else return false;
    }
    public Kanin taUt( ) {
        Kanin k = denne;
        denne = null;
        return k;
    }
}
```

UNIVERSITETET I OSLO

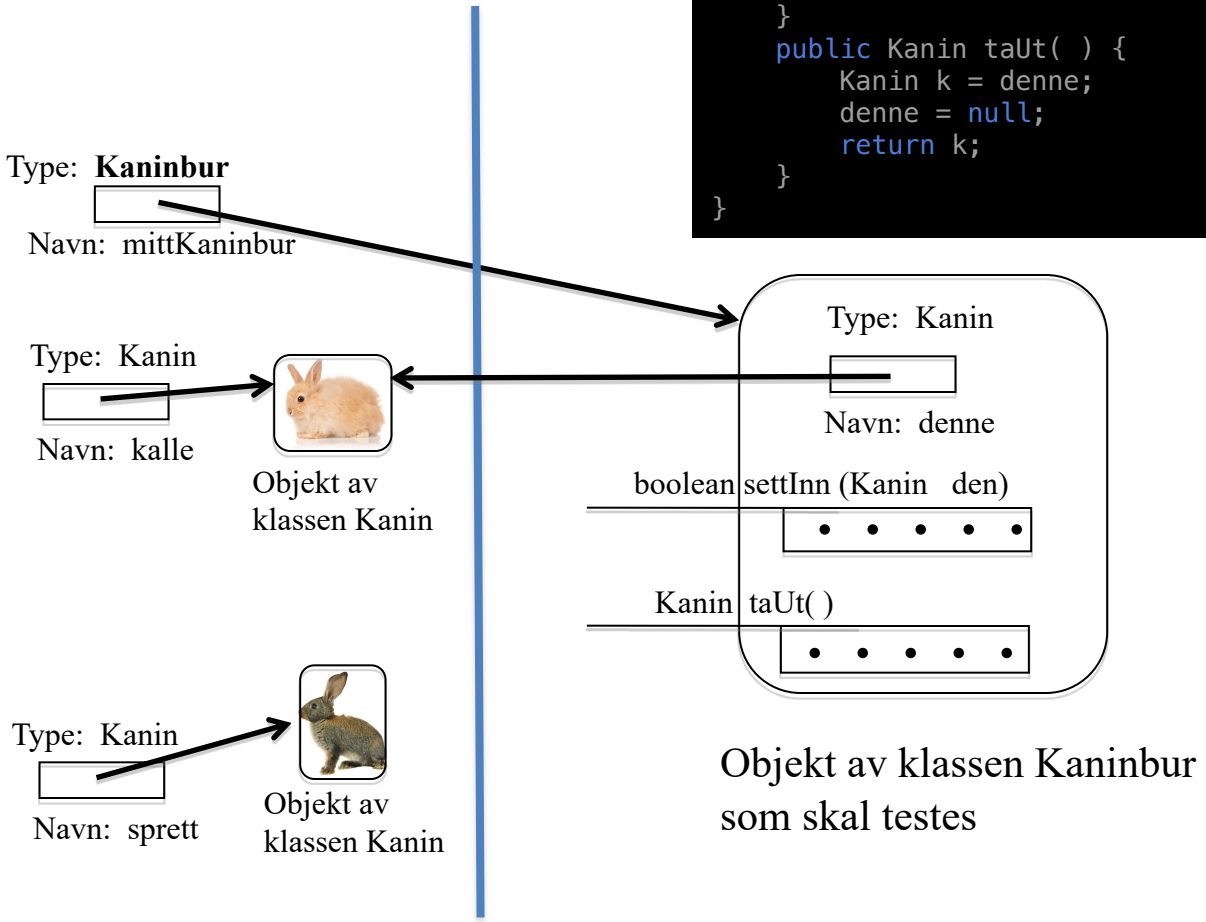
Enhetstesting av class Kaninbur

```
class Kanin{
    private String navn;
    public Kanin(String nv) {navn = nv;}
    public String hentNavn() {return navn; }
}
```

```
class Kaninbur {
    private Kanin denne = null;
    public boolean settInn(Kanin k) {
       if (denne == null)  {
          denne = k;
          return true;
       }
       else return false;
    }
    public Kanin taUt( ) {
        Kanin k = denne;
        denne = null;
        return k;
    }
}
```

Testprogram:

```
class Kanintest {
   public static void main ( String []args) {
      Kaninbur mittKaninbur = new Kaninbur( );



     Kanin kalle = new Kanin("Kalle");
     // Test å sette inn i tomt bur:
     boolean settInnOK = mittKaninbur.settInn(kalle);
     test("Test inn i tomt bur", settInnOK);


     Kanin sprett = new Kanin("Sprett");
     // Test å sette inn i fullt bur:
     boolean settInnOK = mittKaninbur.settInn(sprett);
     test("Test inn i fullt bur", !settInnOK);
     . . .
     . . .
     . . .

  }
  static void test(…) { . . . }
}
```



Om å lage et kaninbur til mange kaniner

3 observatorer

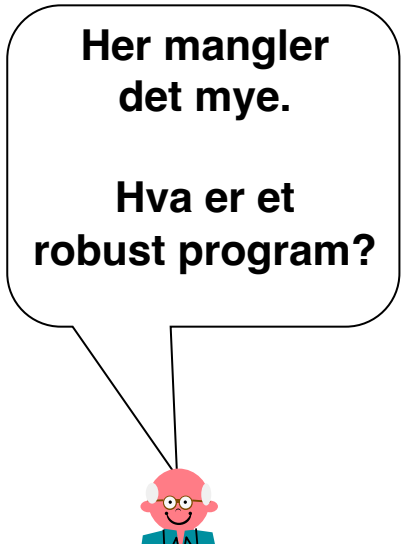
2 modifikatorer

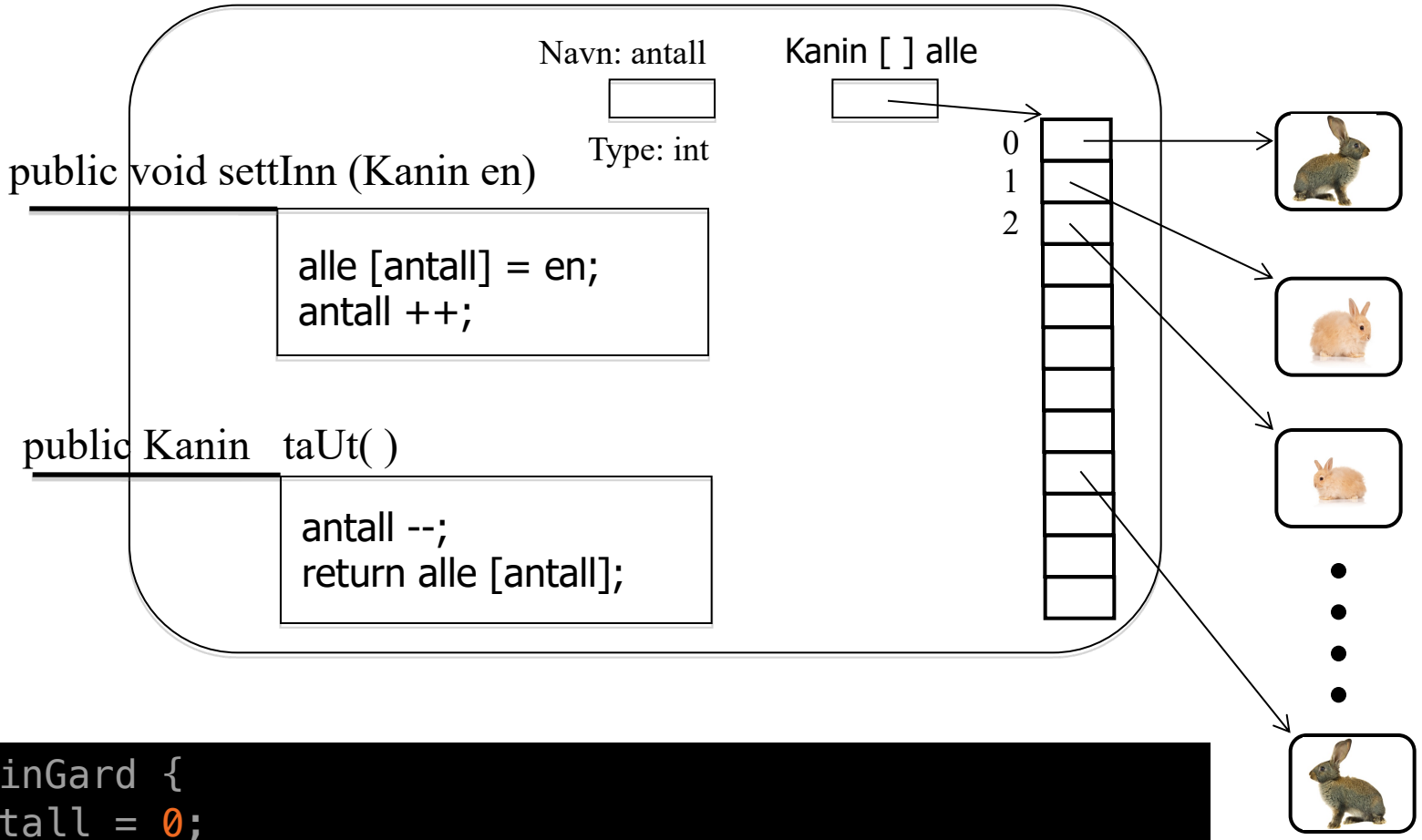
```
class    KaninGård {
   public boolean full( ) { . . . }
   public boolean tom ( ) { . . . }
   public Kanin  finnEn(String navn) { . . . }
   public void settInn (Kanin kn) { . . . }
   public void fjern(String navn) { . . . }
}
```

Men at reglen om at vi bare skal ha helt rene observator-metoder og helt rene modifikator-metoder er kanskje å drive det litt langt.
For eksempel public Kanin hentUt(String navn) { . . . }

Veldig forenklet kaningård med litt andre metoder på neste side

Objekt av klassen ForenkletKaninGård

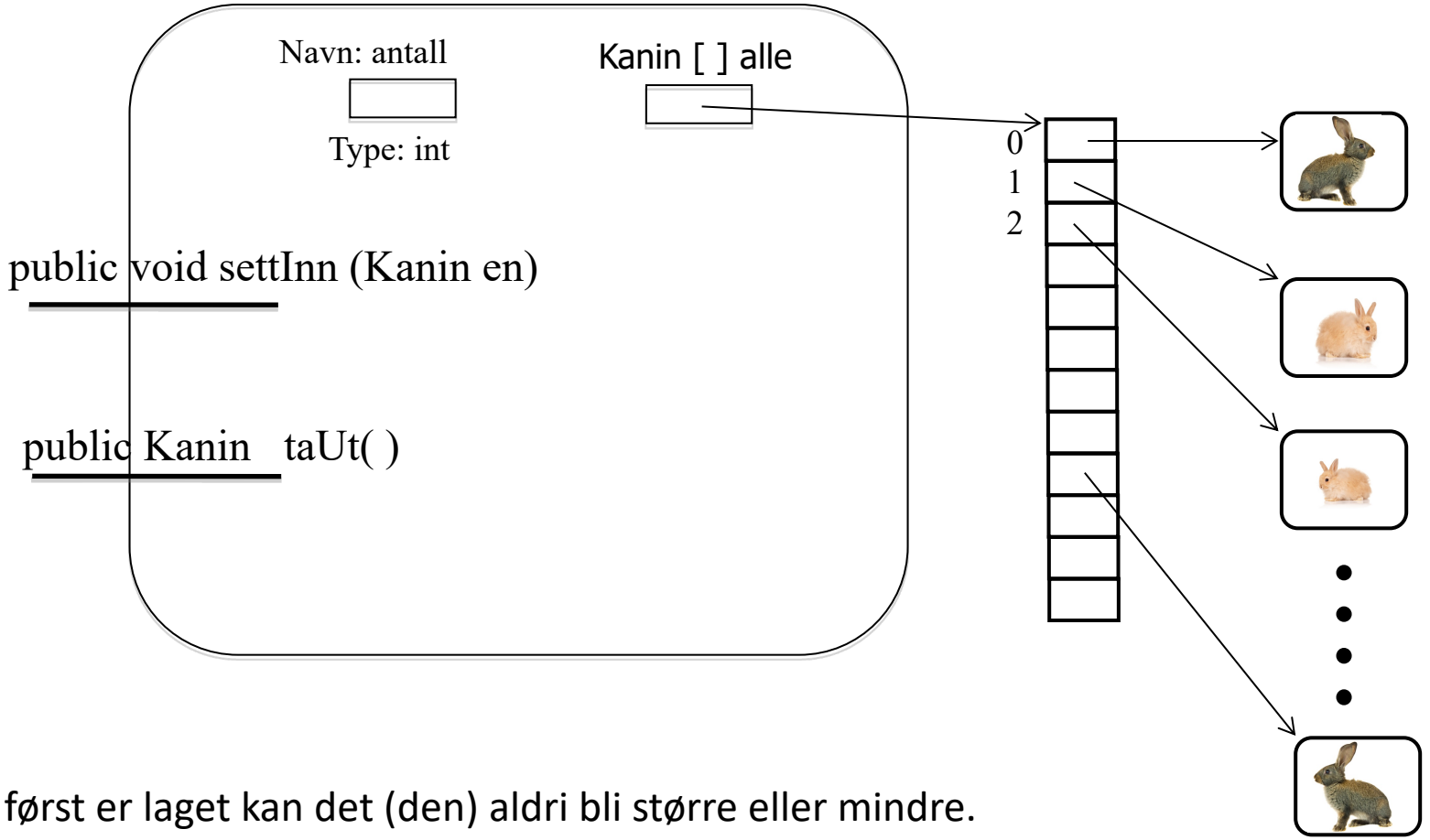





```
class ForenkletKaninGard {
    private int antall = 0;
    private Kanin [ ] alle = new Kanin[100];

    public void settInn(Kanin en) { alle[antall] = en; antall ++; }

    public Kanin taUt( ) { antall --; return alle[antall]; }
}
```

Oppgave:
skriv ferdig klassen KaninGård fra forrige side

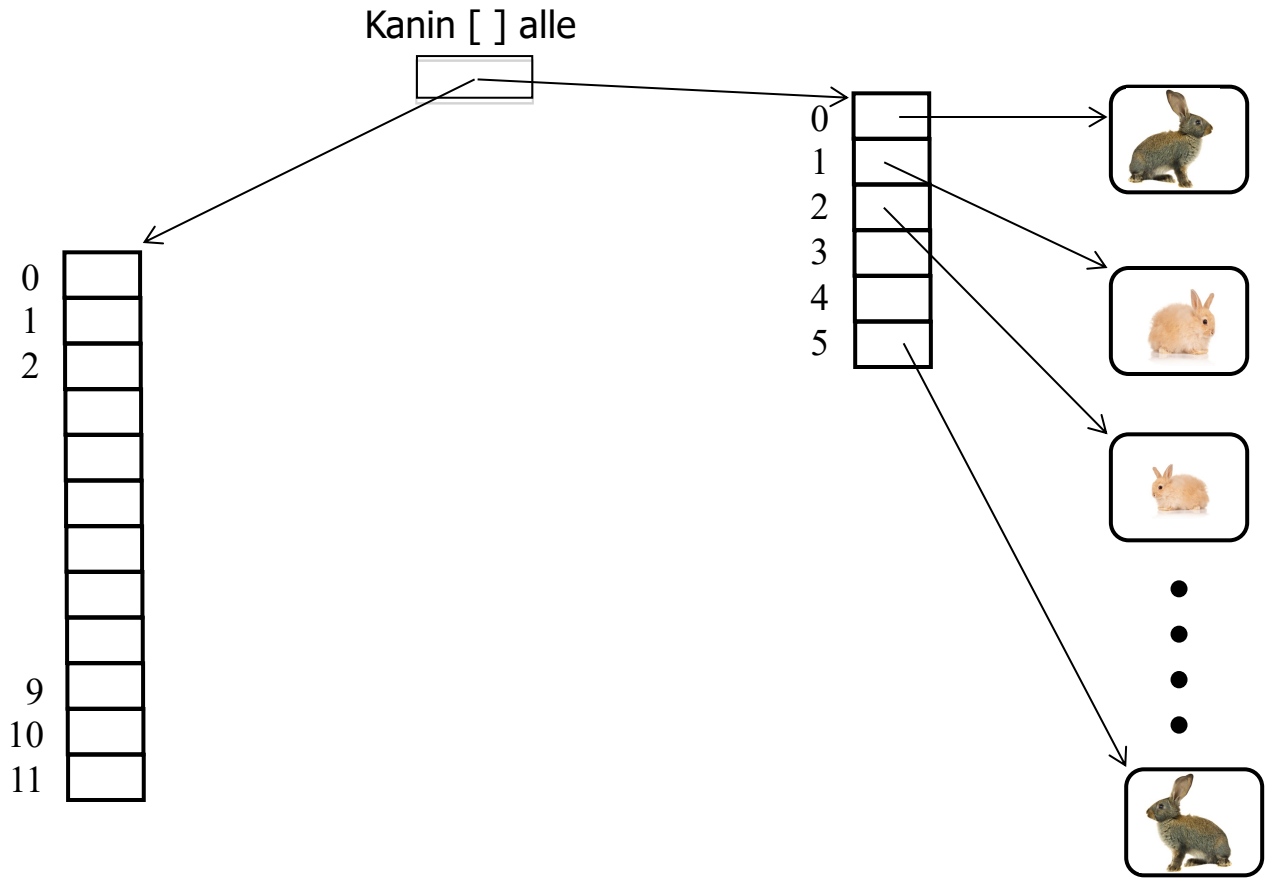
Selv om vi noen ganger tegner objekter inne i hverandre og arrayer inne i objekter, vil alltid objekter og arrayer ligger ved siden av hverandre i minnet.



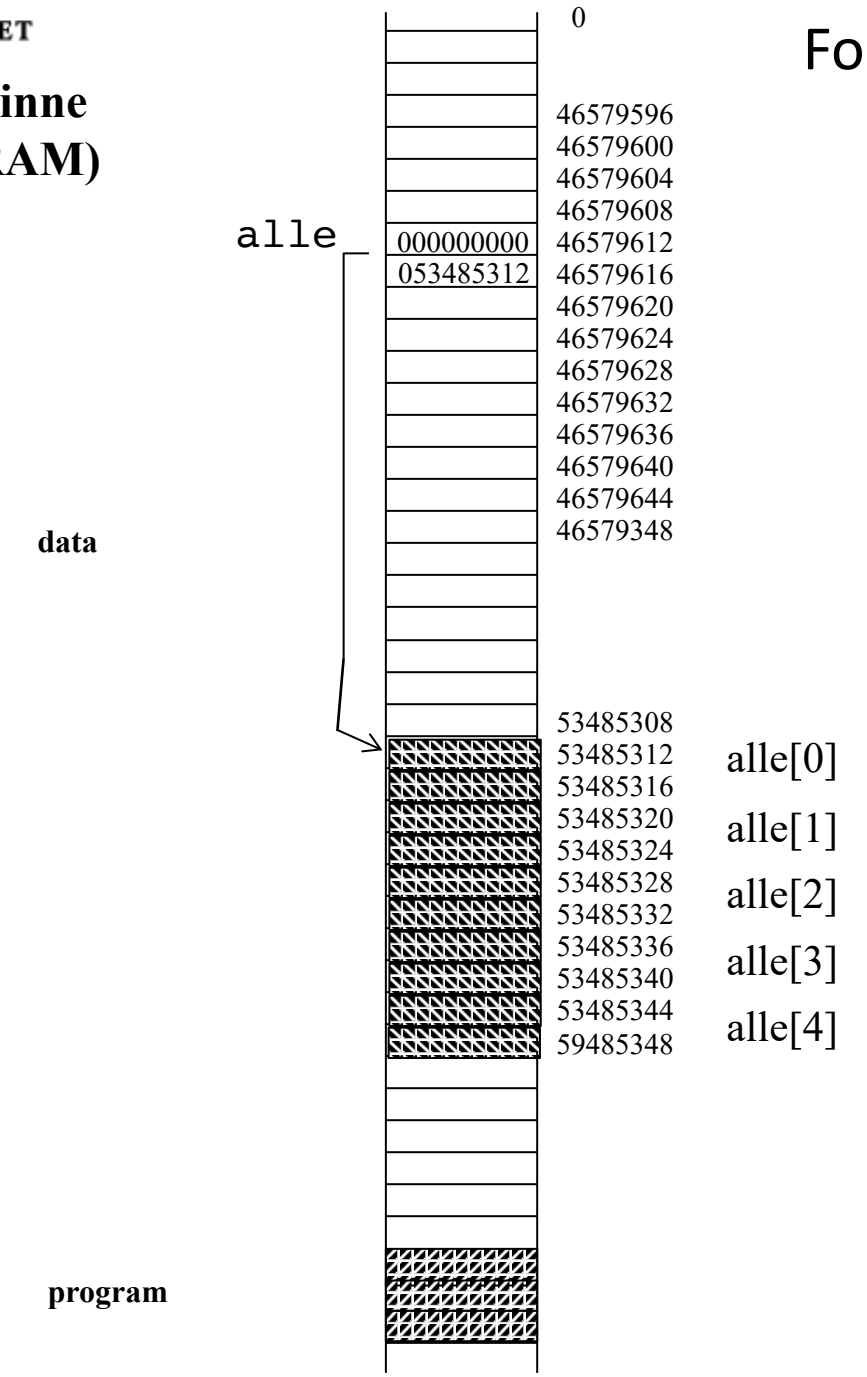
Når et objekt (eller array) først er laget kan det (den) aldri bli større eller mindre.
Det er bare innholdet (variablene) som kan gis nye verdier
(og variablene kan ikke forandre type)

Array-tilordninger

alle = new Kanin[12];



For en uke siden så dere denne tegningen



Array er iboende i datamaskinen og i Java

Oppslag (indeksering) i en array er veldig. fort gjort av datamaskinen

Husk: Minnet er byte-adresserbart

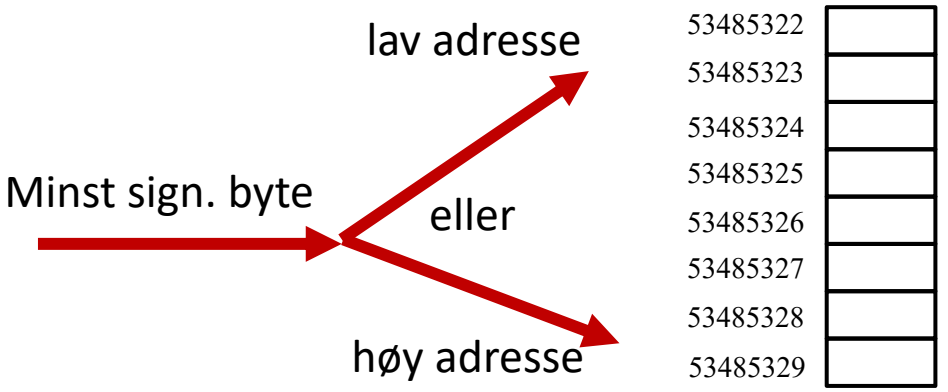
Her har jeg tegnet 4 byte = 32 bit på hver linje og `alle` er en array av referanser (á 64 bit)

For spesielt interesserte: Rekkefølgen i minnet, «endianness».

boolean	00000001
byte	01001110
char	0100111010110110
short	0100100101101110
int	01010010110110101000100101101110
long	0101001. . . . . .01001101010001001011011 10
float	01010010110110101000100101101110
double	01010010110. . . . . .11010100010010110111 0
referanse	0101000110100110 00100101101110

"Little endian"

i RAM:



"Big endian "

Du kan vel historien om “Gullivers reiser”?

To samlinger (beholdere/collections) fra Javas bibliotek:

ArrayList og **HashMap**

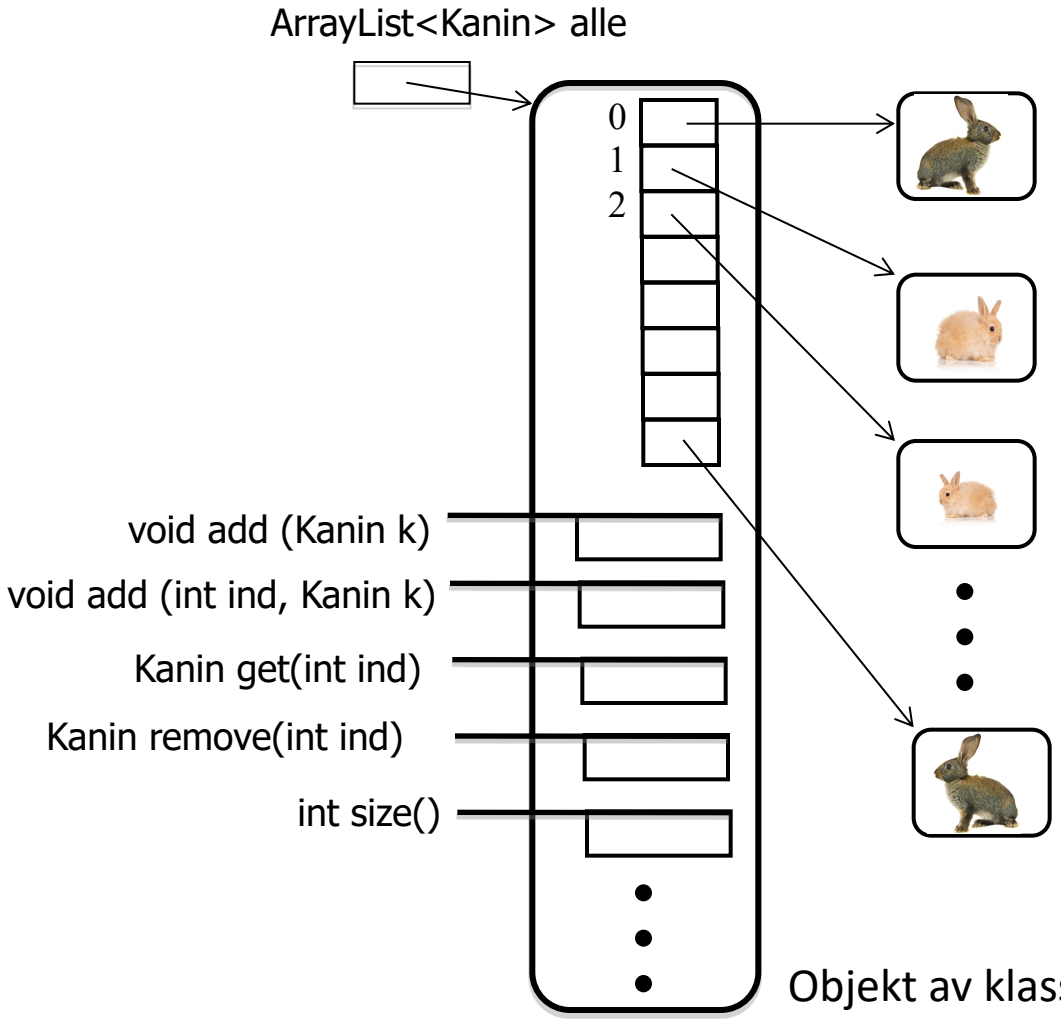
- ArrayList er en fleksibel array som utvider seg og trekker seg sammen etter behov
- ArrayList <Kaniner> mineKaniner = new ArrayList <Kaniner> ();
- Metoder: add, get, remove, . . . se Java-biblioteket

- HashMap er en samling der elementene identifiseres ved en nøkkel / navn
- HashMap<String,Kaniner> alleKaninene = new HashMap<String, Kaniner> ( );
- Metoder: put, get, remove, . . . se Java-bilblioteket

ArrayList

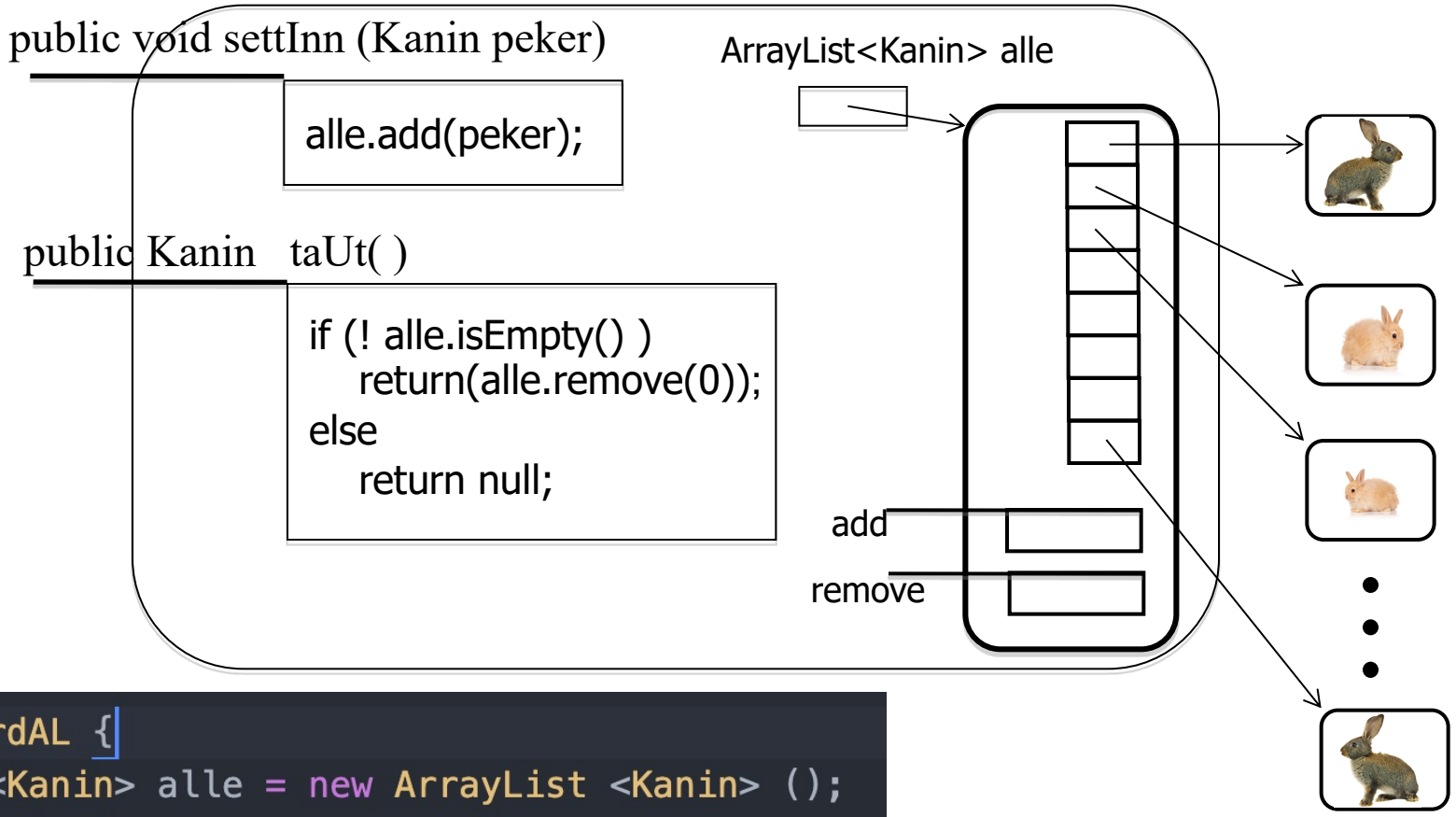
Deklarasjon:

```
ArrayList<Kanin> alle = new ArrayList< >( );
```



Objekt av klassen ArrayList

Objekt av klassen ForenkletKaninGardAL



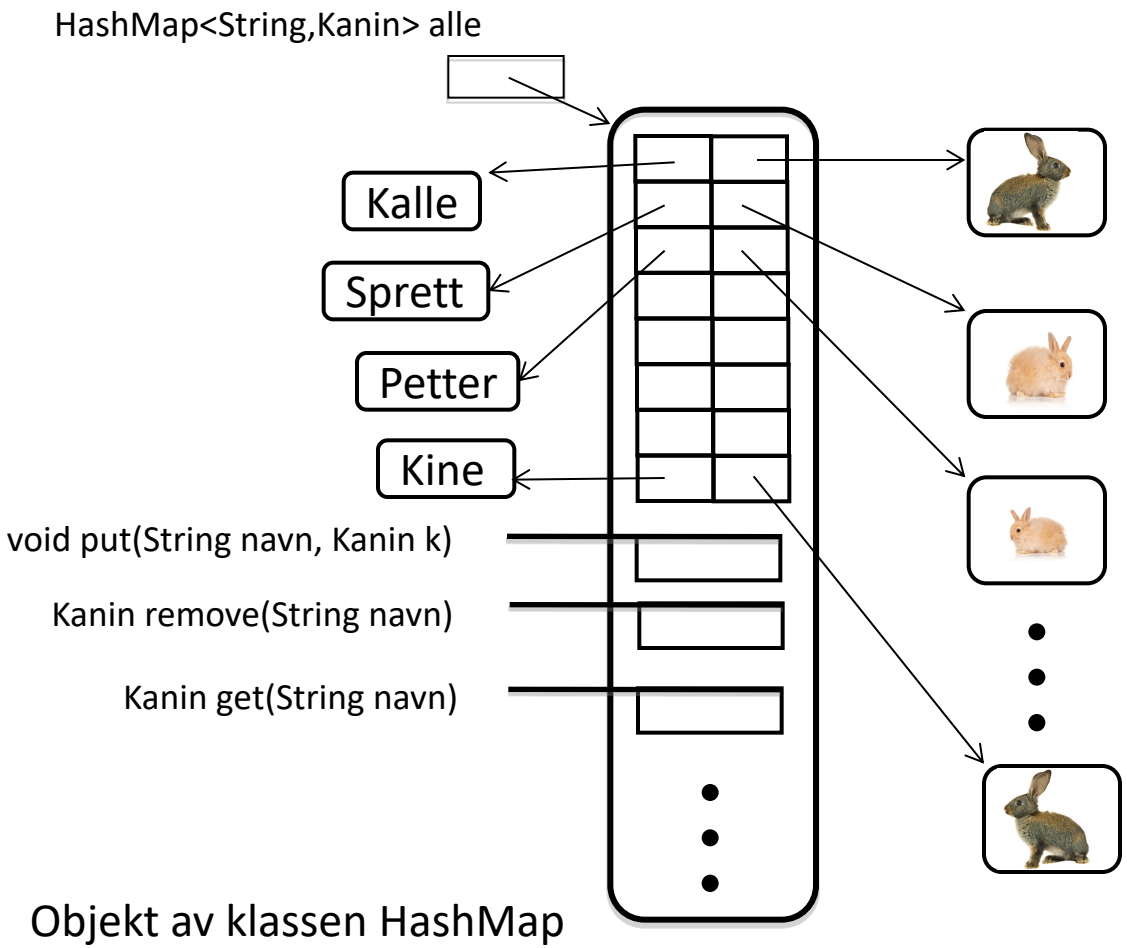

```
class ForenkletKaninGardAL {
    private ArrayList <Kanin> alle = new ArrayList <Kanin> ();
    public void settInn(Kanin peker) {alle.add(peker);}
    public Kanin taUt() {
      if (! alle.isEmpty()) {return (alle.remove(0));}
      else  return null;
    }
}
```

```java
import java.util.ArrayList;

class Kanin{
    private String navn;
    Kanin(String nv) {navn = nv;}
    public String hentNavn ( ) {return navn;}
}

class ForenkletKaninGardAL {
    private ArrayList <Kanin> alle = new ArrayList <Kanin> ();
    public void settInn(Kanin peker) {alle.add(peker);}
    public Kanin taUt() {
      if (! alle.isEmpty()) {return (alle.remove(0));}
      else  return null;
    }
}
```

```
class  KaningardTestArrayList {
  public static void main (String [ ] args) {
    ForenkletKaninGardAL mittKaninbur = new ForenkletKaninGardAL( );
    Kanin kalle = new Kanin("Kalle");
    mittKaninbur.settInn(kalle);
    Kanin sprett = new Kanin("Sprett");
    mittKaninbur.settInn(sprett);
    // test at først inn kommer først ut
    Kanin enKanin = mittKaninbur.taUt();
      test(((enKanin != null) && enKanin.hentNavn().equals("Kalle")),1);
    // test at neste inn nå kommer ut
    enKanin = mittKaninbur.taUt( );
      test(((enKanin != null) && enKanin.hentNavn().equals("Sprett")),2);
    // test at buret nå er tomt
    enKanin = mittKaninbur.taUt();
      test((enKanin == null),3);
  }
   static void test(boolean riktig, int testNr) {
      if (riktig) {
          System.out.println("Riktig test nummer " + testNr);
      }  else {
        System.out.println("Feil test nummer " + testNr);
    }
  }
}
```

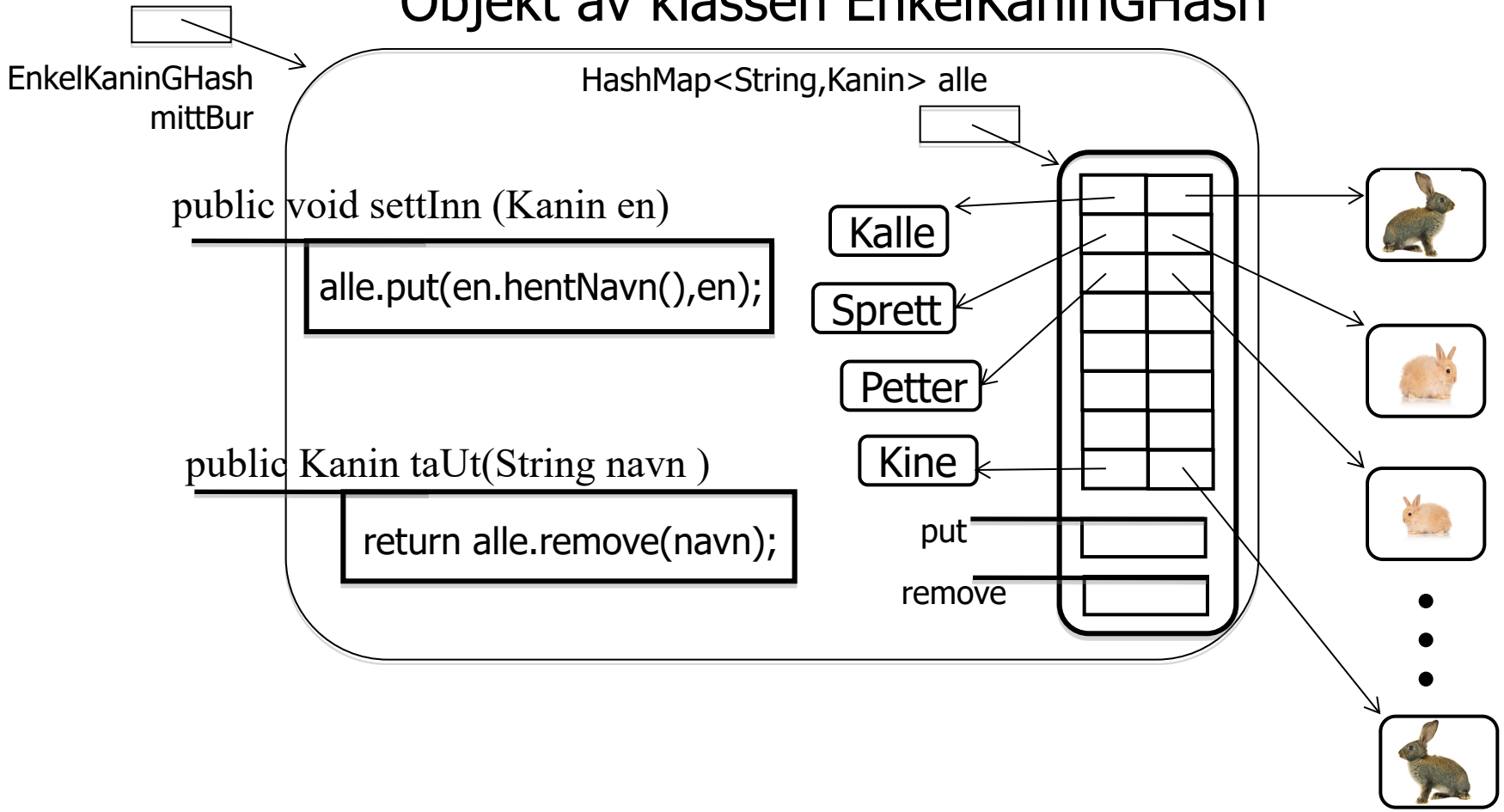
HashMap

Deklarasjon:
HashMap<String,Kanin> alle = new HashMap<String, Kanin>( );



Objekt av klassen HashMap

Objekt av klassen EnkelKaninGHash




```
class EnkelKaninGHash {
      private HashMap <String, Kanin> alle = new HashMap < >();
      public void settInn(Kanin en) {alle.put(en.hentNavn(), en);}
      public Kanin taUt(String navn ) { return alle.remove(navn); }
}
```

```java
import java.util.HashMap;

class Kanin{
  private String navn;
  Kanin(String nv) {navn = nv;}
  public String hentNavn() {return navn;}
}

class EnkelKaninGHash {
      private int antall = 0;
      private HashMap <String, Kanin> alle = new HashMap <String, Kanin>();
      public void settInn(Kanin peker) {alle.put(peker.hentNavn(), peker);}
      public Kanin taUt(String navn) {return alle.remove(navn); }
}
```

```java
class  KaningardTestHash {
  public static void main (String [ ] args) {
    EnkelKaninGHash mittBur = new EnkelKaninGHash();
    Kanin kalle = new Kanin("Kalle");
    mittBur.settInn(kalle);
    Kanin sprett = new Kanin("Sprett");
    mittBur.settInn(sprett);
    // Tester at en som er satt inn kan tas ut
    Kanin enKanin = mittBur.taUt("Kalle");
      test (((enKanin != null) &&  enKanin.hentNavn().equals("Kalle")), 1);
    // Tester at den som er tatt ut virkelig er ute
    enKanin = mittBur.taUt("Kallle");
      test ((enKanin == null),2);
    // . . .
  }
   static void test(boolean riktig, int testNr) {
      if (riktig) {
          System.out.println("Riktig  test nummer " + testNr);
      }  else {
        System.out.println("Feil test nummer " + testNr);
    }
  }
}
```

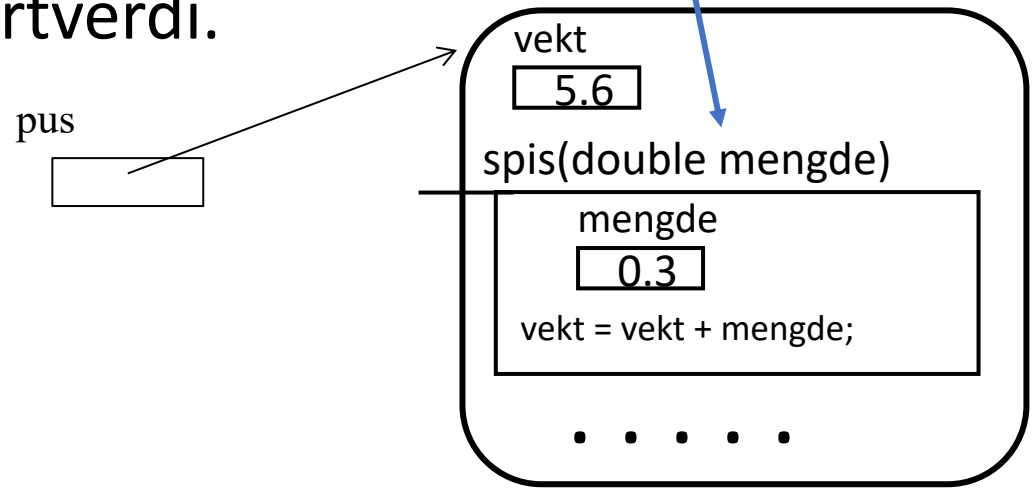
Enhetstesting - oppsummering

- Først: Tenk og planlegg og tenk og planlegg når du programmerer
- Små deler av programmet må testes før de settes sammen til et større program
- En slik liten del kan være et objekt / en klasse
- Intern testing vs. ekstern testing (se på grensesnitt vs. implementasjon)
- Du bør særlig teste grensetilfeller
  - F.eks: Ta ut av tom beholdere – sette inn i full beholder
- Du bør teste at alle metoder som forandrer tilstanden til objektet virker slik du ønsker – men umulig å teste alle tilfeller
    - Kanskje skrive ut alle private variable før og etter et slikt metodekall (toString())
- Du bør teste at alle observator-metodene observerer det du ønsker

Parameteroverføring

- En parameter i en metodedeklarasjonen kalles en **formell parameter**
  - En del av signaturen til metoden
- Den **aktuelle parameteren** på kallstedet er et **uttrykk** som evalueres til en verdi i det metoden kalles
- Denne verdien tas med til metoden og det opprettes en metodeinstans som inneholder den formelle parameteren som lokal variabel og med denne verdien som startverdi.

```
double fisk = 0.2;
double salat = 0.1;

pus.spis(fisk+salat);
```



Parameteroverføring i Java

- Gjelder også for parametre til konstruktører
- Alltid "by value" (verdien av uttrykket tas med til metodeinstansen)
- Referanser/objekter blir da "by reference" (en referanseverdi tas med til metodeinstansen)

Med andre ord:

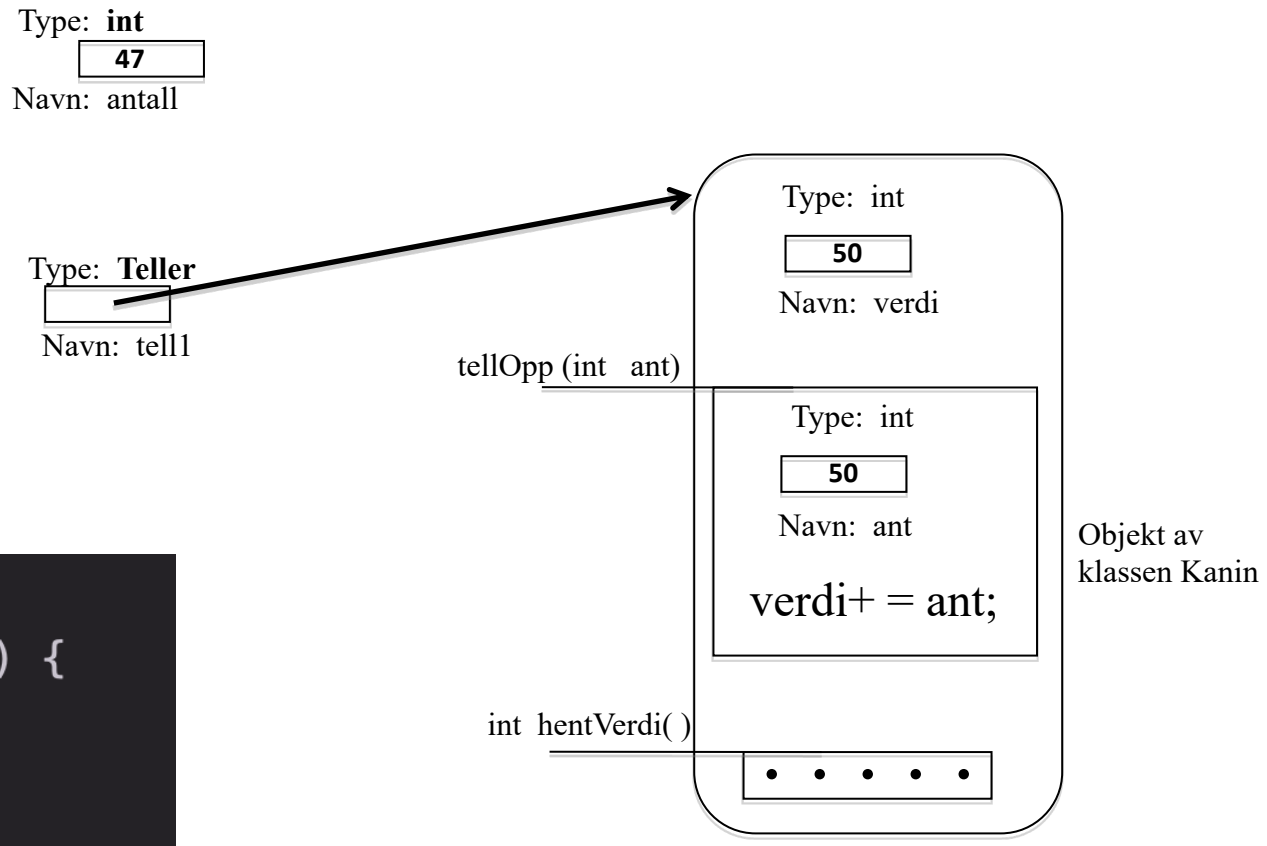
1. Regn ut verdien av uttrykket på kallstedet (aktuell parameter)
2. Opprett en metodeinnstans og ta med denne verdien til metoden
3. La den formelle parameteren i metoden bli en lokal variabel i metoden
4. Verdien som er tatt med blir startverdien til den lokale variablen

To eksempler til på de neste sidene. En konstruktør tar parametre akkurat som en vanlig metode.

Flere eksempler: Parameteroverføring - int


```
class Teller {
      private int verdi = 0;
      public void tellOpp(int ant) {
            verdi = verdi + ant;
       }
      public int hentVerdi( ) {
            return verdi;
      }
}
```

```
class BrukTeller2 {
     public static void main (String [ ] arg) {
            Teller tell1 = new Teller( );
            int antall = 47;
            tell1.tell(antall+3);
      }
}
```

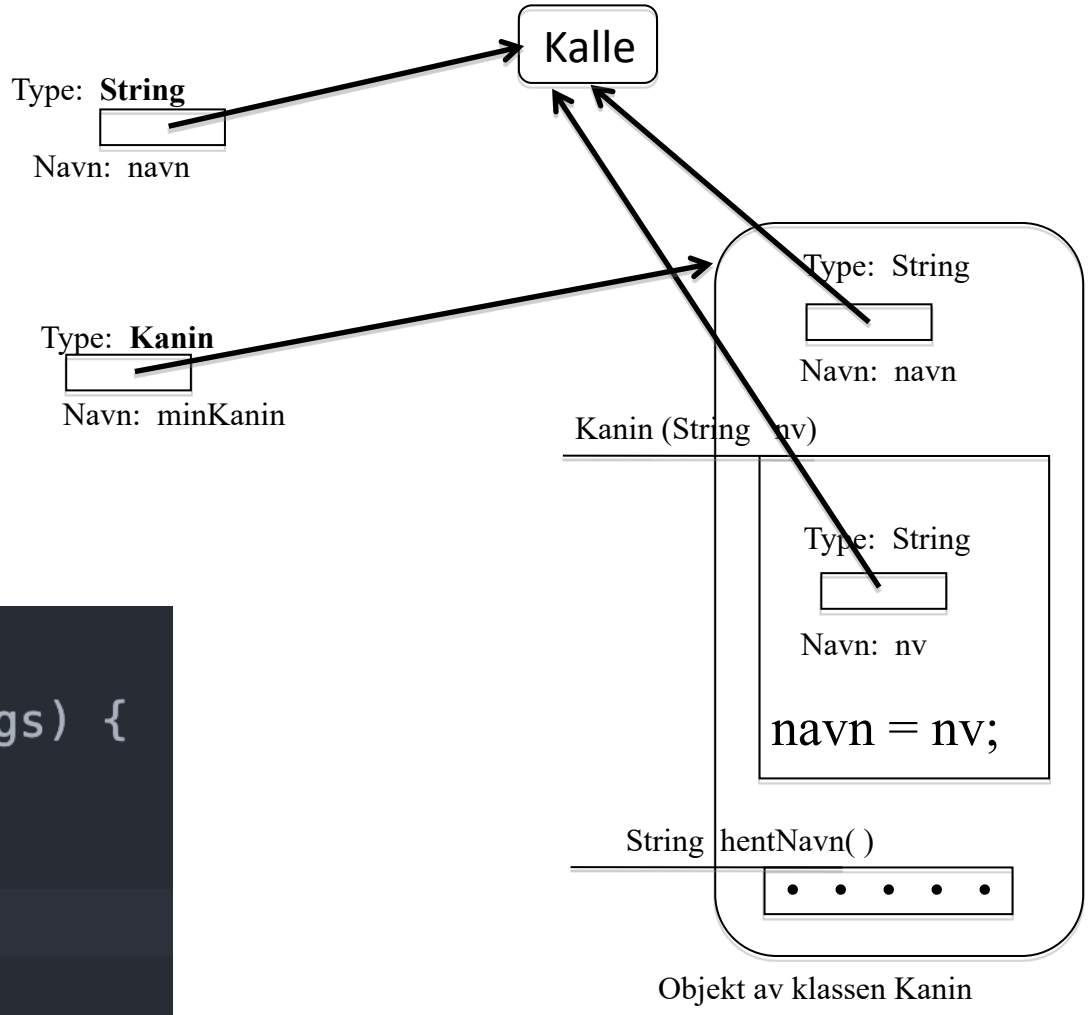


Nå har dere sett så mange klassedatastrukturer med main-metoden, så nå tegner vi det litt enklere

Parameteroverføring - referanse

```
class Kanin{
      private String navn;
      public Kanin(String nv) {
            navn = nv;
      }
      public String hentNavn()  {
          return navn;
      }
}
```

```
class BrukKanin {
    public static void main(String[]args) {
        String navn = "Kalle";
        Kanin minKanin;
        minKanin = new Kanin(navn);
    }
}
```



Nå har dere sett så mange klassedatastrukturer med main-metoden, så nå tegner vi det litt enklere

I dag har vi lært

- Enhetsestesting er viktig i programmering
  - Testing er viktig i tillegg til nøyaktig programmering.
- Nå kan dere «alt» om klasser og objekter (og typer og variabler)
  - Etter denne uken må dere kunne Java like bra som dere kan Python (Horstmann tom. kap 8)
- Dere kjenner til og kan bruke arrayer (innebygget i Java (og i datamaskiner)) og HashMap og ArrayList fra Javabiblioteket.

- Trøst: Neste uke begynner vi «sakte» på subklasser og arv.

Enda en trøst (?)

Jo før du blir hengende etter -

jo mer tid har du til å ta igjen det tapte