UNIVERSITETET I OSLO

Institutt for informatikk

IN1010 våren 2022
25. 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

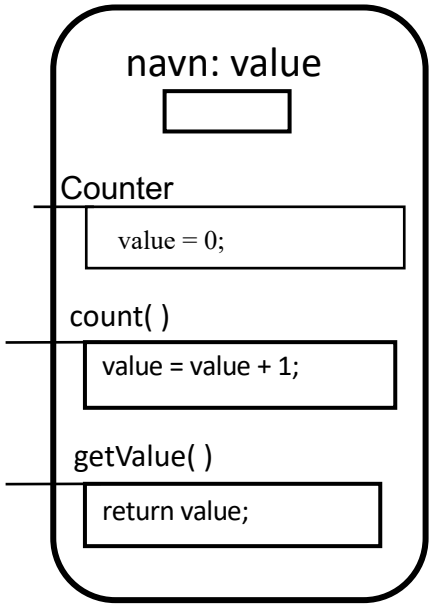
- Hvordan representeres objekter i minnet
- Mer om objekter og innkapsling
  - Enhetstesting
- Mer om arrayer
- Om beholdere i Javas bibliotek:
  - HashMap
  - ArrayList

UNIVERSITETET I OSLO

OO-programmering

Institutt for informatikk

- Objekter er levende (Kristen Nygaard)
- I noen OO-språk sender man meldinger mellom objekter

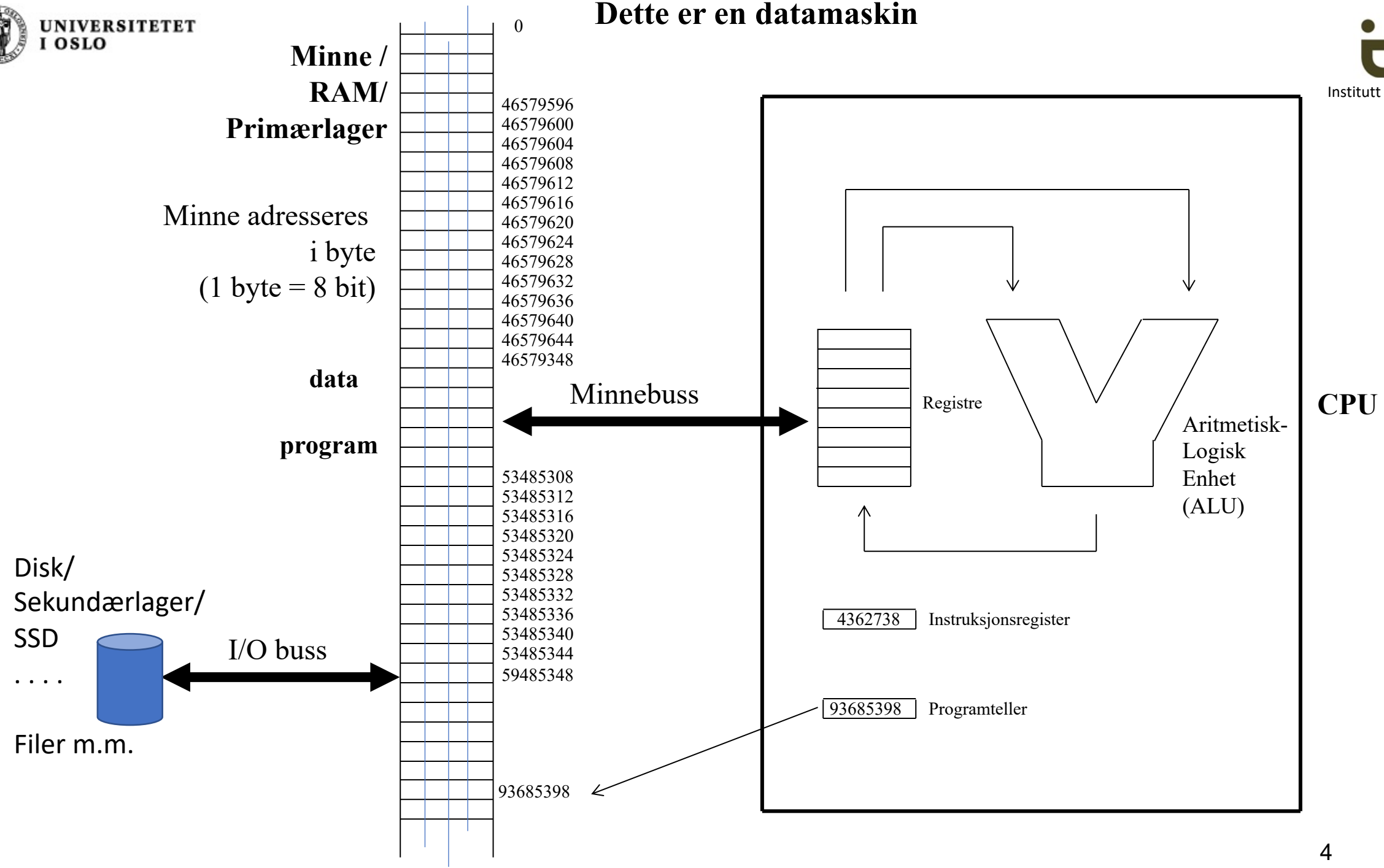


Et **objekt** av klassen Counter

- **Variable og konstanter - "DATA"**
  Kristen Nygaard: SUBSTANS
- **Metoder – handlinger**
- **Eksternt blikk: public-metoder**
- **Internt blikk: implementasjon**

```
class Counter {
    private int value;
    public Counter() {
        value = 0;
    }
    public void count( ) {
        value = value + 1;
    }
    public int getValue( ) {
        return value;
    }
}
```

Dette er en datamaskin



Sist snakket vi om Javas primitive/basale **verdier**

Institutt for informatikk

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	

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

Type: int Navn: tall

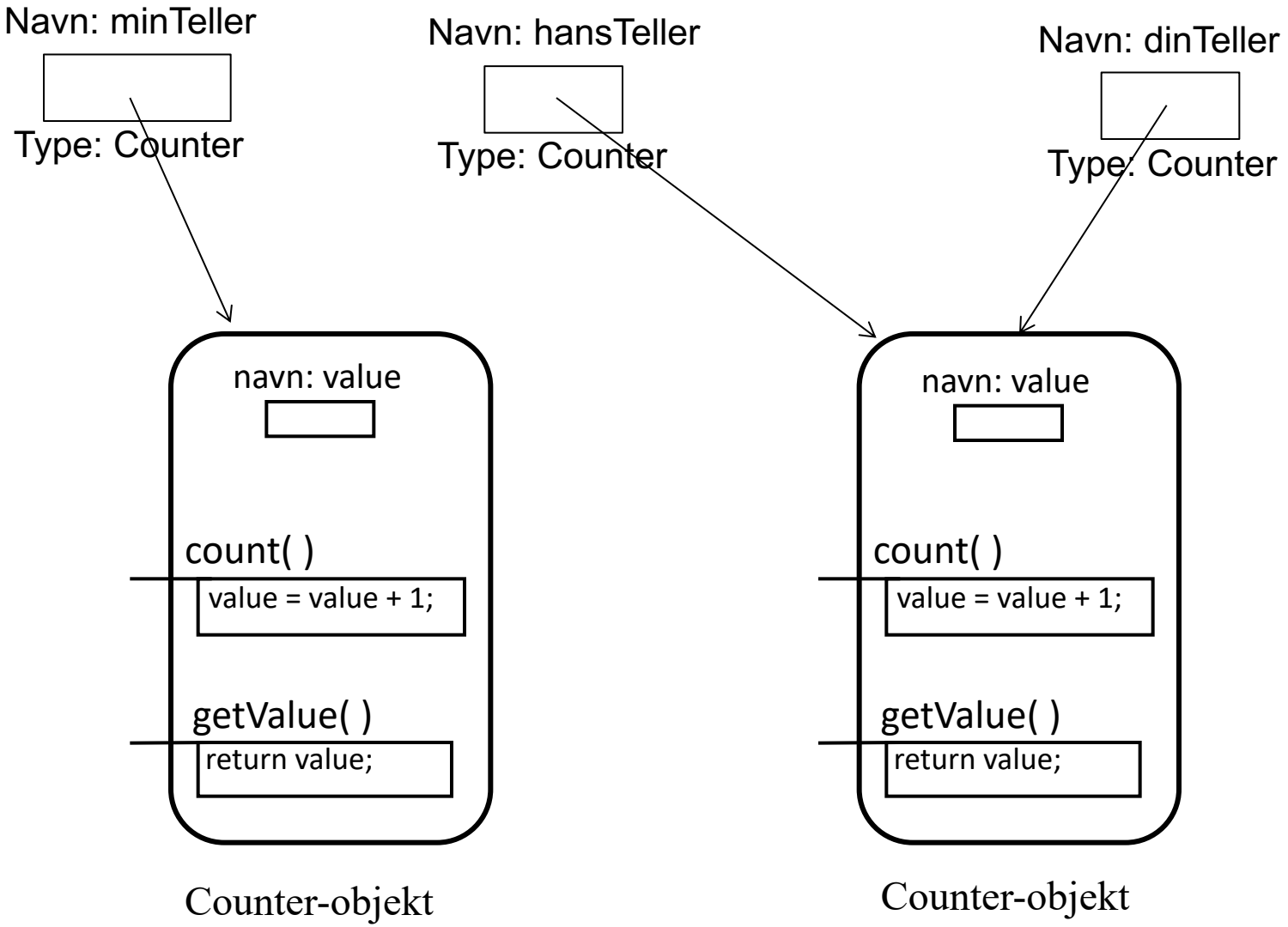
00000000000000000000000000001010

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~~

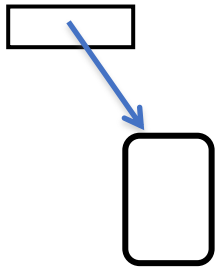
Nå skal vi snakke mer om referanseverdier



Hva er en referanse / peker

Python:

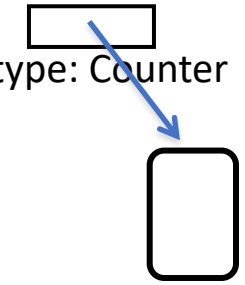
navn: minTeller



Java:

navn: minTeller

type: Counter



F.eks. deklarasjonen: Counter minTeller = new Counter();

En variabel: som innholder verdien:

navn: minTeller type: Counter

00000110010100000010000001100100000011001010000001000000 1100111

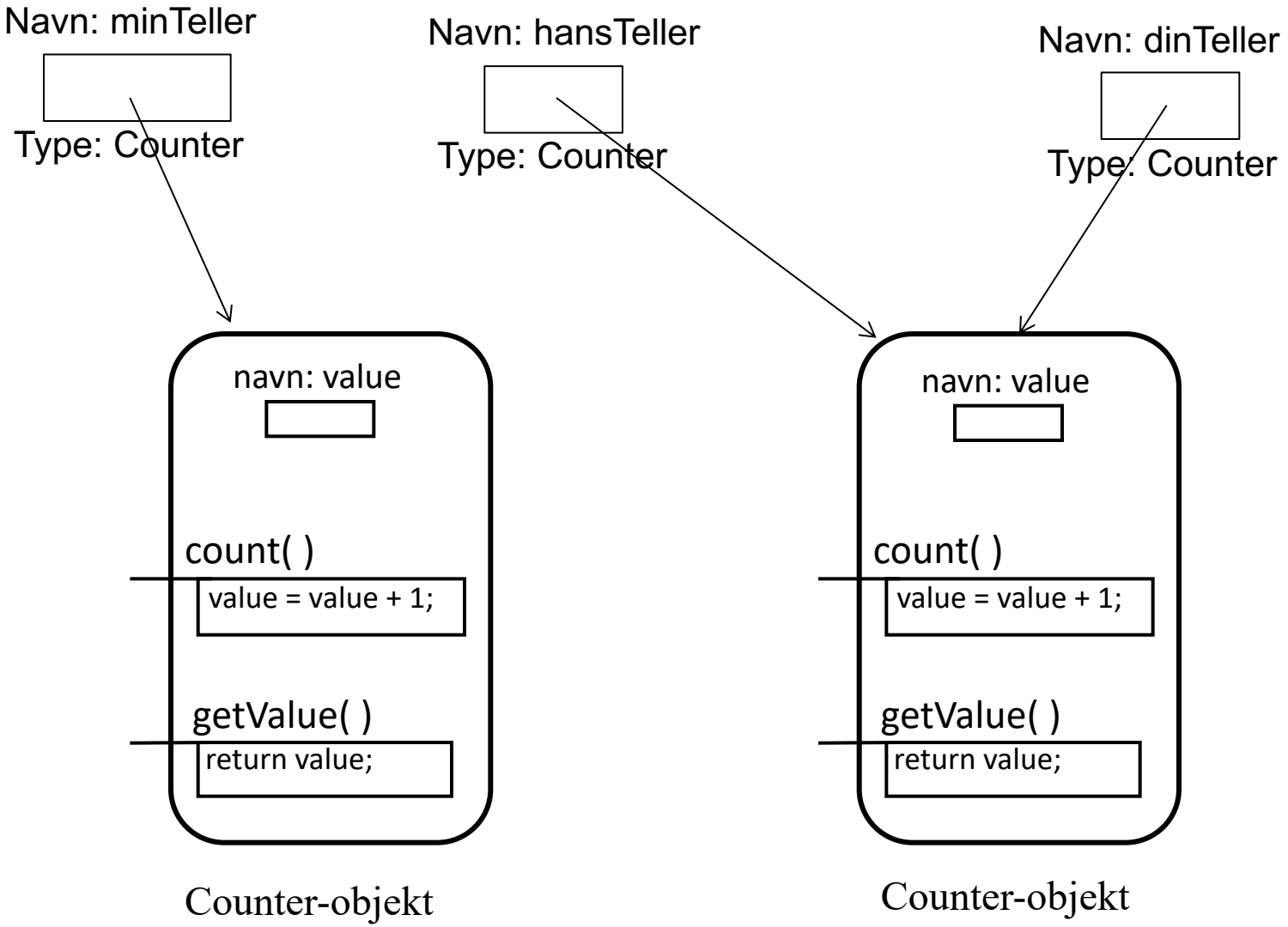
En variabel er et sted i minnet

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** Counter-objekt

Mer om referanser



Referanser

Navn: minTeller

53485320

Type: Counter

Navn: hansTeller

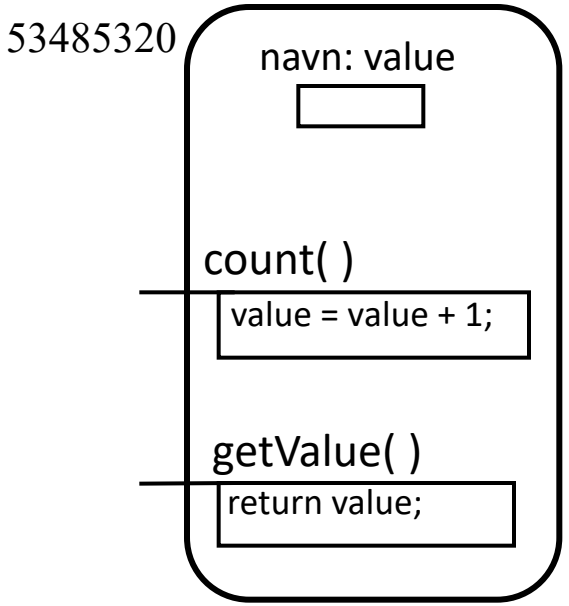
9483567

Type: Counter

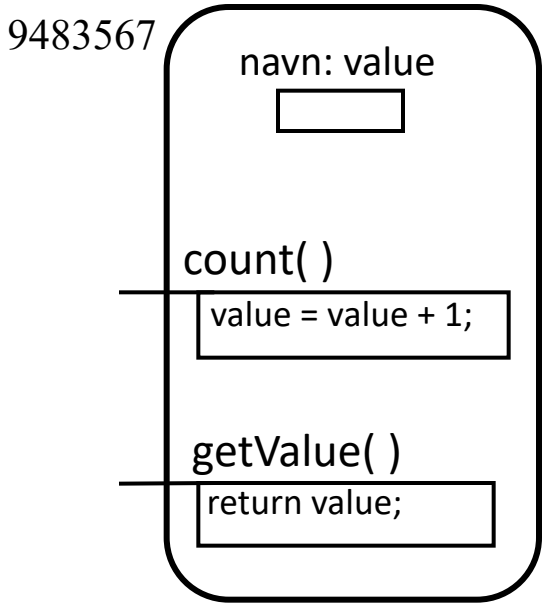
Navn: dinTeller

9483567

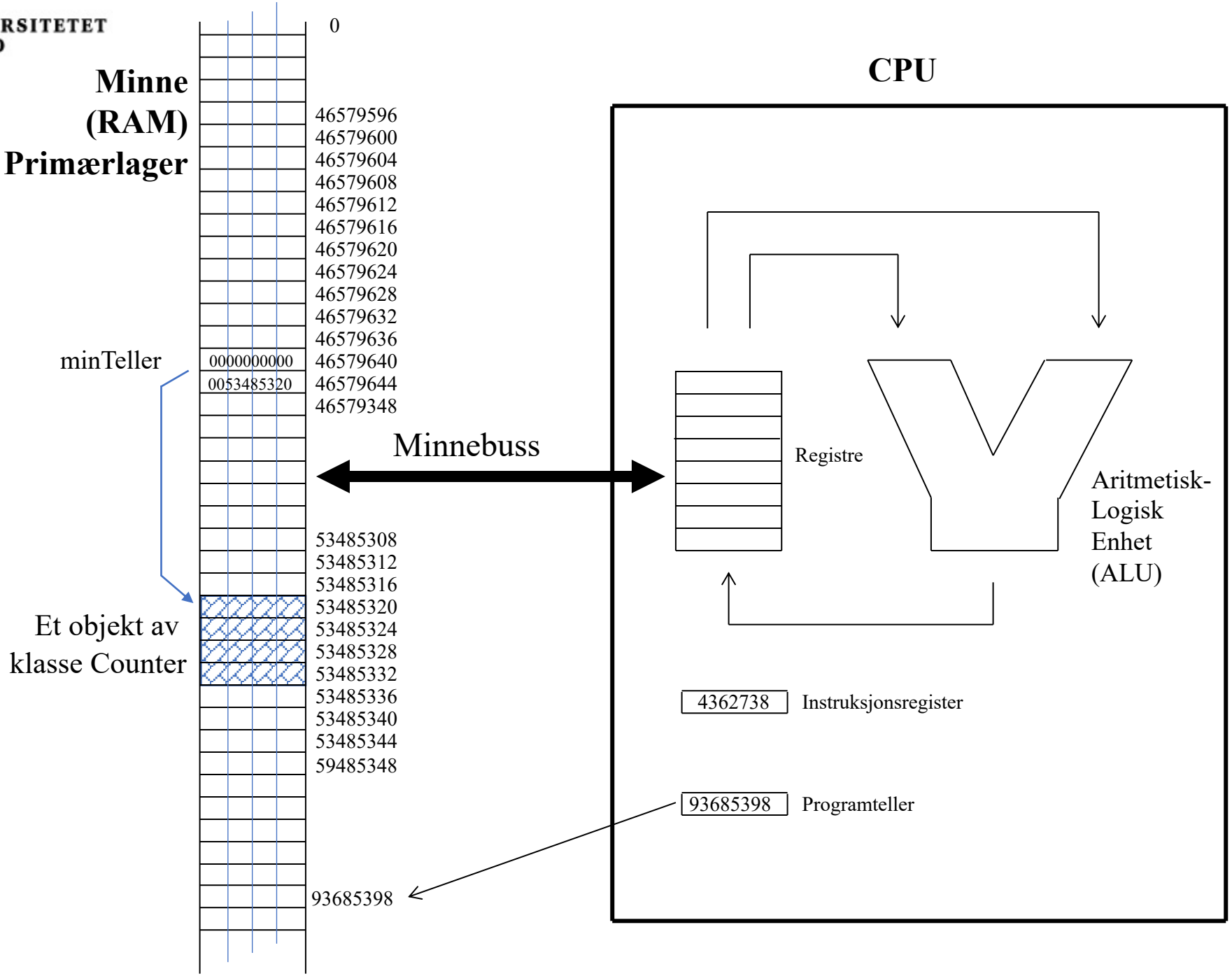
Type: Counter



Counter-objekt



Counter-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
klasse Counter
93685398
CPU
Registre
Aritmetisk-
Logisk
Enhet
(ALU)
4362738
Instruksjonsregister
93685398
Programteller

Javas **uttrykk**

- **Uttrykk** (expressions) 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;

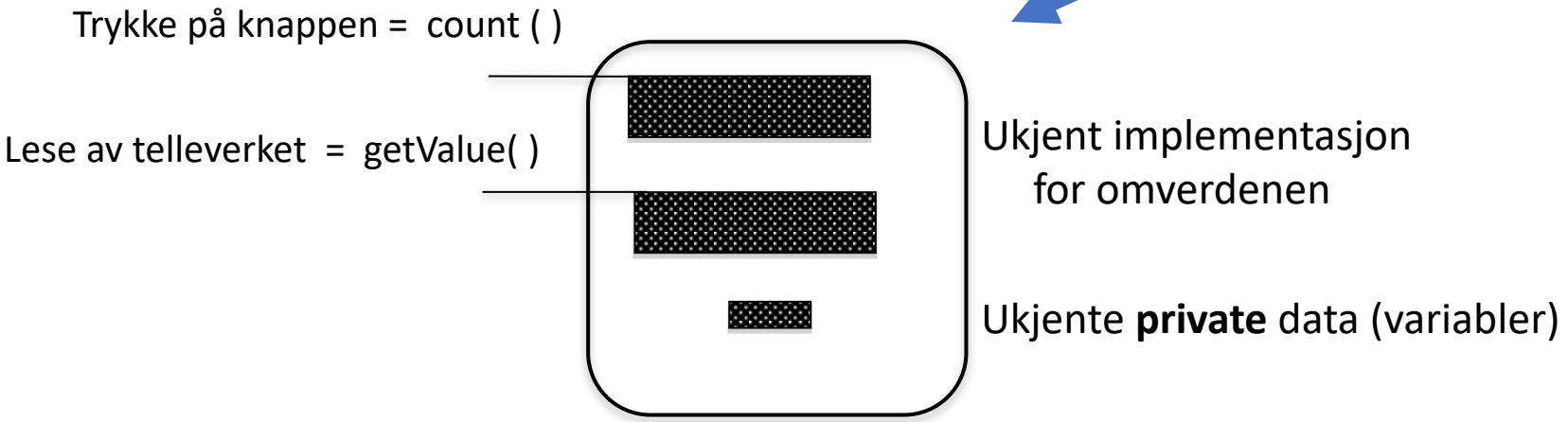
- Husk at en aktuell parameter (argument) er et uttrykk
  - som evalueres til en verdi av en gitt type
  - Denne verdien blir startverdien til den formelle parameteren (en lokal variabel)
  - Se til slutt i dag

Objektorienter (OO) programmering

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

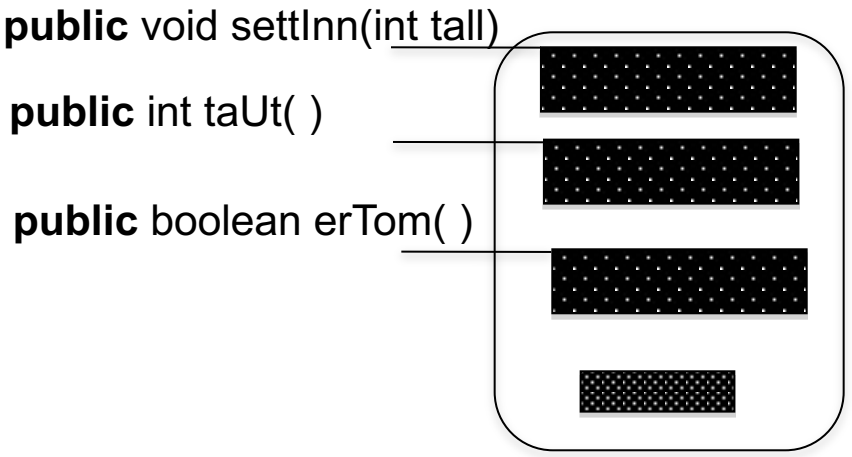
Hva er (objektorientert) programmering ?

Forrige uke: Et objekt er en sort boks + *grensesnittet* mot omverdenen



Et objekt kan også være en hjelpekonstruksjon i programmet

Et objekt som tar vare på ett tall
(kan bli en mer "intelligent variabel").



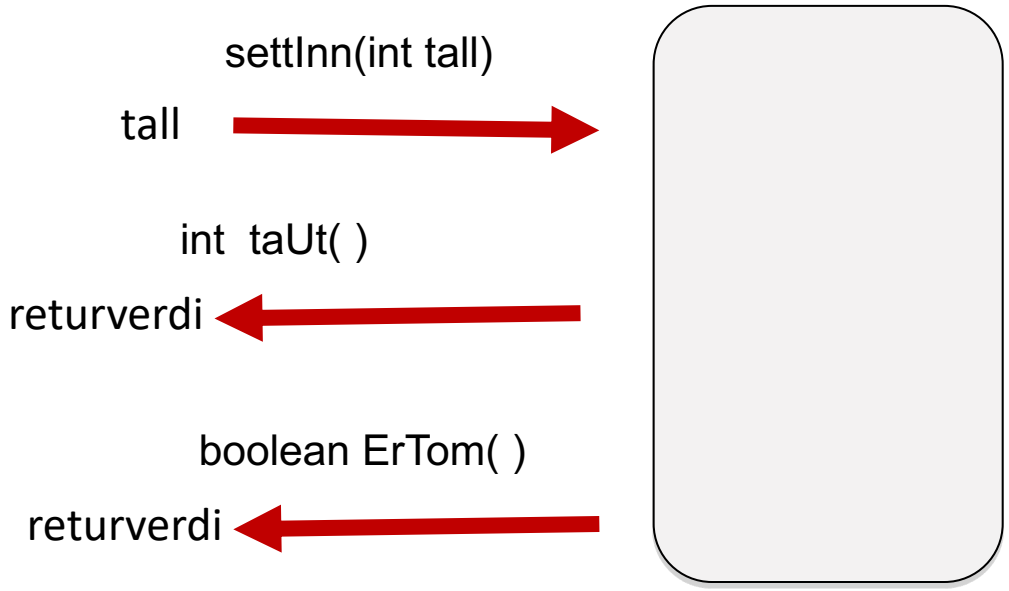
Ukjent implementasjon av metode

Ukjent implementasjon av metode

Ukjent implementasjon av metode

Ukjente **private** data og ukjente **private** metoder

En sort boks som tar vare på ett tall



Hvordan virker settInn hvis det er et tall der fra før?

Hvordan virker taUt hvis det ikke er noe tall i boksen?

. . .

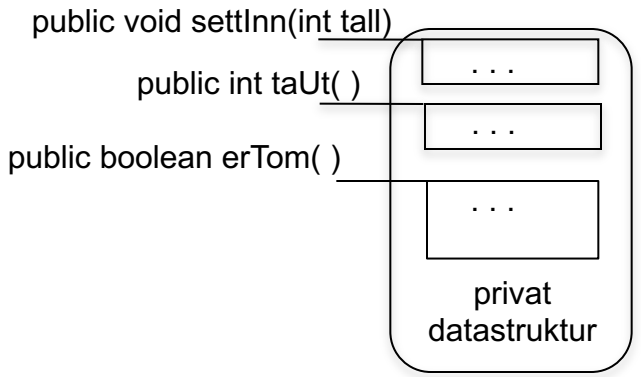
Grensesnitt / tjenester

Hvilke metoder trengs ?
Hva er deres parametre ?
Hvordan skal disse metodene virke ?

Vi lager en klasse som vi kan lage objekter av:

```
class EnkelHeltallsbeholder {
    private . . . . ;

    public void settInn(int tall) {
        . . . . .
    }
    public int taUt( ) {
        . . . . .
    }
    public boolean erTom ( ) {
        . . . . .
    }
}
```

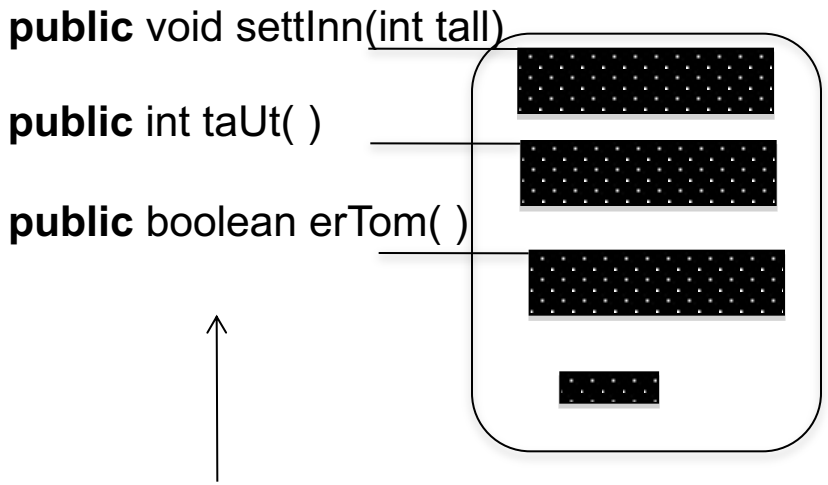
new EnkelHeltallsbeholder()
gir dette objektet:



Objekt av klassen
EnkelHeltallsbeholder

Denne koden kan *nesten* kompileres (og kjøres)!

En metodes *signatur*

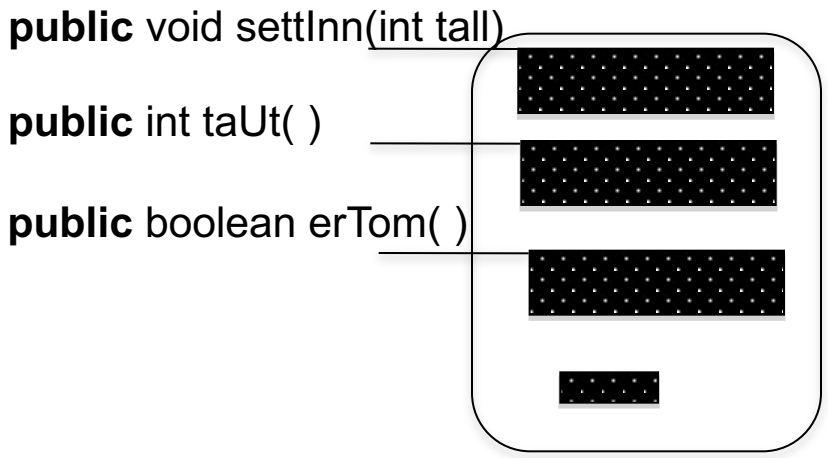


Dette kaller vi metodenes **signaturer** (skrivemåte, syntaks)

Signaturen til en metode er

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

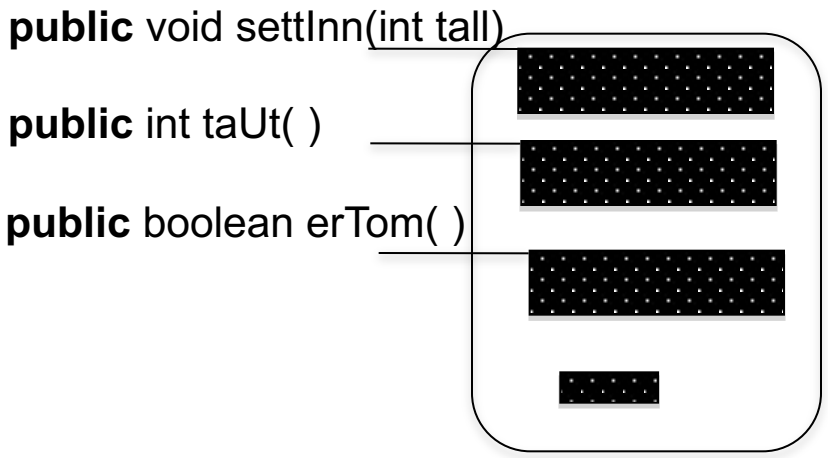
Metodenes **semantikk**



Hva gjør disse metodene? Hvordan virker de? Hva er sematikken til metodene? Hvilke tjenester tilbyr de?

Semantikk betyr virkemåte

Metodenes **semantikk**



Dette er spesifikasjonen av tjenestene objektet tilbyr, **dvs. utgangspunktet for programmeringen av klassen**

- Forslag til semantikk:
  - Metoden "settInn" gjør at objektet tar vare på tallet som er parameter til metode. Hvis det er et tall i objektet fra før, har metoden ingen virkning.
  - Metoden "taUt" tar ut av objektet det tallet som tidligere er satt inn. Metoden returnerer det tallet som slettes.
  - Metoden "erTom" returnerer sann om objektet er tomt, usann ellers.

- Informatikkens 3. lov: ☺

1. Først betemmer vi semantikken og signaturene til metodene
2. Deretter implementerer vi metodene
   samtidig som vi bestemmer oss for 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

Testing -- Enhetstesting

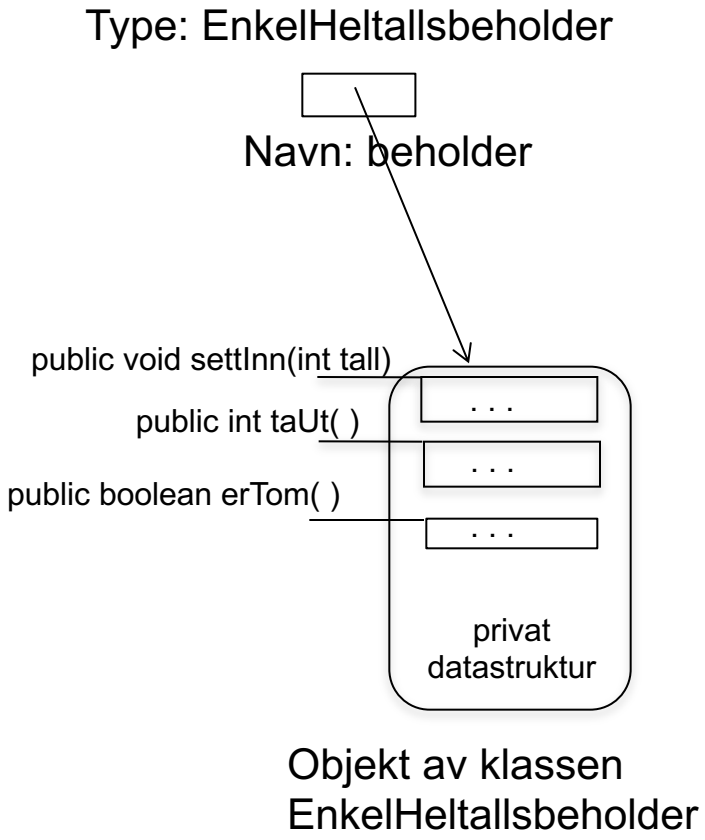
- Når vi planlegger og skriver programmer prøver vi å overbevise oss selv (og dem vi skriver sammen med) at den koden vi skriver kommer til å utføre det vi ønsker (oppfylle semantikken)
- Men vi kommer alltid til å tenke og skrive feil
- Derfor må vi **teste** programmene våre
- Objektorientering / modularisering:
  - 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 5. april

Om å teste et objekt (som tar vare på ett tall)


```
class EnkelHeltallsbeholder {
    private . . . . ;

    public void settInn(int tall) {
        . . . . .
    }
    public int taUt( ) {
        . . . . .
    }
    public boolean erTom ( ) {
        . . . . .
    }
}
```

```
class TestAvBeholder {
    public static void main (String[ ] args){
        EnkelHeltallsbeholder beholder =
                    new EnkelHeltallsbeholder();
        . . .
        . . .
    }
}
```

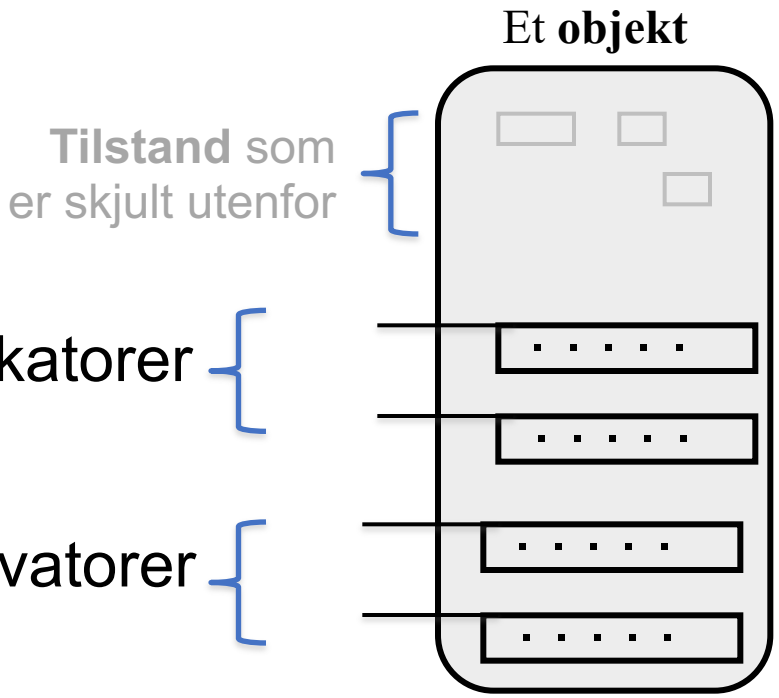


Når skal vi skrive testprogrammet?

Om du er alene?
Om dere er flere?

Modifikatorer og Observatorer

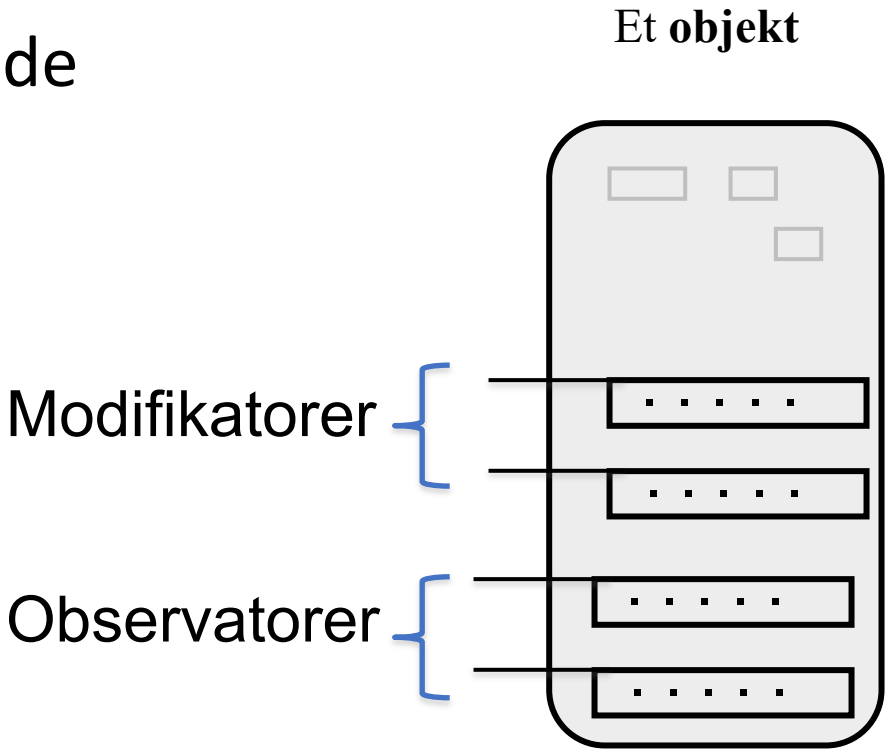
- En modifikator-metode forandrer tilstanden til et objekt
- En observator-metode leser av tilstanden uten å forandre den



Modifikator, f.eks. settInn(), Observator, f.eks. les()

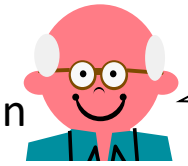
Testing - - Enhetstesting

- 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

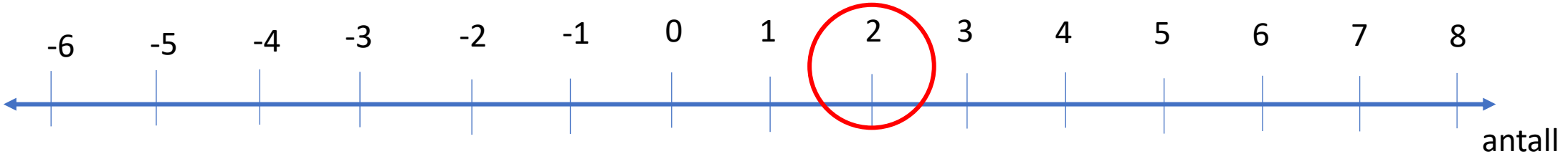
Institutt for informatikk

En liten digresjon:

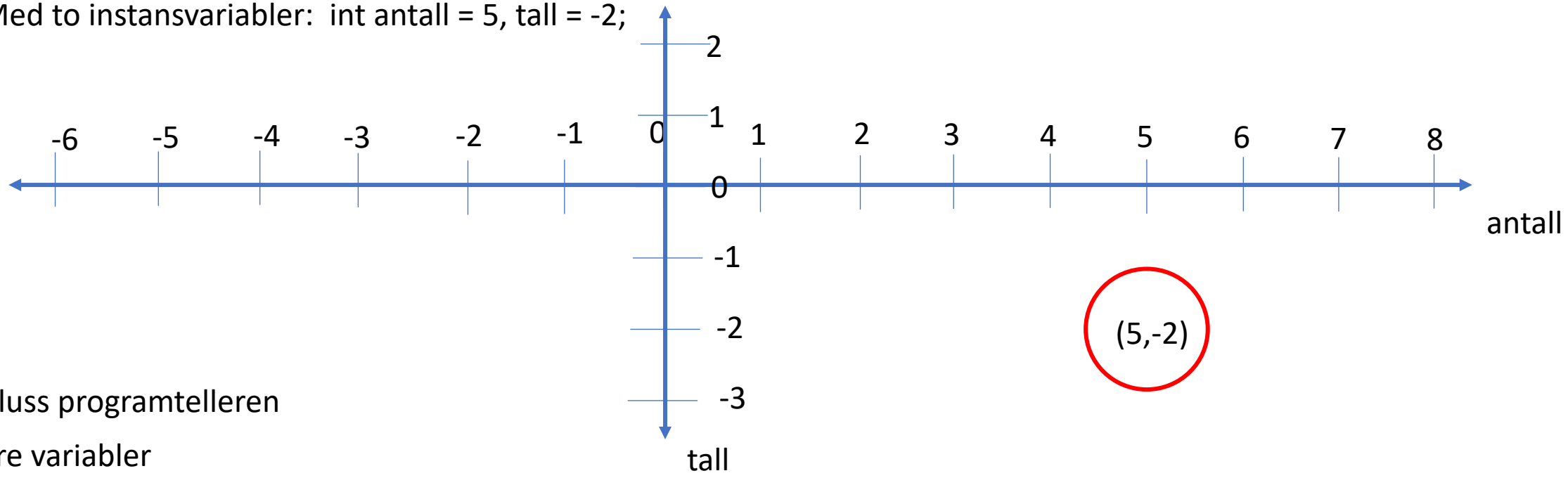
Tillstanden (til et objekt) - tilstandsrommet

(Et litt matematisk syn)

- Med én instansvariabel: int antall = 2;



- Med to instansvariabler: int antall = 5, tall = -2;



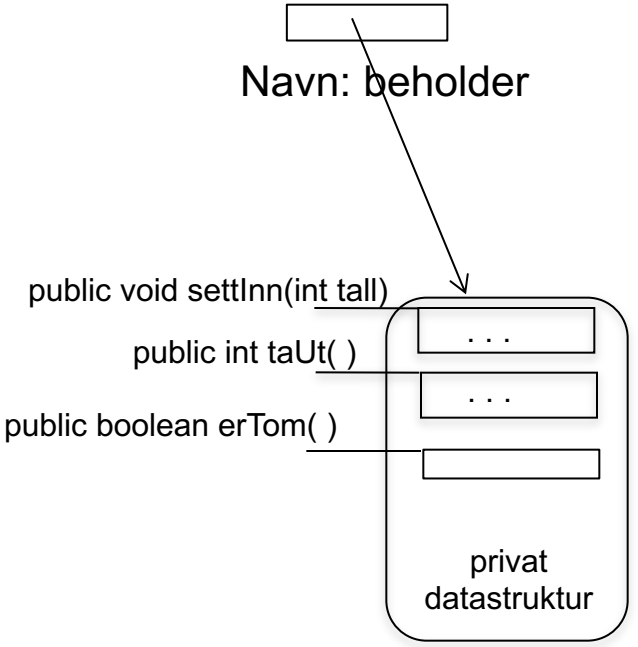
- Pluss programtelleren
- Tre variabler
- Fire variabler . . . . .

Veldig enkel test av beholder


```
class EnkelHeltallsbeholder {
    private int tallet;
    private boolean tom = true;
    public void settInn(int tall) {
        if (tom) tallet = tall;
    }
    public int taUt( ) {
        tom = true;
        return tallet;
    }
    public boolean erTom ( ) {
        return tom;
    }
}
```

```
class VeldigEnkelTestAvBeholder {
    public static void main (String[ ] args){
        EnkelHeltallsbeholder beholder =
                    new EnkelHeltallsbeholder();
        beholder.settInn(17);
        if (beholder.taUt() == 17)
                {System.out.println ("Riktig");}
        else {System.out.println("Feil");}
      }
}
```

Type: EnkelHeltallsbeholder



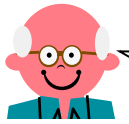
Objekt av klassen
EnkelHeltallsbeholder

```
>java VeldigEnkelTestAvBeholder
Riktig
>
```

Enkel enhetstesting av heltallsbeholder


```
public class EnkelTestAvHeltallsbeholder {
   public static void main (String[ ] arg) {
       EnkelHeltallsbeholder beholder =
                 new EnkelHeltallsbeholder();
       beholder.settInn(17);
       if (beholder.taUt() == 17)
            {System.out.println ("Riktig 1");}
       else {System.out.println("Feil 1");}
       beholder.settInn(18);
       beholder.settInn(17);
       if (beholder.taUt() == 18)
            {System.out.println ("Riktig 2");}
       else {System.out.println("Feil 2");}
       if (beholder.erTom())
            {System.out.println ("Riktig 3");}
       else {System.out.println("Feil 3");}
       beholder.settInn(19);
       if (! beholder.erTom())
            {System.out.println ("Riktig 4");}
       else {System.out.println("Feil 4");}
   }
}
```

```
class EnkelHeltallsbeholder {
     private int tallet;
     private boolean tom = true;
     public void settInn(int tall) {
         if (tom) tallet = tall;
     }
     public int taUt( ) {
         tom = true;
         return tallet;
     }
     public boolean erTom ( ) {
         return tom;
     }
}
```



Test en (modifikator)metode om gangen
Lag gode tekster for testene
Lag egne test-metoder

```
>java EnkelTestAvHeltallsbeholder
Riktig 1
Feil 2
Riktig 3
Feil 4
>
```

Med Javadoc (og uten? feil)


```
/** Objekter av denne klassen tar vare på
 *  ett  heltall.
 *  Initielt er beholderen tom
 *
 * @author Stein Gjessing
 * versjon 21. januar 2022
 */
public class EnkelHeltallsbeholder {
      private boolean tom = true;
      private int tallet;
     /**
      * Gjør at objektet tar vare på tallet som
      * er parameter til metoden.
      * Hvis det allerede er lagret et tall i objektet,
      * dvs. at beholderen ikke er tom, har denne
      *  metoden ingen virkning
      *
      *@param tall  tallet som objektet skal
      *    ta vare på
      */
      public void settInn(int tall) {
            if (tom) tallet = tall;
            tom = false;
      }
```

```
      /**
      * Sjekkeren om objektet er tomt
      *
      @return objektet er tomt
      */
      public boolean erTom ( ) {
            return tom;
      }
      /**
      * Tar ut av objektet det tallet objektet
      *   tar vare på.
      * Om objektet alt er tomt, returneres en
      *  ubestemt verdi.
      * Etter dette kallet er objektet tomt.
      *
     @return tallet som tas ut. eller en
      *     ubestemt verdi om objektet er tomt
      */
       public int taUt( ) {
             tom = true;
             return tallet;
       }
}
```

UNIVERSITETET I OSLO

Javadoc resultat

PACKAGE CLASS TREE INDEX HELP

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

Class Enkelheltallsbeholder

java.lang.Object
Enkelheltallsbeholder

```
public class Enkelheltallsbeholder
extends Object
```

Objekter av denne klassen tar vare på ett heltall. Initielt er beholderen tom

Constructor Summary

Constructors

Constructor	Description
`Enkelheltallsbeholder()`	

Method Summary

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

Modifier and Type	Method	Description
boolean	`erTom()`	Sjekkeren om objektet er tomt
void	`settInn(int tall)`	Gjør at objektet tar vare på tallet som er parameter til metoden.
int	`taUt()`	Tar ut av objektet det tallet objektet tar vare på.

Methods inherited from class java.lang.Object

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

Constructor Details

Enkelheltallsbeholder

```
public Enkelheltallsbeholder()
```

Method Details

settInn

```
public void settInn(int tall)
```

Gjør at objektet tar vare på tallet som er parameter til metoden. Hvis det allerede er lagret et tall i objektet, dvs. at beholderen ikke er tom, har denne metoden ingen virkning

Parameters:

`tall` - tallet som objektet skal ta vare på

erTom

```
public boolean erTom()
```

Sjekkeren om objektet er tomt

Returns:

objektet er tomt

taUt

```
public int taUt()
```

Tar ut av objektet det tallet objektet tar vare på. Om objektet alt er tomt, returneres en ubestemt verdi. Etter dette kallet er objektet tomt.

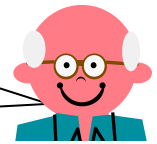
Returns:

tallet som tas ut. eller en ubestemt verdi om objektet er tomt

```
steing@eduroam-193-157-201-38 programmer % javadoc Enkelheltallsbeholder.java
Loading source file Enkelheltallsbeholder.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 ./Enkelheltallsbeholder.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@eduroam-193-157-201-38 programmer %
```

UNIVERSITETET I OSLO

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



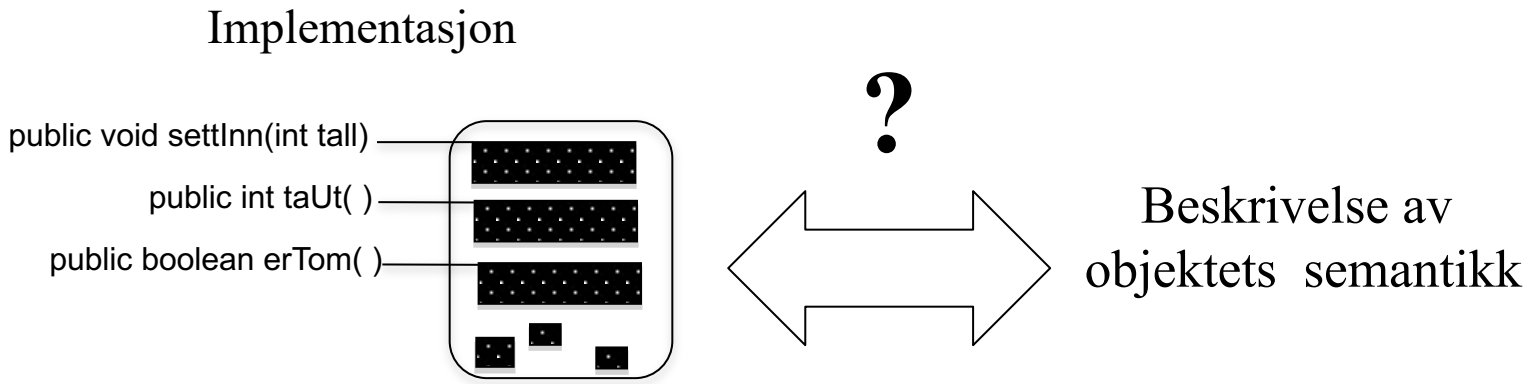
Institutt for informatikk

```
/** Objekter av denne klassen tar vare på
 *  ett  heltall.
 *  Initielt er beholderen tom
 *
 * @author Stein Gjessing
 * versjon 21. januar 2022
 */
public class Enkelheltallsbeholder {
      private boolean tom = true;
      private int tallet;
     /**
      * Gjør at objektet tar vare på tallet som
      * er parameter til metoden.
      * Hvis det allerede er lagret et tall i objektet,
      * dvs. at beholderen ikke er tom, har denne
      *  metoden ingen virkning
      *
      *@param tall  tallet som objektet skal
      *    ta vare på
      */
      public void settInn(int tall) {
            if (tom) tallet = tall;
            tom = false;
      }
      /**
      * Sjekkeren om objektet er tomt
      *
      @return objektet er tomt
      */
      public boolean erTom ( ) {
            return tom;
      }
      /**
      * Tar ut av objektet det tallet objektet
      *   tar vare på.
      * Om objektet alt er tomt, returneres en
      *  ubestemt verdi.
      * Etter dette kallet er objektet tomt.
      *
     @return tallet som tas ut. eller en
      *     ubestemt verdi om objektet er tomt
      */
       public int taUt( ) {
             tom = true;
             return tallet;
       }
}
```

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)

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( ) { . . . }
}
```

Semantikk (virkemåte):

- Hvis objektet 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 allerede er 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;
    }
}
```

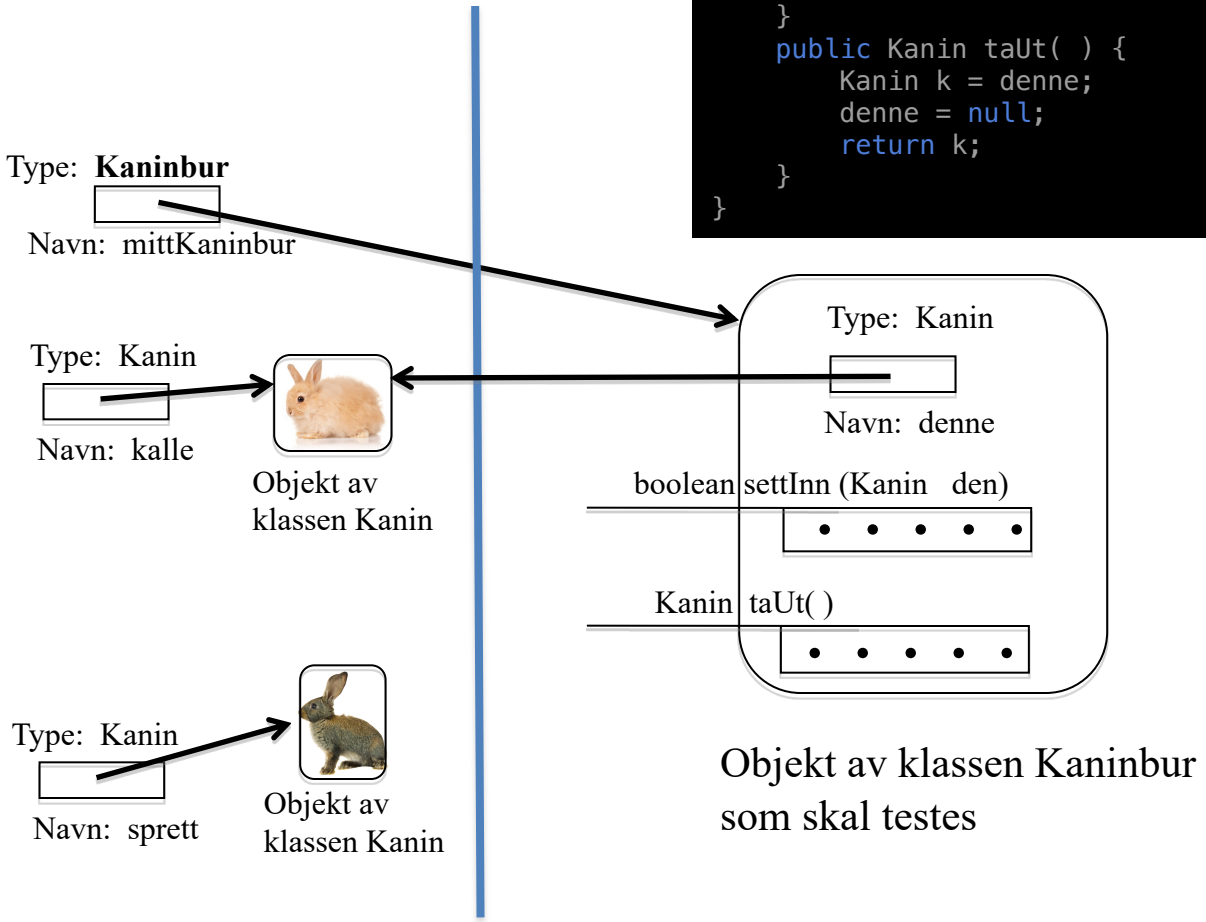
Enhetstesting av class Kaninbur

```
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

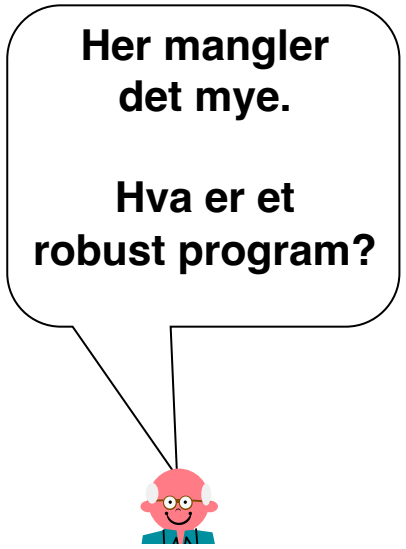
2 modifikatorer

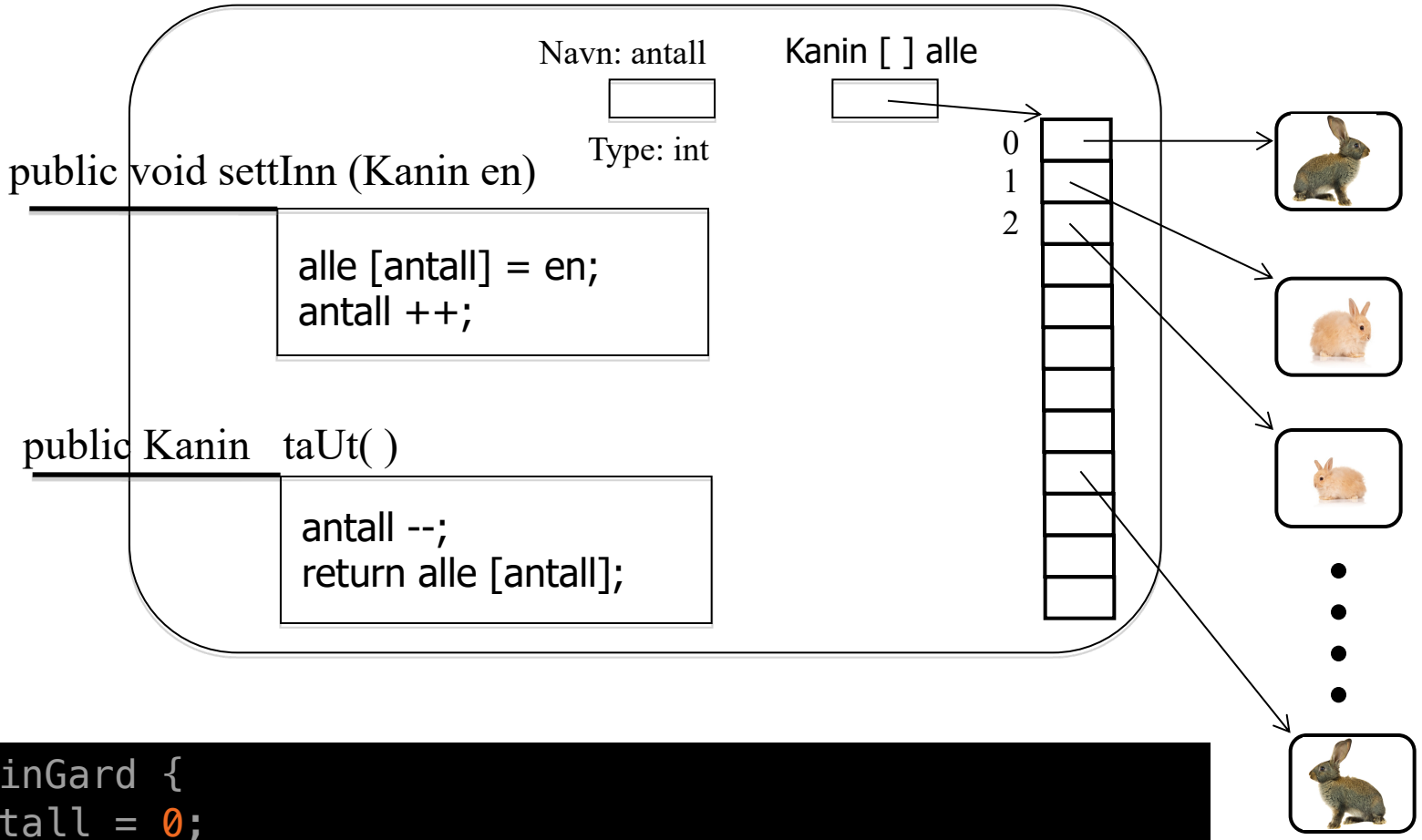
```
class    KaninGård {
   public boolean full( ) { . . . }
   public boolean tom ( ) { . . . }
   public Kanin  finnEn(String navn) { . . . }
   public void settInn (Kanin kn) { . . . }
   public void fjern(String navn) { . . . }
}
```

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

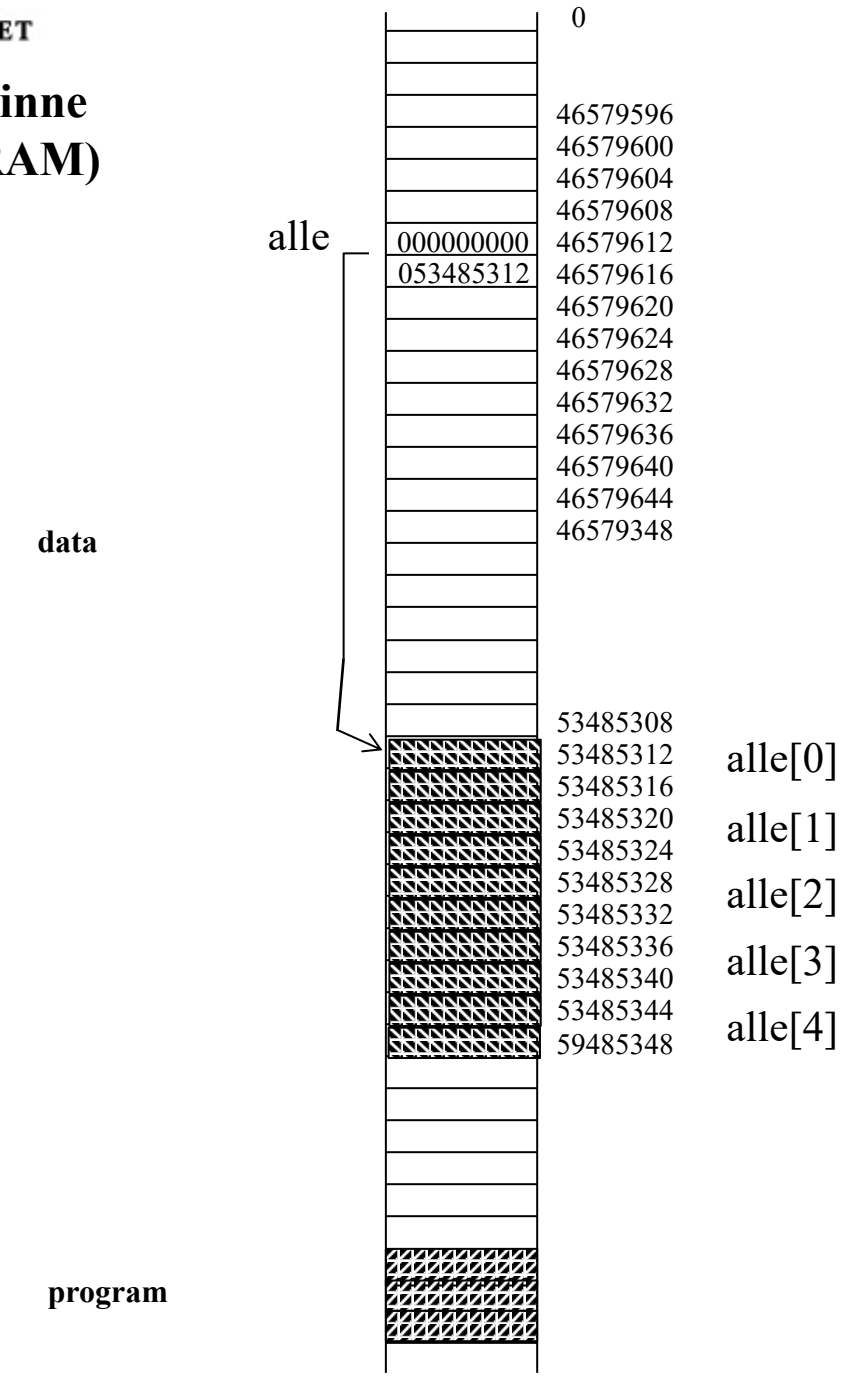
Objekt av klassen ForenkletKaninGård






```
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



Array er iboende i datamaskinen og i Java

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

Husk: Minnet er byte-adresserbart

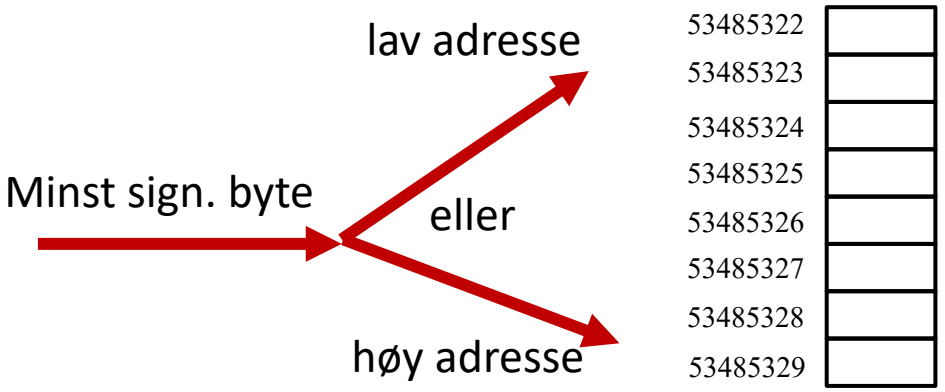
Her har jeg tegnet 4 byte = 32 bit på hver linje og alle er en array av referanser (á 64 bit)

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**

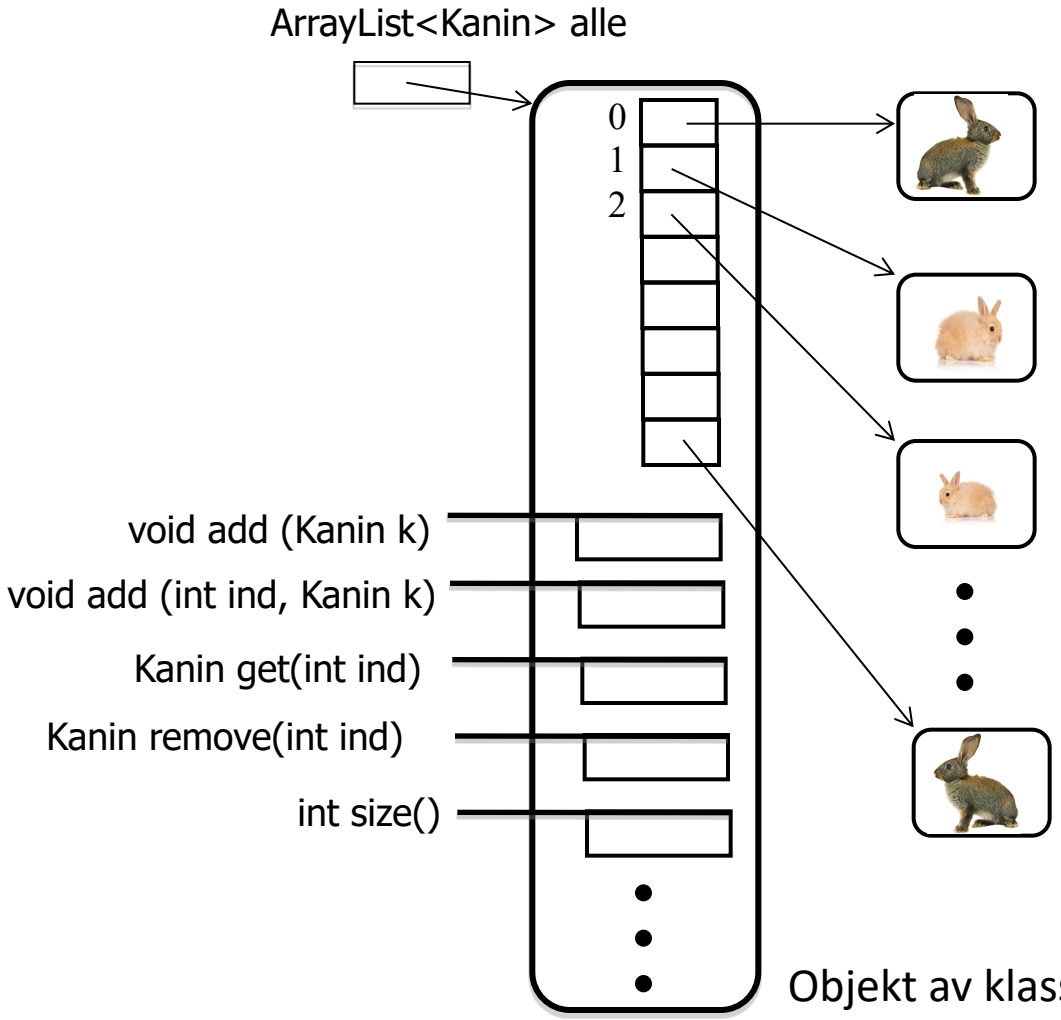
- ArrayList er en fleksibel array som utvider seg og trekker seg sammen etter behov
- ArrayList <Kaniner> mineKaniner = new ArrayList <Kaniner> ();
- Metoder: add, get, remove, . . . se Java-biblioteket

- 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

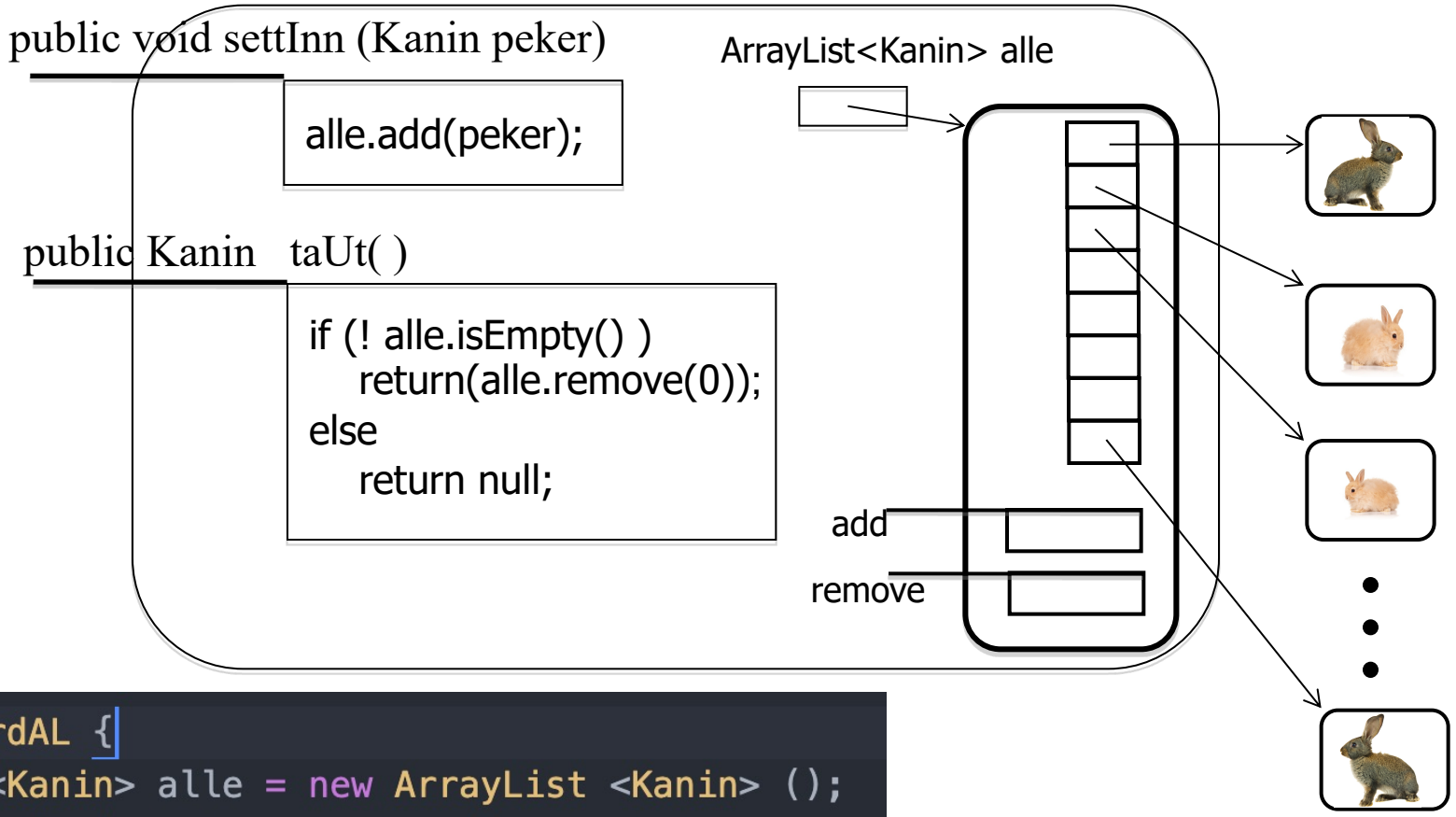
Deklarasjon:

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



Objekt av klassen ArrayList

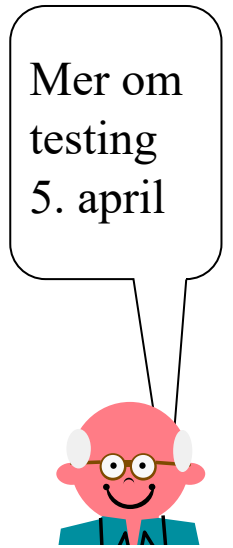
Objekt av klassen ForenkletKaninGardAL




```
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;
    }
}
```

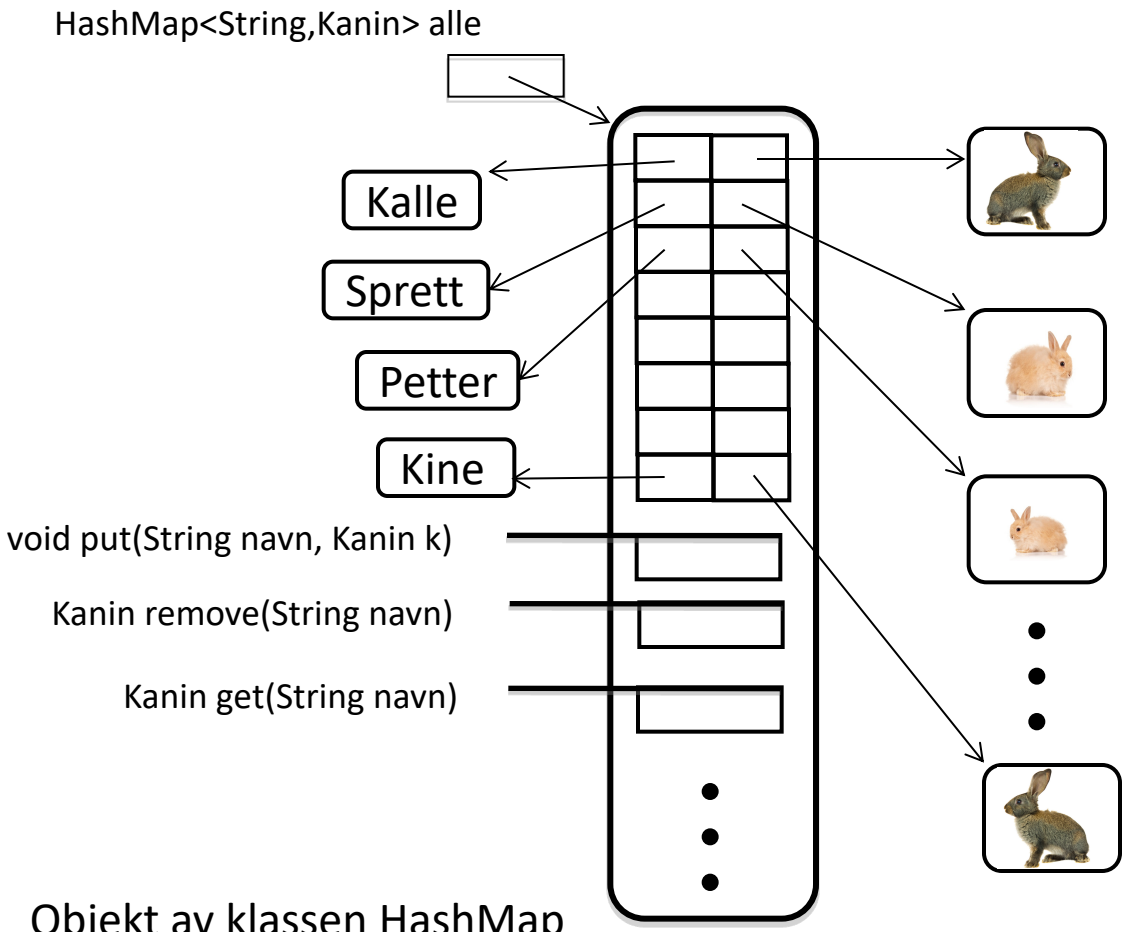

```java
import java.util.ArrayList;

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;
    }
}
```

```java
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);
    }
  }
}
```



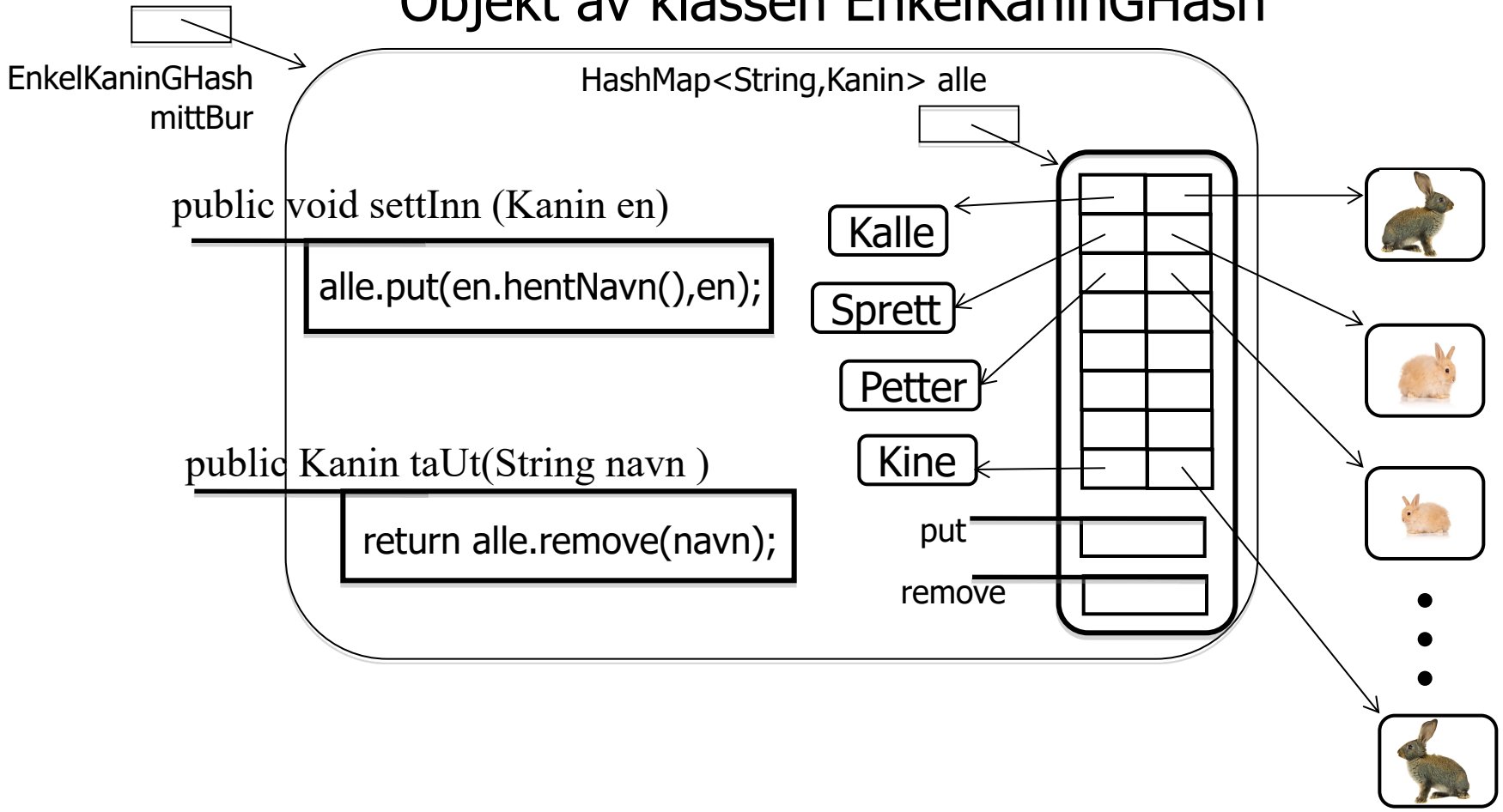
HashMap

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

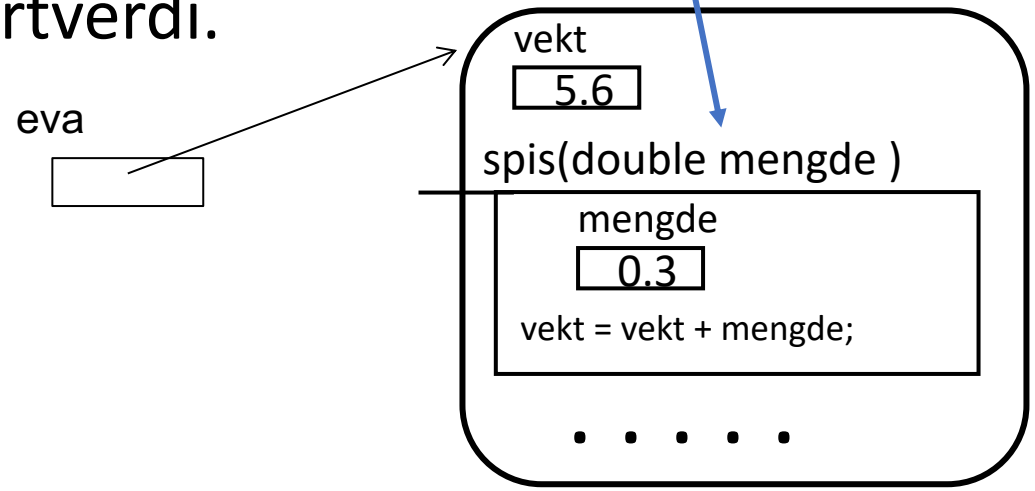
- Først: Tenk, tenk og tenk når du programmerer
- Små deler av programmet må testes før de settes sammen til et større program
- En slik liten del kan være et objekt / en klasse
- Intern testing vs. ekstern testing (grensesnitt vs. implementasjon)
- Du bør særlig teste grensetilfeller
  - F.eks: Ta ut 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;

eva.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:

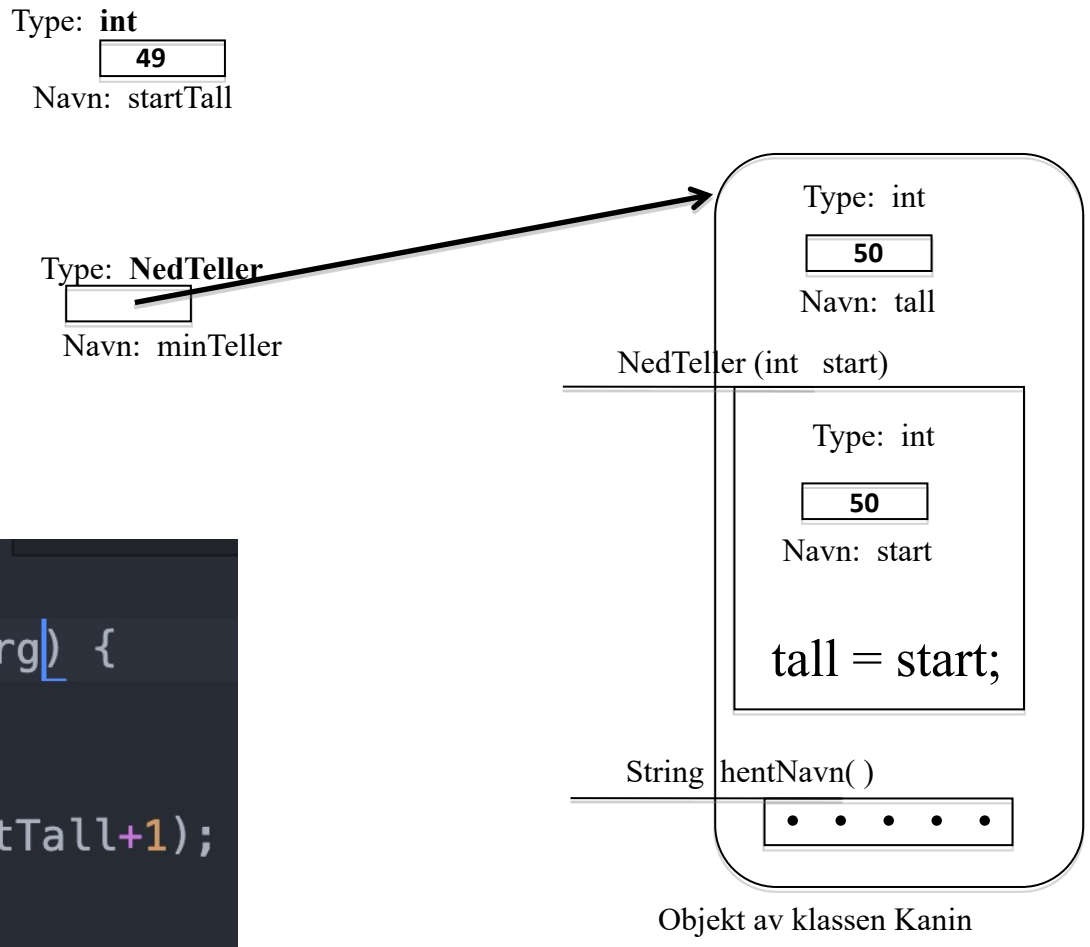
1. Regn ut verdien av uttrykket på kallstedet (aktuell parameter)
2. Opprett en metodeinnstans og ta med denne verdien til metoden
3. La den formelle parameteren i metoden bli en lokal variabel i metoden
4. Verdien som er tatt med blir startverdien til den lokale variablen

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

Flere eksempler: Parameteroverføring - int


```
class NedTeller {
  private int tall;
  public NedTeller(int start) {
        tall = start;
    }
    public void tellNed() {
      tall-- ;
    }
    public int hentTall()  {
      return tall;
    }
}
```

```
class BrukNedTeller {
    public static void main(String[ ] arg) {
          int startTall = 49;
          NedTeller  minTeller;
          minTeller = new NedTeller(startTall+1);
    }
}
```

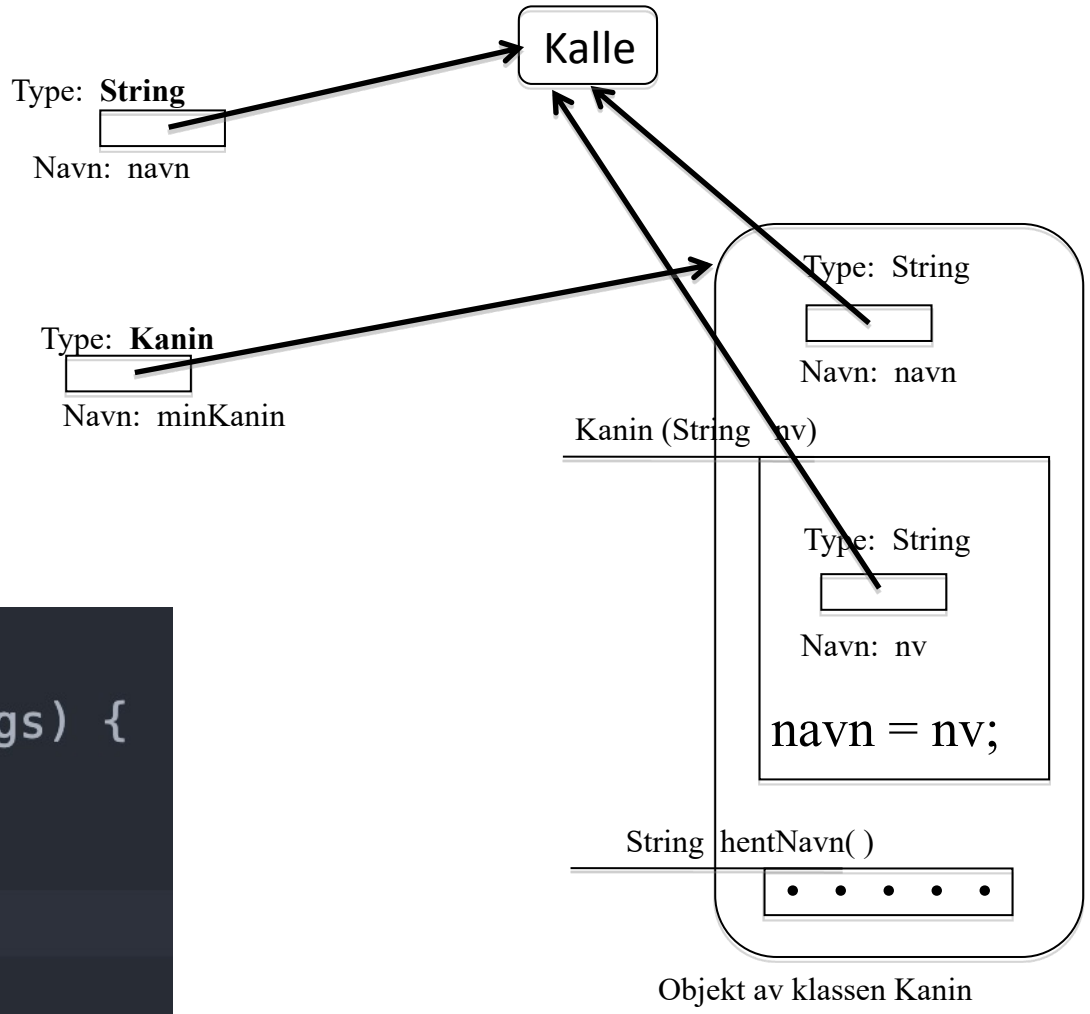


Nå har dere sett så mange klassedatastrukturer med main-metoden, så nå tegner vi det litt enklere

Parameteroverføring - referanse

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

```java
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

Oppsummering

- 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