IN1010 våren 2019
23. januar

Objektorientering i Java

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

Stein Gjessing
Institutt for informatikk

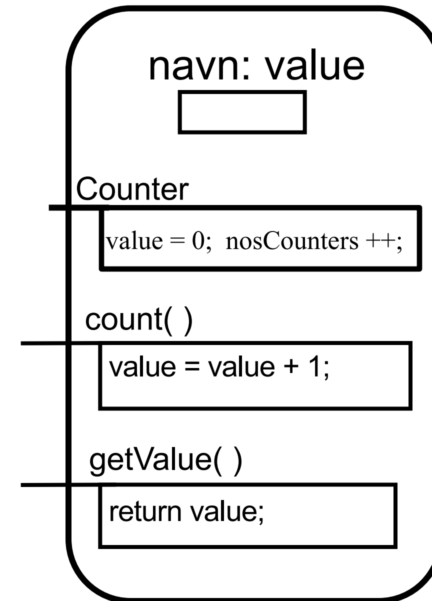
Agenda

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

Hva er et objekt ?

Et **objekt** av klassen Counter

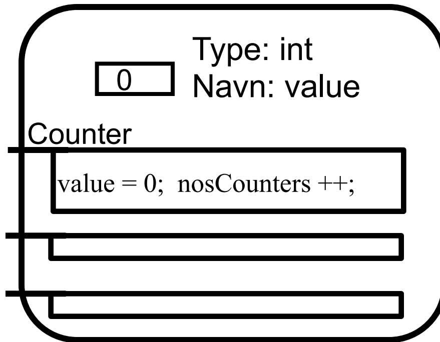
- Objekter inneholder
  - **Variable og konstanter - "DATA"**
    Kristen Nygaard: SUBSTANS
  - **Metoder – handlinger**

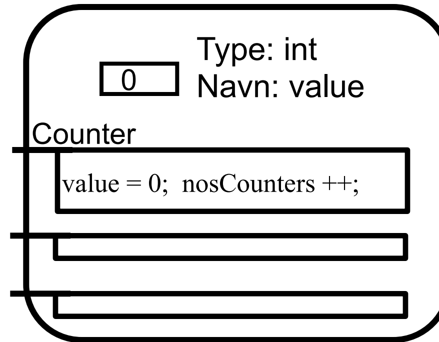


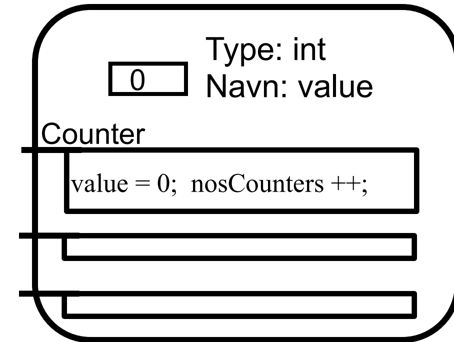

```
class Counter {
    private static int nosCounters =0;
    private int value;
    public Counter() {
        value = 0;
        nosCounters ++;
    }
    public void count( ) {
        value = value + 1;
    }
    public int getValue( ) {
        return value;
    }
}
```

Hva er et objekt ?

Etter f.eks. 3 kall på new Counter() har vi 3 objekter:

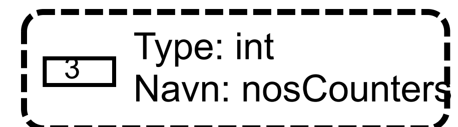







```
class Counter {
    private static int nosCounters =0;
    private int value;
    public Counter() {
        value = 0;
        nosCounters ++;
    }
    public void count( ) {
        value = value + 1;
    }
    public int getValue( ) {
        return value;
    }
}
```

Og en klasse-datastruktur:



Javas byggestener

- 8 **primitive typer**: boolean, byte, char, short, int, long, float, double.
  - Disse verdien tar typisk opp mellom en bit og 8 byte (oktetter) i primærlageret
  - (1 byte = 1 oktett = 8 bit)
- **Referanser** (objekt-referanser): adressen til et objekt og har en type som er et klassenavn (noe lignende for arrayer) (typisk 4 byte)
- En variabel (og konstanter) inneholder en verdi av en primitive type eller en referanse
  - Er Javas byggestener for data-manipulering
- Uttrykk (expressions) evalueres til en verdi av en primitiv type eller en referanse.

Referanser

Navn: minTeller

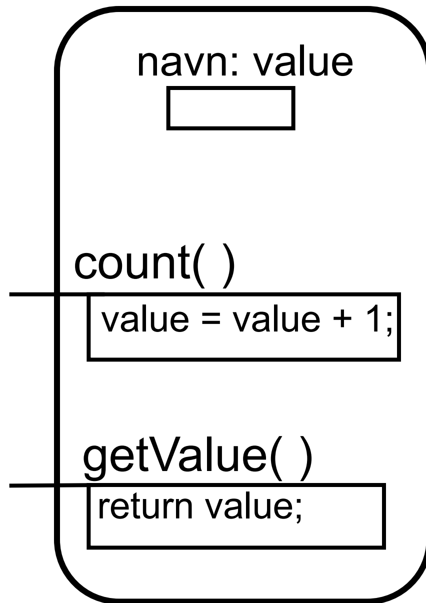
Type: Counter

Navn: hansTeller

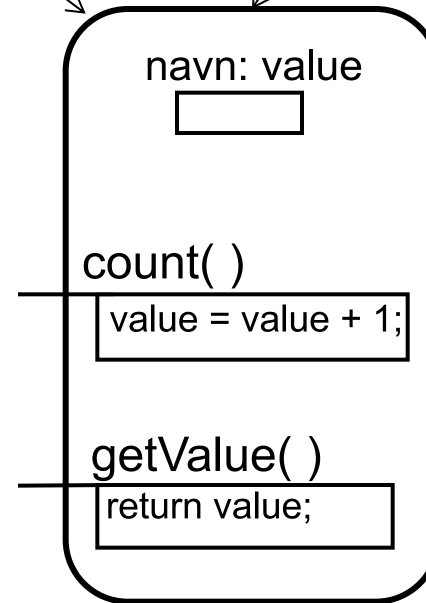
Type: Counter

Navn: dinTeller

Type: Counter



Counter-objekt



Counter-objekt

Referanser

Navn: minTeller

53485325

Type: Counter

Navn: hansTeller

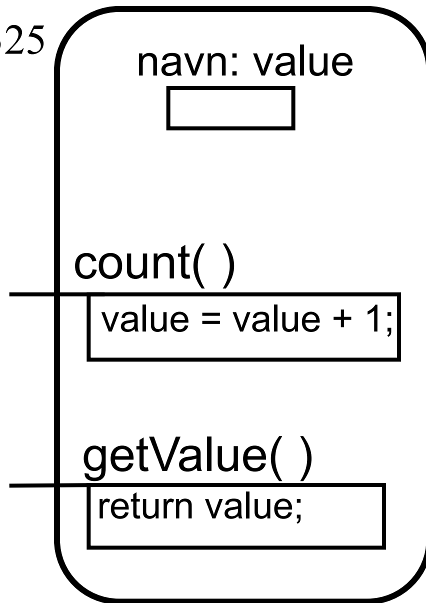
9483567

Type: Counter

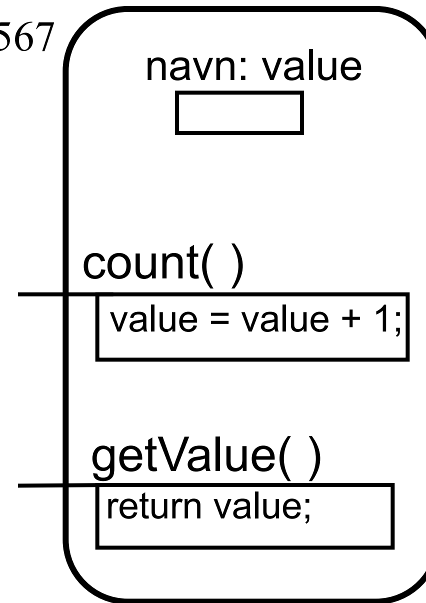
Navn: dinTeller

9483567

Type: Counter



Counter-objekt



Counter-objekt

Repetisjon fra IN1020

Minne (RAM)

0
46579598
46579599
46579600
46579601
46579602
46579603
46579604
46579605
46579606
46579607
minTeller 53485325 46579608
46579609
46579610
46579311

data

Minnebuss

53485321
53485322
53485323
53485324
53485325
53485326
53485327
53485328
53485329
53485330

program

4362738
93685398

CPU

Registre

Aritmetisk-Logisk Enhet (ALU)

4362738 Instruksjonsregister

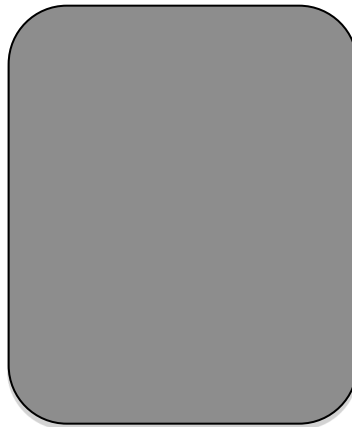
93685398 Programteller

Hva er objektorientert programmering ?

F.eks: En sort boks som tar vare på ett tall:

settInn(int tall)

taUt( )



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

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

. . .

Hvilke metoder trengs ?
Hvordan skal disse metodene virke?

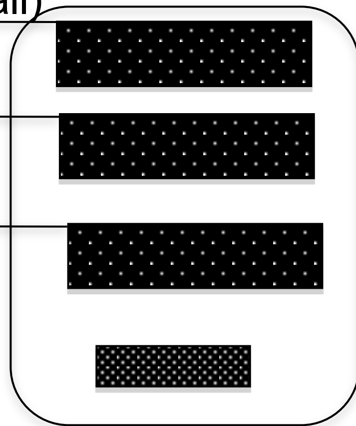
Hva er objektorientert programmering ? Hva er et objekts grensesnitt mot omverdenen? Svar: De “public” metodene.

Sort boks som tar vare på ett tall:

public void settInn(int tall)

public int taUt( )

public boolean erTom( )



Ukjent implementasjon av metode

Ukjent implementasjon av metode

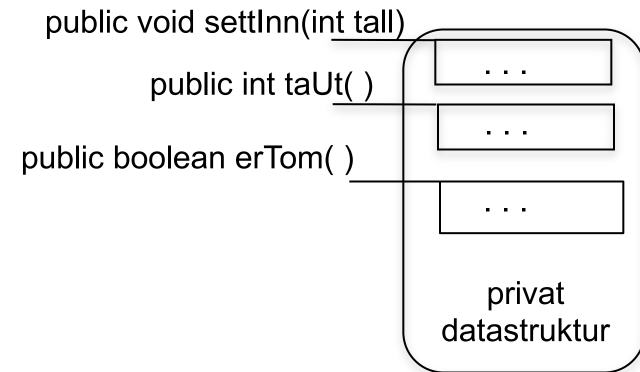
Ukjent implementasjon av metode

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

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:



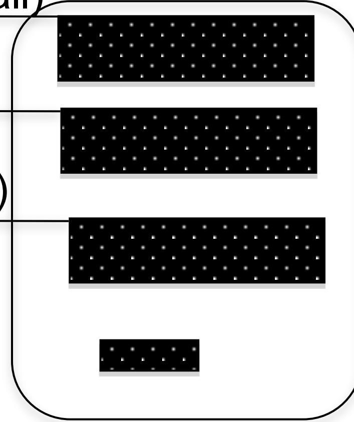
Objekt av klassen EnkelHeltallsbeholder

Metodenes **signaturer**

public void settInn(int tall)

public int taUt( )

public boolean erTom( )



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

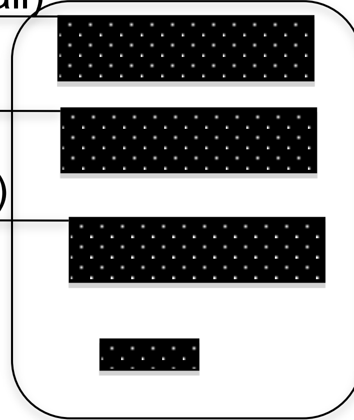
Signaturen til en metode er
- navnet på metoden
- Typene, rekkefølgen og navnene til parametrene
- retur-typen (ikke i Java)

Metodenes **semantikk**

public void settInn(int tall)

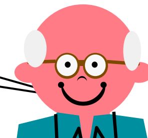
public int taUt( )

public boolean erTom( )

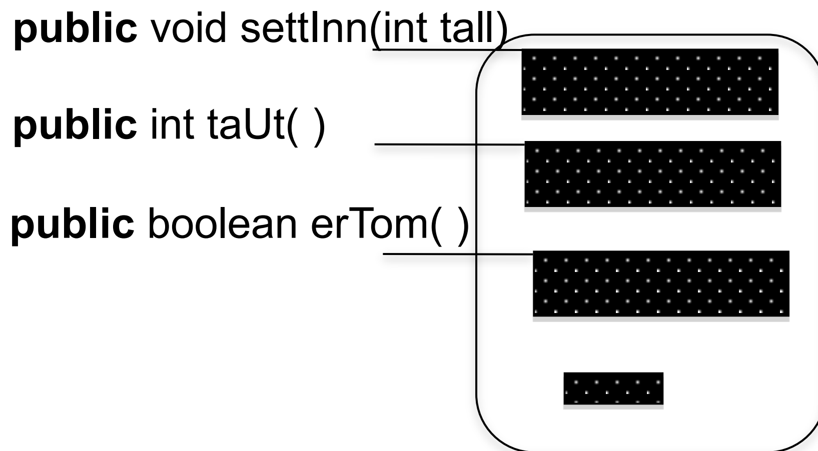


Hva gjør disse metodene? Hvordan virker de? Hva er sematikken til metodene?

Semantikk betyr virkemåte



Metodenes **semantikk**



- 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: ☺

- Først betemmer vi semantikken og signaturene
- Deretter implementerer vi metodene
  samtid som vi bestemmer oss for hva de private dataene skal være

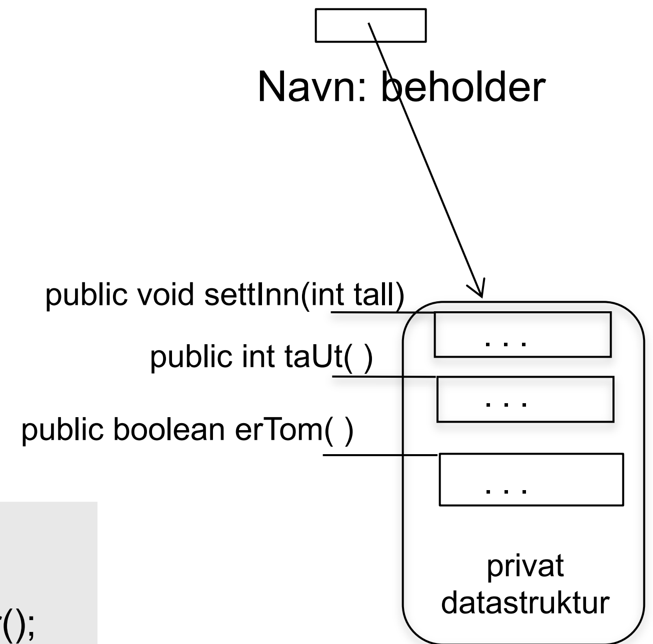
Dette gjelder for alle programmeringsspråk - dette er ikke Java-spesifikt.

☺ 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
- Men vi kommer alltid til å tenke og skrive feil
- Derfor må vi **teste** programmet vårt
- Objektorientering / modularisering:
  - Test et objekt eller en modul om gangen
    - Sørg for at den er så riktig som mulig
  - Deretter kan vi test sammensettingen av objektene / modulene

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

                . . .
        }
}
```




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

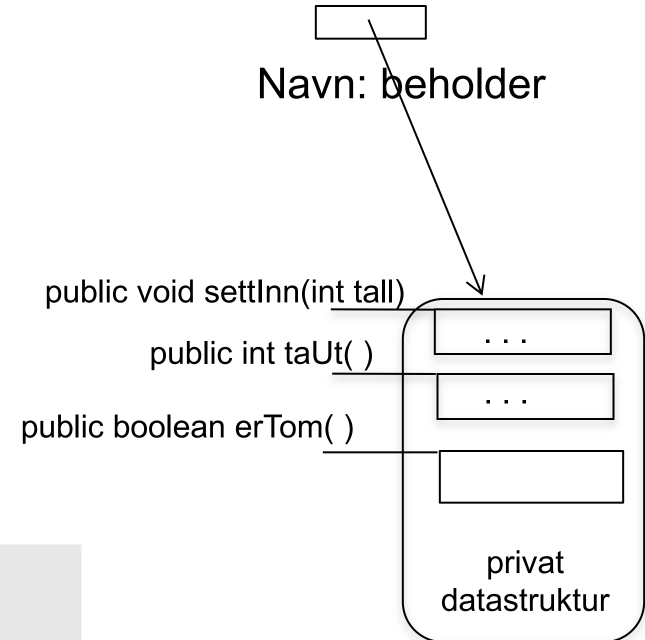
Oppgave: Når skal vi skrive testprogrammet?

Om du er alene?
Om dere er flere?

Veldig enkel test av beholder


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

FULLSTENDIG KJØRBART PROGRAM




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

```
>java VeldigEnkelTestAvBeholder
Riktig
>
```

UNIVERSITETET I OSLO

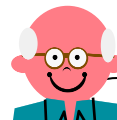
Enkel test av heltallsbeholder

Institutt for informatikk

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

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



Test en ting om gangen
Lag gode tekster for testene

Med Javadoc (og uten ? feil)

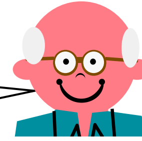

```
/** Objekter av denne klassen tar vare på
 *  ett  heltall.
 *  Initielt er beholderen tom
 *
 * @author Stein Gjessing
 * versjon 5. januar 2017
 */
public class Enkelheltallsbeholder {
     private boolean tom = true;
     private int tallet;
    /**
     * Gjør at objektet tar vare på tallet som
     * 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;
      }
}
```

Javadoc resultat


```
mos:programmer steing$ javadoc Enkelheltallsbeholder.java
Loading source file Enkelheltallsbeholder.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_31
Building tree for all the packages and classes...
Generating ./Enkelheltallsbeholder.html...
. . . . . . .
Generating ./deprecated-list.html...
Building index for all classes...
Generating ./allclasses-frame.html...
Generating ./allclasses-noframe.html...
Generating ./index.html...
Generating ./help-doc.html...
mos:programmer steing$
```

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



```
/** Objekter av denne klassen tar vare på
 *  ett  heltall.
 *  Initielt er beholderen tom
 *
 * @author Stein Gjessing
 * versjon 5. januar 2017
 */
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;
      }
```

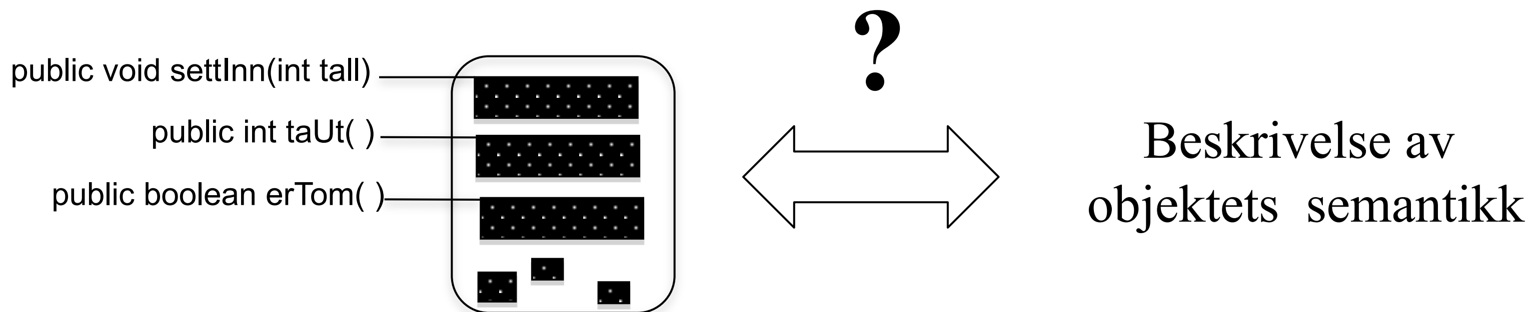
```
     /**
      * Sjekkeren om objektet er tomt
      *
      *@return objektet er tomt
      */
      public boolean erTom ( ) {
            return tom;
      }
     /**
      * Tar ut av objektet det tallet objektet
      *   tar vare på.
      * Om objektet alt er tomt, returneres en
      *  ubestemt 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;
      }
}
```

(Samme Java-kode som på lysark 13)

Institutt for informatikk har 12 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: Kaniner og 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( ) {
                . . . .
                . . . .
        }
}
```

Mer om metoders signatur og metoders semantikk

Signaturer:

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

Semantikk:

- 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

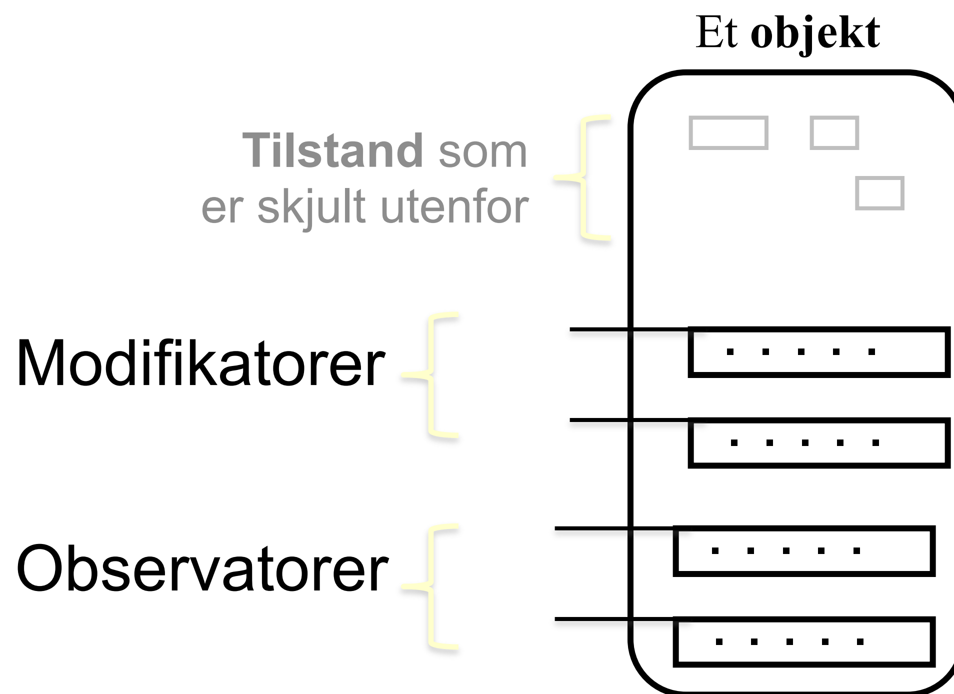

```
class Kanin{
    private String navn;
    public Kanin(String nv) {navn = nv;}
    public String hentNavn() {return navn;}
}
```

```
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;
        }
}
```

Som bakgrunn for enhetstesting:

Modifikatorer og Observatorer

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



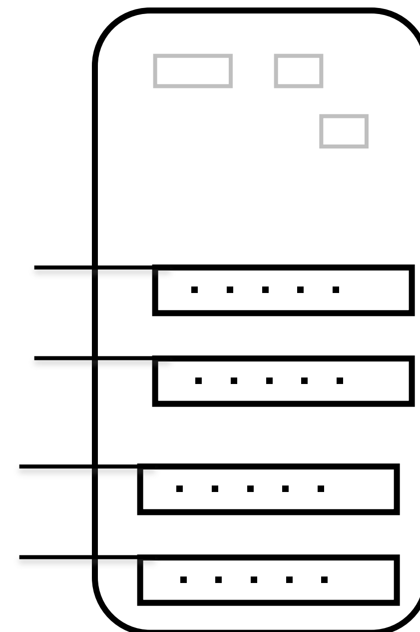
Modifikator, f.eks. settInn(), Observator, f.eks. les()

Testing - - Enhetstesting

- **Når vi skal teste et objekt kan vi først kalle en modifikator-metode og deretter en observator-metode og se om vi observerer det ønskede resultat**

Modifikatorer

Observatorer



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

Enhetstesting av class Kaninbur


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

```
class Kaninbur  {
            . . . .
}
```

```
// Testprogram:
public static void main ( . . . ) {
    Kaninbur mittKaninbur = new Kaninbur( );
```

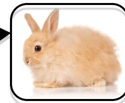
Type: **Kaninbur**

Navn: mittKaninbur

```
    Kanin kalle = new Kanin("Kalle");
```

Type: Kanin

Navn: kalle



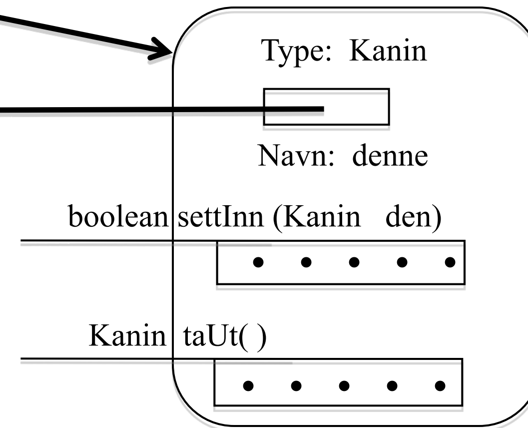
Objekt av klassen Kanin

```
    mittKaninbur.settInn(kalle);

    Kanin sprett = new Kanin("Sprett");
    boolean settInnOK = mittKaninbur.settInn(sprett);
    test("Test inn i fullt bur", settInnOK);
    . . . .
}
```

Type: Kanin

Navn: sprett

Objekt av klassen Kanin



Objekt av klassen Kaninbur

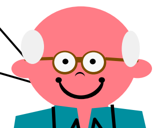
Eksempel på kaninbur til mange kaniner

3 observatorer

2 modifikatorer

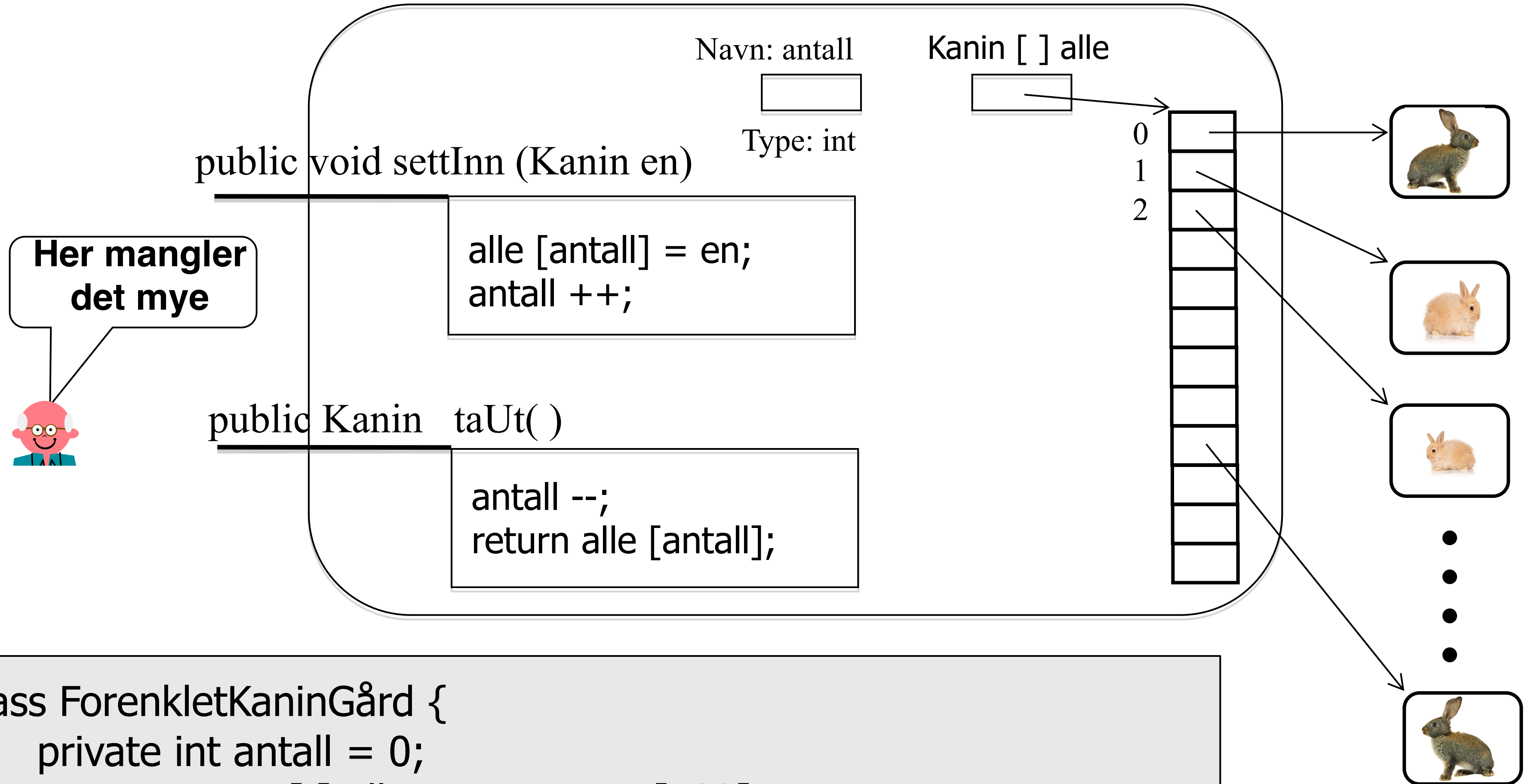
```
class   KaninGård {
        public boolean full( ) { . . . }
        public boolean tom ( ) { . . . }
        public Kanin  finnEn(String navn) { . . . }
        public void settInn (Kanin kn) { . . . }
        public void fjern(String navn) { . . . }
}
```

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 på neste side

Objekt av klassen ForenkletKaninGård




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

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

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

Oppgave: skriv ferdig klassen KaninGård fra forrige side

Array er iboende i datamaskinen og i Java

To beholdere (containere) fra Javas bibliotek**:**
ArrayList og **HashMap**

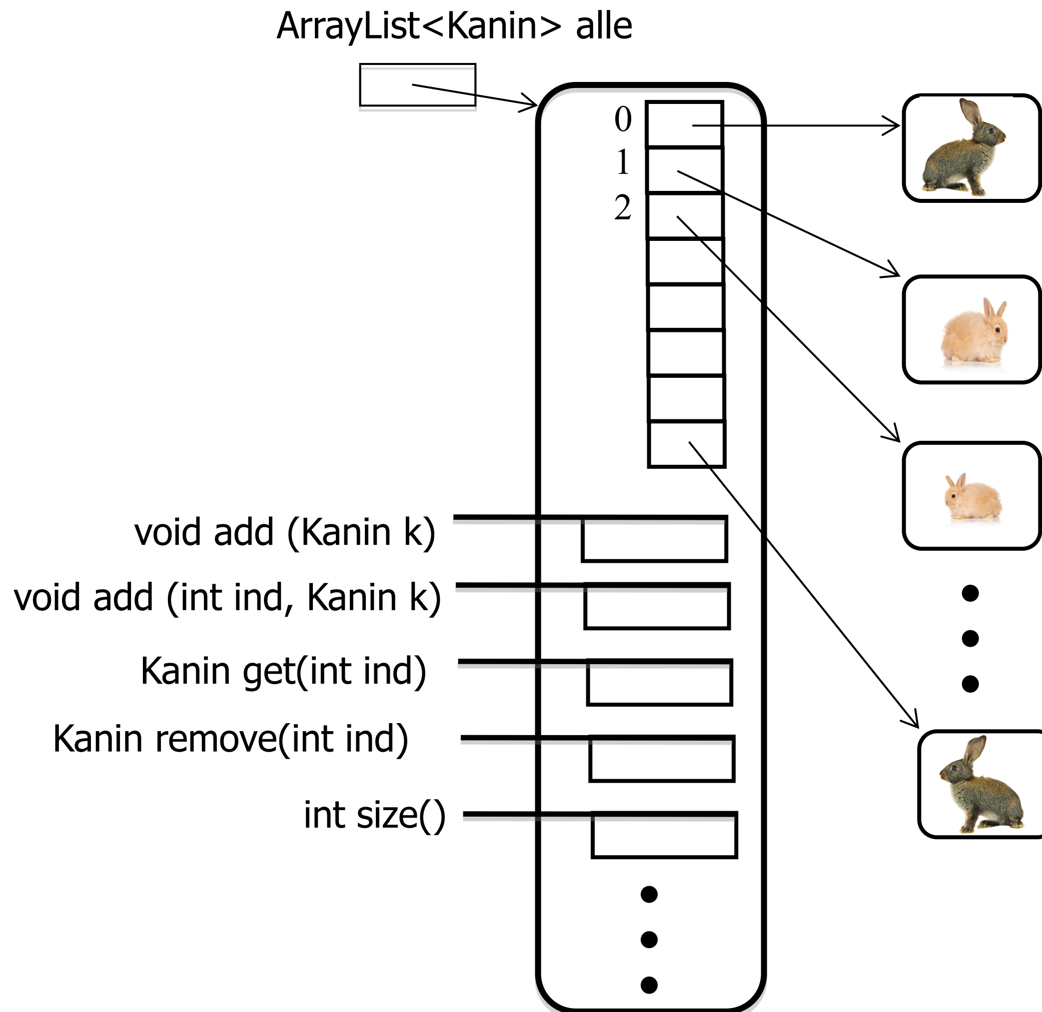
- ArrayList er en fleksibel array som utvider seg og trekker seg sammen etter behov
- ArrayList <Kaniner> mineKaniner = new ArrayList <Kaniner> ();
- Metoder: add, get, remove, . . . se Java-biblioteket

- HashMap er en beholder 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<Kanin>( );



Objekt av klassen ForenkletKaninGårdAL

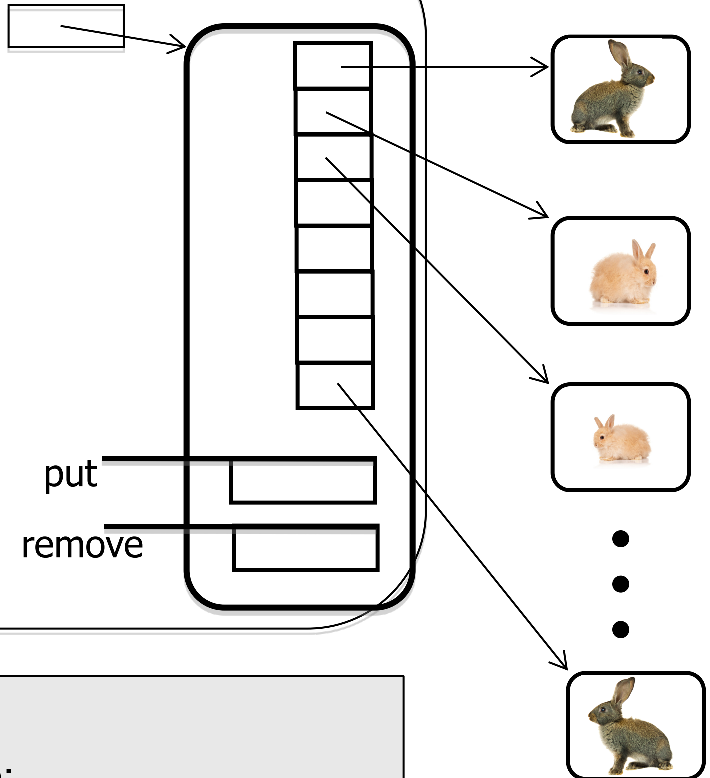
public void settInn (Kanin en)

```
alle.add(en);
```

public Kanin   taUt( )

```
if (! alle.isEmpty() )
    return(alle.remove(0));
else
    return null;
```

ArrayList<Kanin> alle



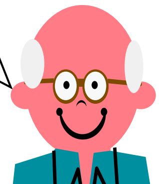

```
class ForenkletKaninGårdAL {
    private ArrayList<Kanin> alle = new ArrayList<Kanin>( );
    public void settInn(Kanin en) { alle.add(en); }
    public Kanin taUt( )  {
        if (! alle.isEmpty() ) {
            return(alle.remove(0));
        }   else   { return null; }
    }
}
```

Oppgave:
skriv ferdig klassen KaninGård fra forrige side

```
import java.util.ArrayList;

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

class ForenkletKaninGardAL {
    private int antall = 0;
    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;
          }
    }
}
```

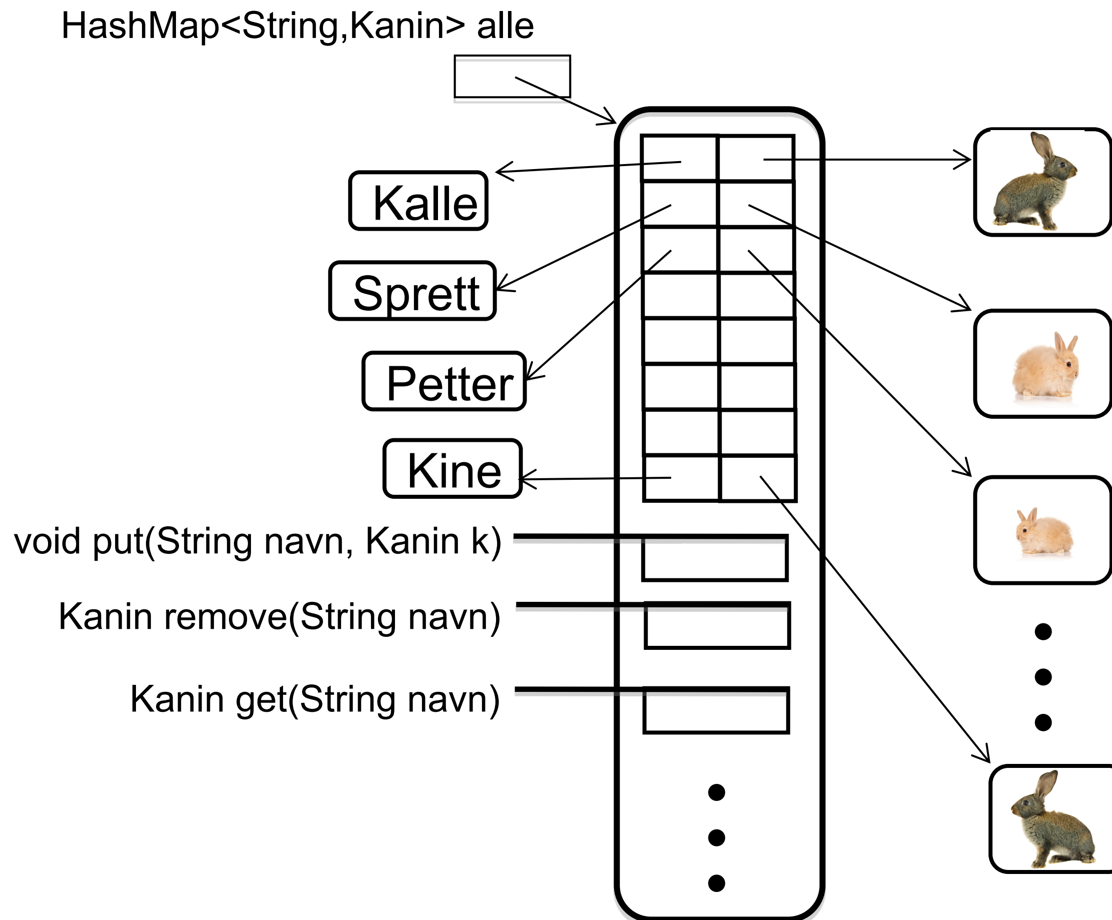
```
class  KaningardTestArrayList {
    public static void main (String [ ] args) {
        ForenkletKaninGardAL mittKaninbur = new ForenkletKaninGardAL( );
        Kanin kalle = new Kanin("Kalle");
        mittKaninbur.settInn(kalle);
        Kanin sprett = new Kanin("Sprett");
        mittKaninbur.settInn(sprett);
        Kanin enKanin = mittKaninbur.taUt();
        test (((enKanin != null) &&  enKanin.hentNavn().equals("Kalle")), 1);
        enKanin = mittKaninbur.taUt( );
        test (((enKanin != null) &  enKanin.hentNavn().equals("Sprett")),2);
        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);
        }
    }
}
```

Mer om testing 1. april

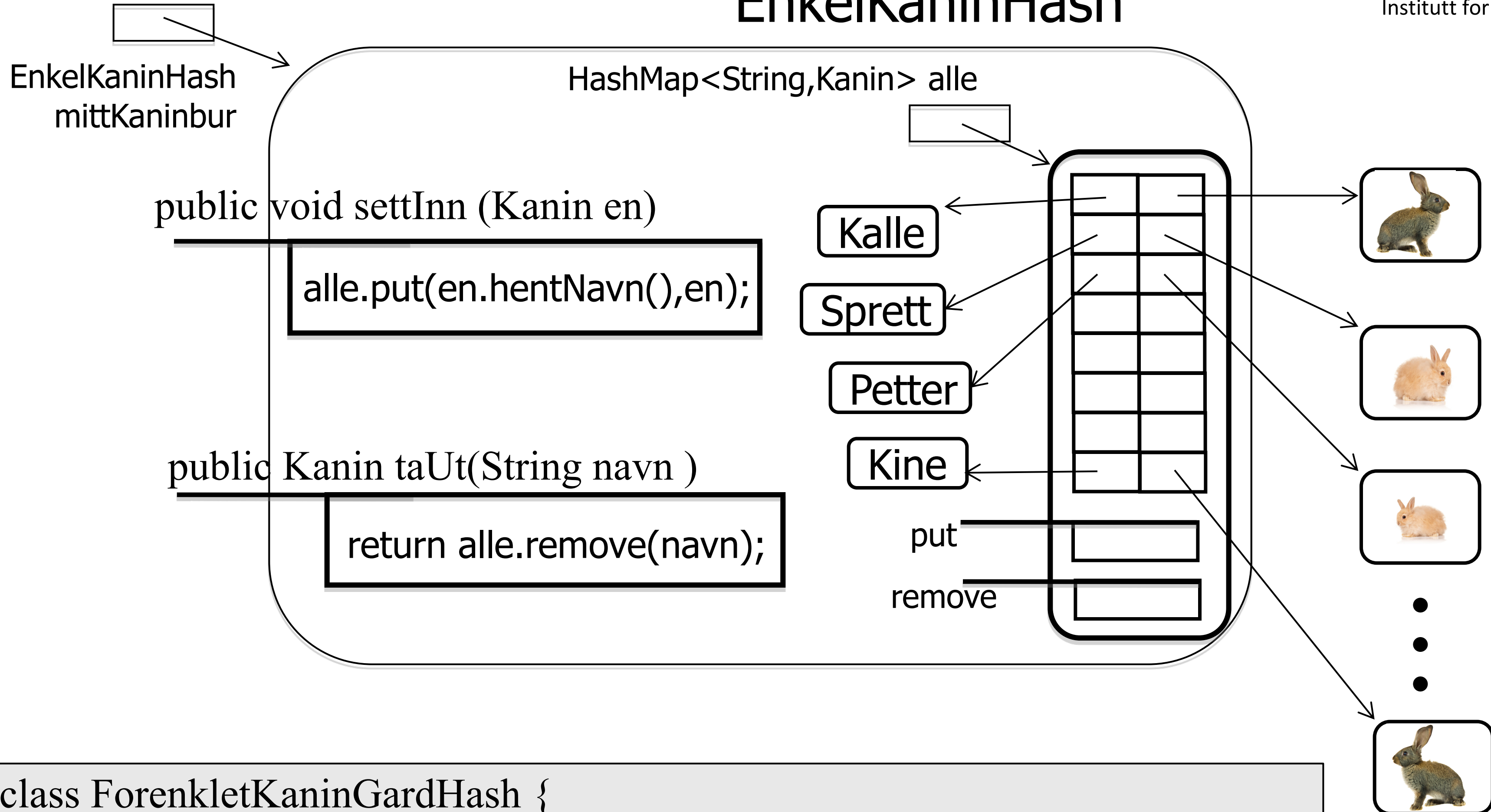


HashMap

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



Objekt av klassen EnkelKaninHash




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

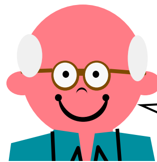
```
import java.util.HashMap;

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

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

```
class  KaningardTestHash {
    public static void main (String [ ] args) {
        EnkelKaninHash mittKaninbur = new EnkelKaninHash( );
        Kanin kalle = new Kanin("Kalle");
        mittKaninbur.settInn(kalle);
        Kanin sprett = new Kanin("Sprett");
        mittKaninbur.settInn(sprett);
        Kanin enKanin = mittKaninbur.taUt("Kalle");
        test (((enKanin != null) &&  enKanin.hentNavn().equals("Kalle")), 1);
        enKanin = mittKaninbur.taUt("Sprett");
        test (((enKanin != null) &&  enKanin.hentNavn().equals("Sprett")),2);
        enKanin = mittKaninbur.taUt("Petter");
        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);
         }
     }
}
```

Unntak / Exceptions

- En metode som kan komme til å gjøre en IO-feil på fil må enten behandle denne selv, eller *kaste feilen videre* (også i main):

```
public void  mittProgMedIO( ) throws IOException {
    < kode som gjør fil-behandling>
}
```



Vi skal lære mer om unntak og feilbehandling etter hvert, **bare litt her i dag**

(IO: Innlesing/Utskrift)

throws er et Java nøkkelord

UNIVERSITETET
I OSLO

Institutt for informatikk

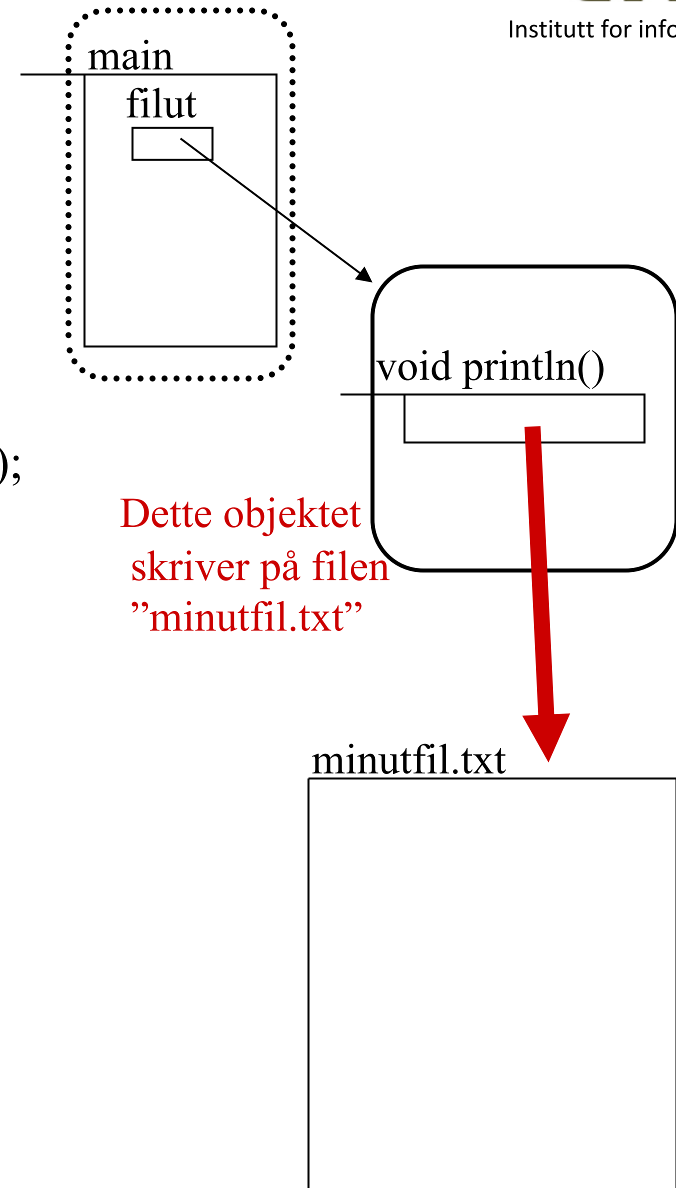
Array indeks utenfor sine grenser

```
int [ ] tallVektor;
tallVektor = new int [100];
tallVektor[101] = 17;
```

```
Seacobra:programmer steing$ javac Test.java
Seacobra:programmer steing$ java Test
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 101
at Test(Test.java:8)
Seacobra:programmer steing$
```

Du kan behandle unntaket selv:

```
import java.io.*;

try {
  PrintWriter  filut = new PrintWriter (”minutfil.txt”);

  // Utskrift skjer som til skjerm:

  filut.println( ”utskrift” + 17 ):

  // For at innholdet på den nye filen skal
  // bevares må vi til slutt si:

  filut.close( );

}
catch (FileNotFoundException f) { . . .  }
```



Generelt om unntak / feil - behandling i Java

- Mye kode kan feile og feilaktige situasjoner (unntak) kan oppstå.
- Kode som kan feile ***kan -*** og som oftest ***må*** - vi legge følgende rundt:

```
try {
            <... Kode som kan feile …>
 }
 catch  (Exception e)  {
            < .... Gjør noe med feilen ,
                     prøv å rett opp ...>
 }
```

Feiler koden blir denne blokken utført med feilobjektet som e peker på som parameter

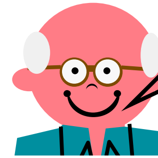
Fem reserverte Java ord

- **try** - Står foran en blokk som er usikker dvs. der det kan oppstå et unntak
- **catch** - Står foran en blokk som behandler et unntak.
  Har en referanse til et unntaksobjekt som parameter
- **finally** - blir alltid utført (mer senere)
- **throw** - Starter å kaste et unntak (mer senere)
  throw <en peker til et unntaksobjekt>
  f.eks throw new Unntak();
- **throws** - Kaster et unntak videre
  Brukes i overskriften på en metode som ikke selv vil behandle et unntak
- **Viktigst bruk:**

```
try {
        <kode som kan feile>
}
catch (Unntaksklasse u) {
                <behandle  unntaket,  u peker på et objekt som beskriver  unntaket>
 }
```

Fange divisjon med '0'

```
public class TryTest
{
    public static void  main ( String [ ] args) {
        int i=1;
        for (int j=0; j < 5; j++)
            try{
                i = 10/j;
                System.out.println("Det gikk OK, i:" + i + ", j:" + j);

            } catch (Exception e) {
                System.out.println("Feil i uttrykk: "+ e.getMessage( ));
            }

    }
}
```



Her tar programmet seg av "hele feilen"

```
snidil> java TryTest
Feil i uttrykk: / by zero
Det gikk OK, i:10, j:1
Det gikk OK, i:5, j:2
Det gikk OK, i:3, j:3
Det gikk OK, i:2, j:4
snidil>
```

Oppsummering

- Enhetsestesting er viktig i programmering
  - Gir både gode råd ved valg av programutforming og gode råd ved testing
- Nå kan dere «alt» om klasser og objekter (og typer)
  - Etter denne uken må dere kunne Java for IN1000
- Dere kjenner til og kan bruke arrayer (innebygget i Java (og i datamaskiner)) og HashMap og ArrayList fra Javabiblioteket.

- Java har gode mekansimer for unntakshåndtering
  - som vi skal se mer på etter hvert