

IN1010 våren 2019

23. januar

Objektorientering i Java

Om enhetstesting

Om arrayer og noen klasser som kan ta vare på objekter

Stein Gjessing
Institutt for informatikk



Agenda

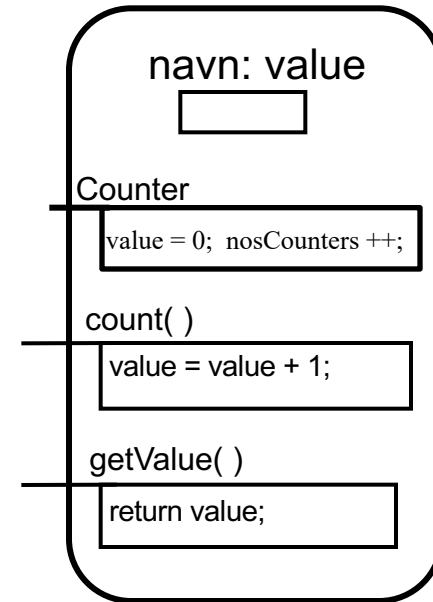
- Kort repetisjon om objekter
- Hvordan representeres objekter i minne
- Mer om objekter
 - Enhetstesting
- Mer om arrayer
- Om beholdere innebygget i Java:
 - HashMap
 - ArrayList

Hva er et objekt ?

- Objekter inneholder
 - **Variable og konstanter - “DATA”**
 - **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;  
    }  
}
```

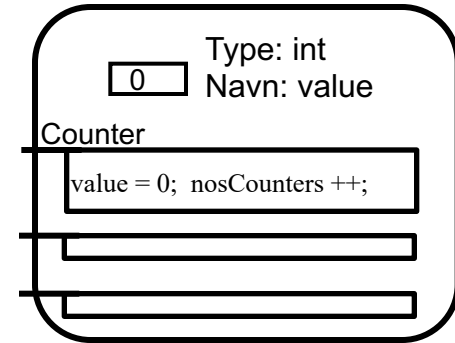
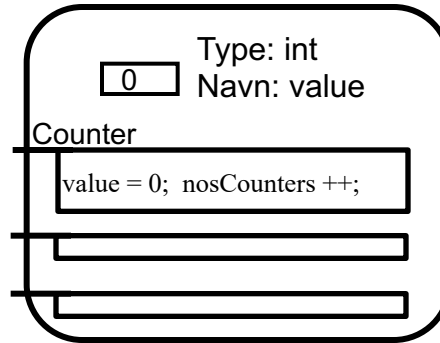
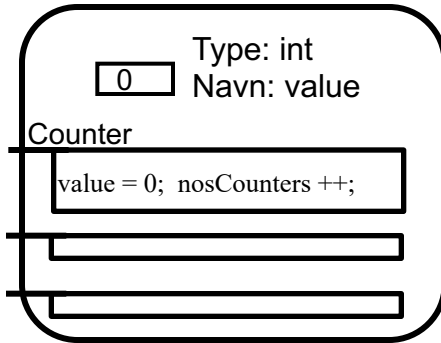
Et **objekt** av
klassen Counter



Repetisjon

Hva er et objekt ?

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



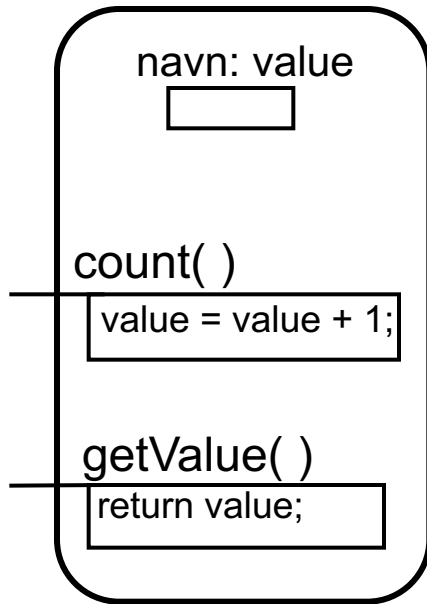
```
class Counter {  
    private static int nosCounters = 0;  
    private int value;  
    public Counter() {  
        value = 0;  
        nosCounters ++;  
    }  
    public void count() {  
        value = value + 1;  
    }  
    public int getValue() {  
        return value;  
    }  
}
```

Repetisjon

Referanser

Navn: minTeller

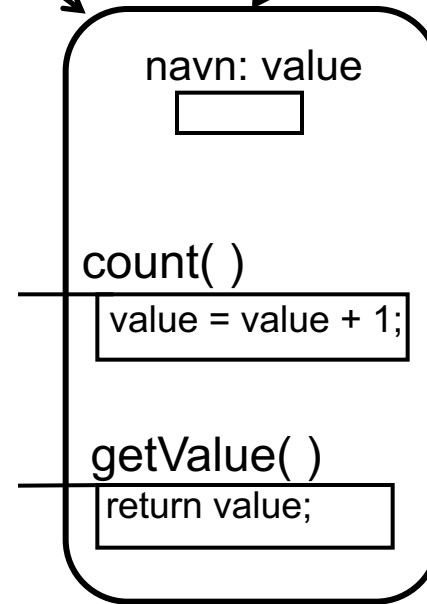
Type: Counter



Counter-objekt

Navn: hansTeller

Type: Counter



Counter-objekt

Navn: dinTeller

Type: Counter

Referanser

Navn: minTeller

53485325

Type: Counter

Navn: hansTeller

9483567

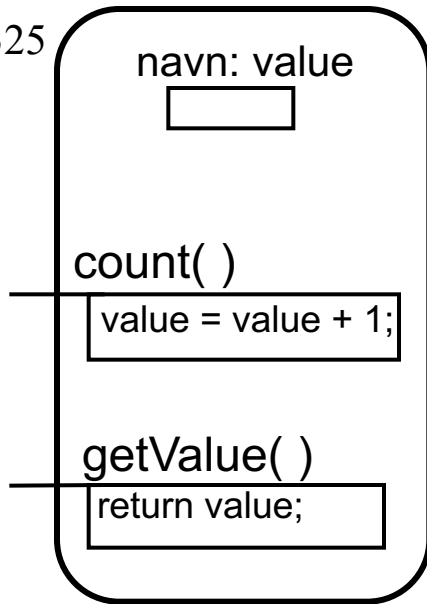
Type: Counter

Navn: dinTeller

9483567

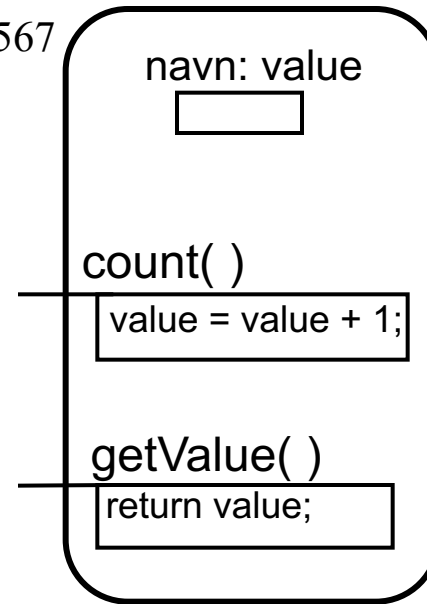
Type: Counter

53485325



Counter-objekt

9483567



Counter-objekt

**Minne
(RAM)**

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

minTeller

53485325

data

Minnebuss

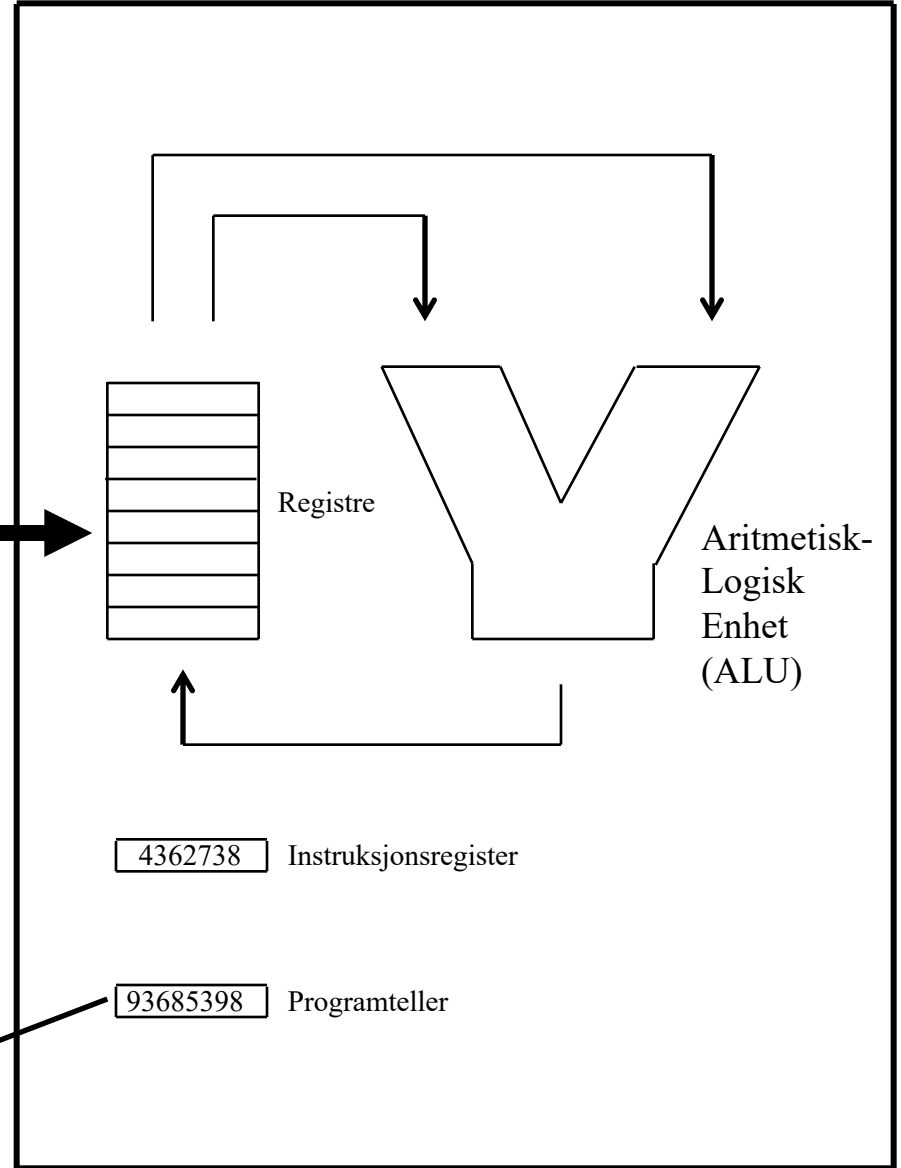


53485321
53485322
53485323
53485324
53485325
53485326
53485327
53485328
53485329
53485330

program

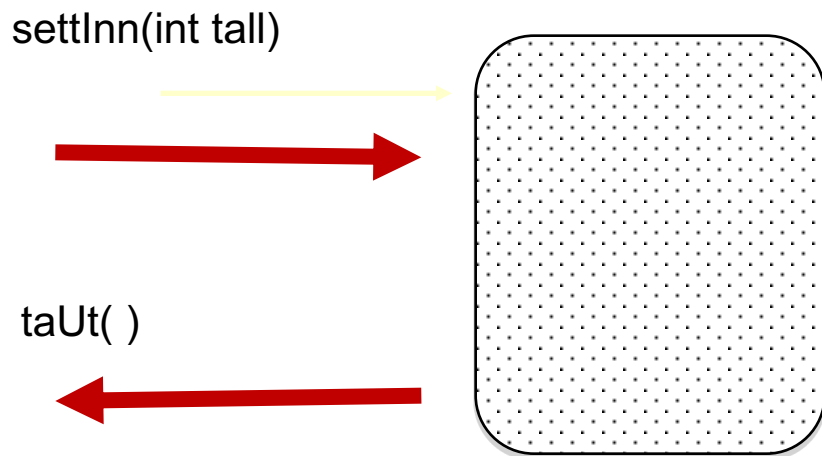
4362738

93685398



Hva er objektorientert programmering ?

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



Masse “problemer”, selv med et så lite eksempel:

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

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

...

Så: Hvilke metoder trengs ?
Hvordan skal disse metodene virke?

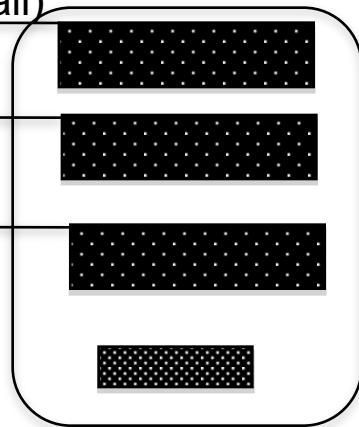
Hva er objektorientert programmering ?
Hva er et objekts grensesnitt mot omverdenen?
Svar: De “public” metodene.

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

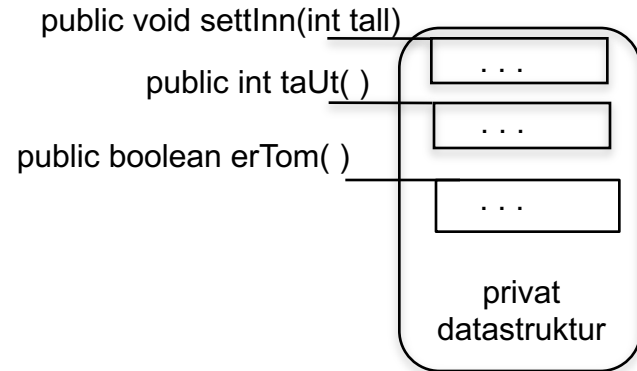
Ukjente **private** data og ukjente
private metoder

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:

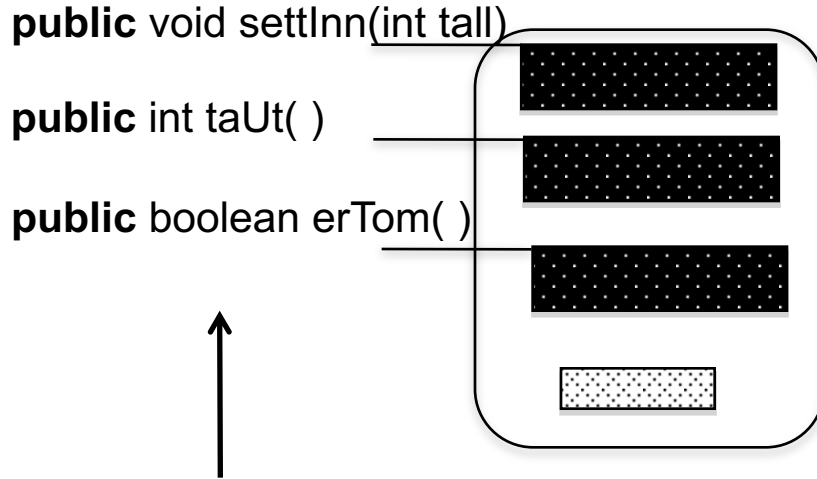


Objekt av klassen
EnkelHeltallsbeholder

Dette er en historie fra virkeligheten !



Metodenes signaturer



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

Signaturen til en metode er

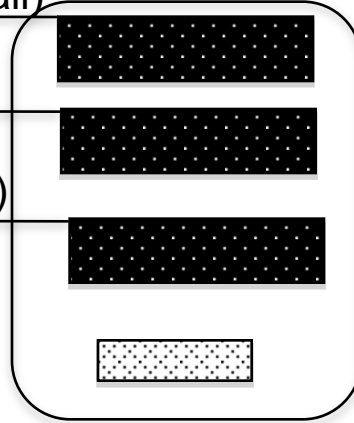
- navnet på metoden
- Typene, rekkefølgen og navnene til parametrene
- retur-typen (ikke i Java)

Metodenes semantikk

public void settInn(int tall)

public int taUt()

public boolean erTom()

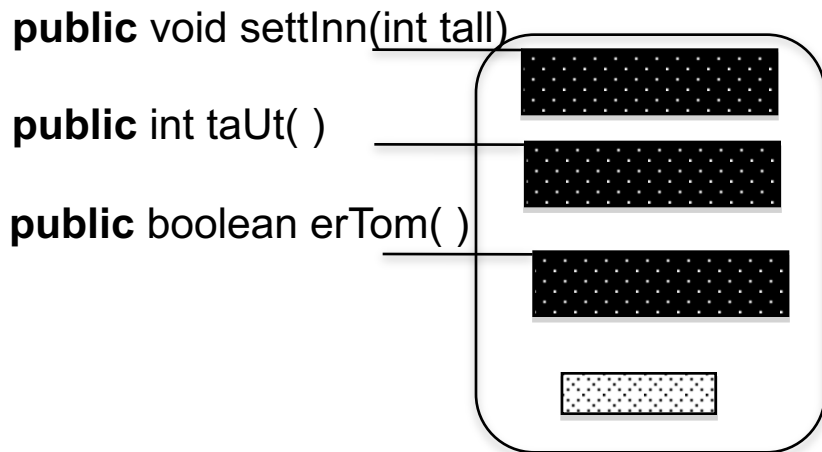


Hva gjør disse metodene? Hvordan virker de? Hva er semantikken til metodene?

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 tallet der 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 bestemmer 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



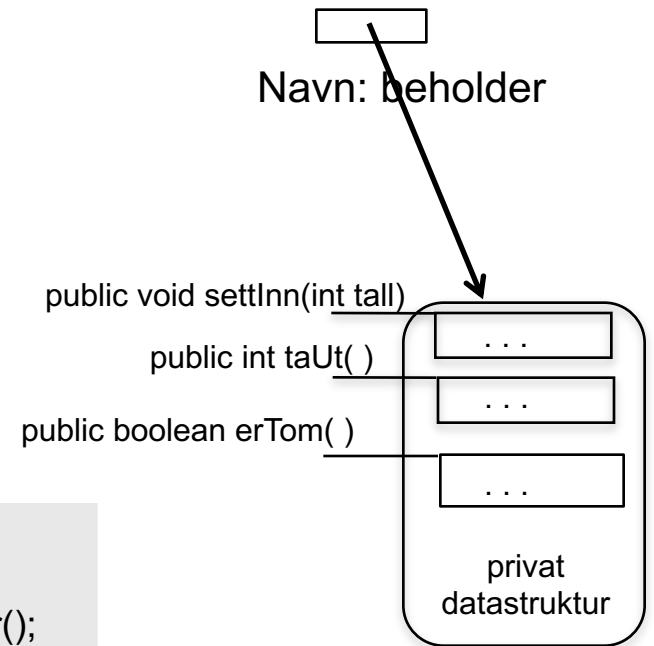
Testing -- Enhetstesting

- 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();  
        ...  
    }  
}
```

Type: EnkelHeltallsbeholder

Objekt av klassen
EnkelHeltallsbeholder

Oppgave: Når skal vi skrive testprogrammet?

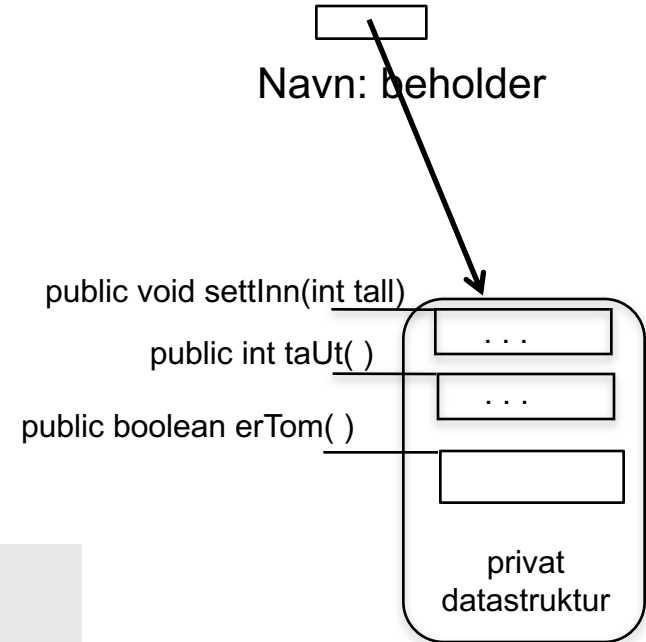
Om du er alene?
Om dere er flere?


```
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;  
    }  
}
```

FULLSTENDIG KJØRBART PROGRAM



Type: EnkelHeltallsbeholder



Objekt av klassen
EnkelHeltallsbeholder

```
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  
>
```



```
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;
    }
}
```

```
>java EnkelTestAvHeltallsbeholder
```

Riktig 1

Feil 2

Riktig 3

Feil 4

>

Hvor er feilen ??



```
/** 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;
}
}
```

Enkelheltallsbeholder x In Java, what ... x Enkelheltallsbehol... x Enkelheltallsbehol... x +

file:///Users/steing/Documents/inf1010/ java not a nur

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

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

Class Enkelheltallsbeholder

java.lang.Object
Enkelheltallsbeholder

```
public class Enkelheltallsbeholder
extends java.lang.Object
```

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

Constructor Summary

Constructors

Constructor and Description

Enkelheltallsbeholder()

Method Summary

All Methods Instance Methods Concrete Methods

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

Methods inherited from class java.lang.Object

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

Constructor Detail

Enkelheltallsbeholder

```
public Enkelheltallsbeholder()
```

Method Detail

settInn

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

GjÅ_r at objektet tar vare påY 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 påY

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 påY. 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

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

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

```
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;
}
}
```

Institutt for informatikk har 12 forskningsgrupper.

En av disse heter “Pålitelige systemer” (PSY).

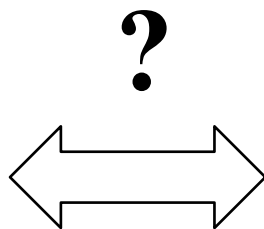
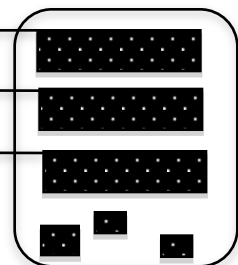
Her arbeider de bl.a. med å formalisere disse sematiske kravene, slik at du kan få hjelp av datamaskinen til å sjekke at implementasjonen overholder de semantiske kravene. Litt mer på en senere forelesning.

Implementasjon

public void settInn(int tall)

public int taUt()

public boolean erTom()



Beskrivelse av
objektets semantikk

*De sematiske kravene kalles også en “kontrakt”
(mellom brukerne av objektet og objektet selv)*

```
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() {  
        ....  
        ....  
    }  
}
```



Igjen: Signatur (syntaks) og Sematikk

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.**

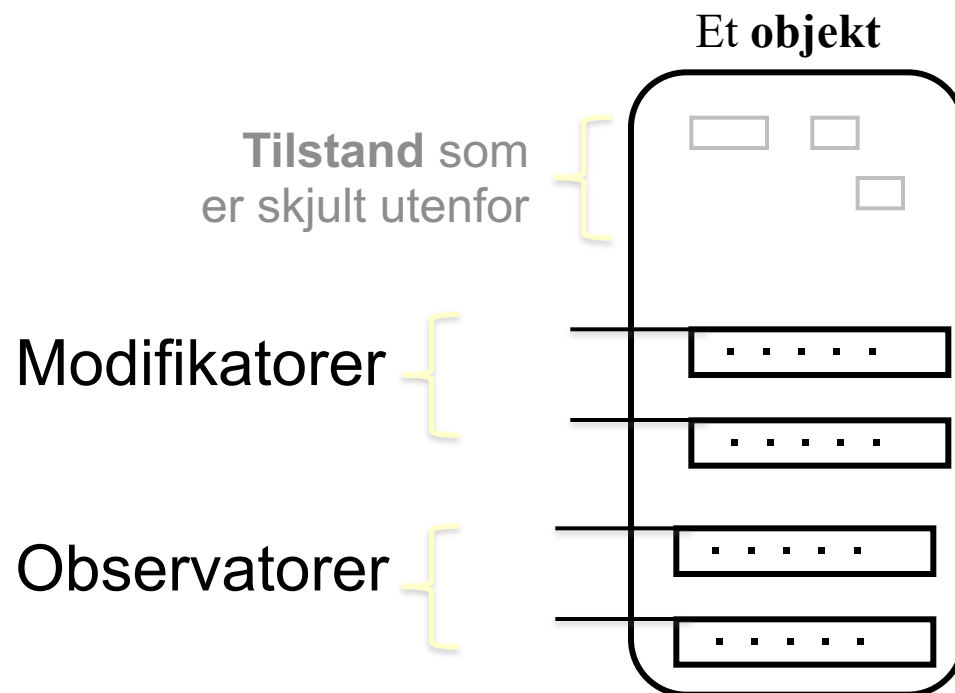

```
class Kanin {  
    private String navn;  
    public Kanin(String nv) {navn = nv;}  
    public String hentNavn() {return navn;}  
}
```



```
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;  
    }  
}
```

Modifikatorer og Observatorer

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



Modifikator, f.eks. `set()`,

Observator, f.eks. `les()`

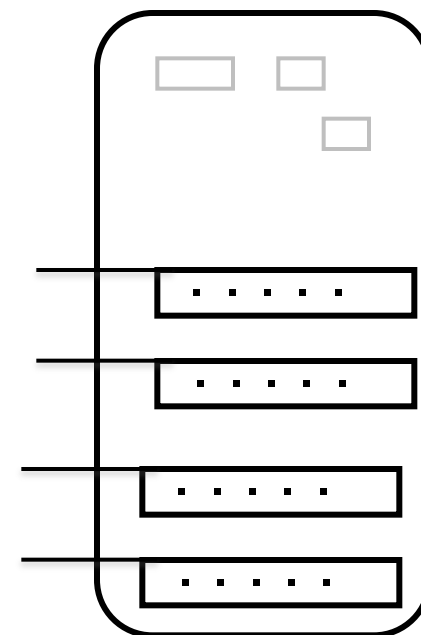
Testing - - Enhetstesting

- 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 {

Et objekt



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);
```

....

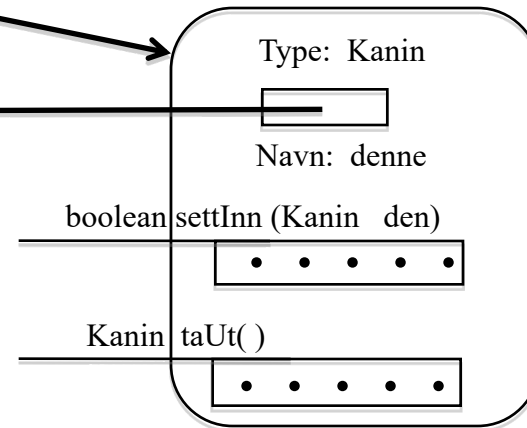
Type: Kanin



Navn: sprett



Objekt av
klassen Kanin



Objekt av klassen Kaninbur

Eksempel på kaninbur til mange kaniner

3 observatorer

2 modifikatorer

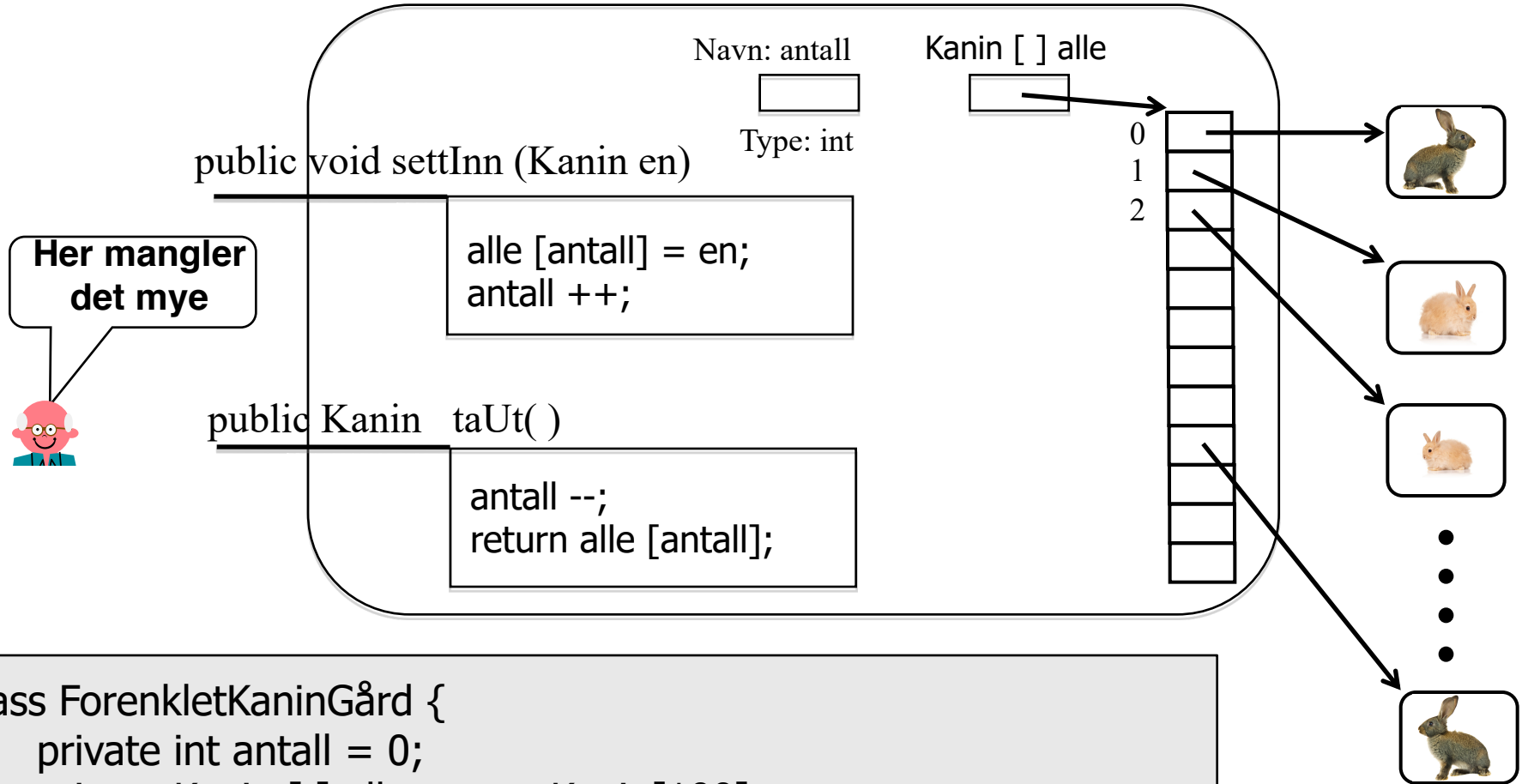
```
class KaninGård {  
    public boolean full() { ... }  
    public boolean tom () { ... }  
    public Kanin finnEn(String navn) { ... }  
    public void settInn (Kanin kn) { ... }  
    public void fjern(String navn) { ... }  
}
```

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**

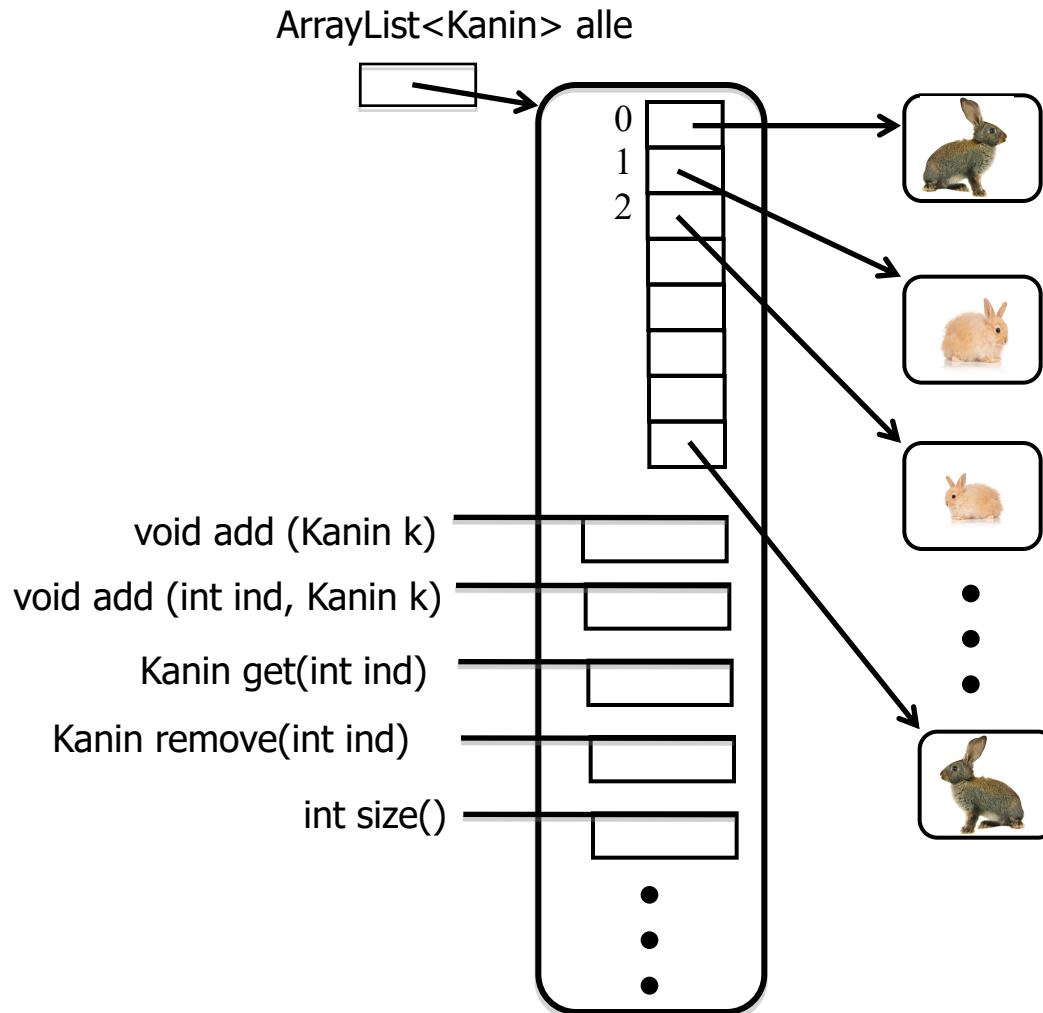
- ArrayList er en fleksibel array som utvider seg og trekker seg sammen etter behov
- `ArrayList <Kaniner> mineKaniner = new ArrayList <Kaniner> ();`
- Metoder: `add`, `get`, `remove`, . . . se Java-biblioteket

- 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-biblioteket

ArrayList

Deklarasjon:

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



Objekt av klassen ForenkletKaninGårdAL

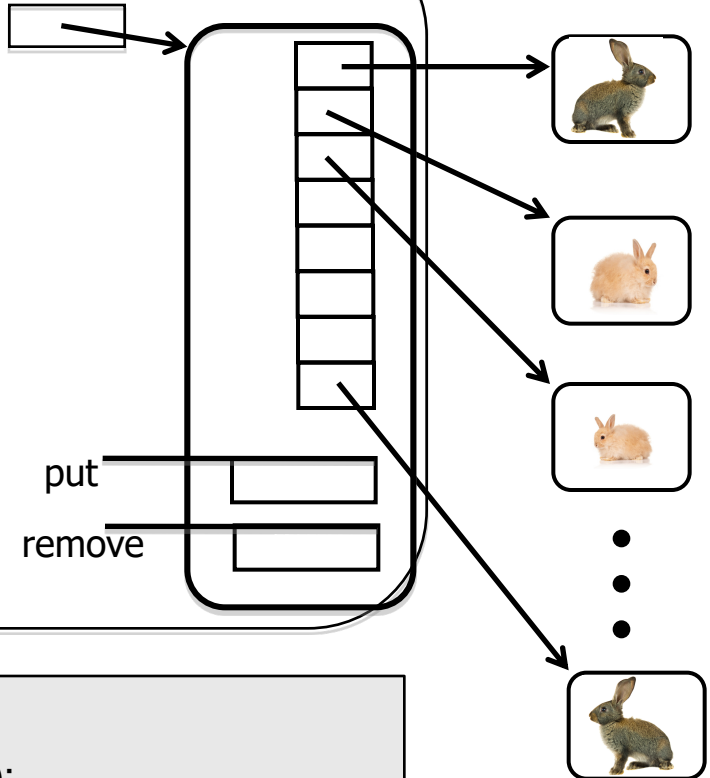
```
public void settInn (Kanin en)
```

```
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;  
        }  
    }  
}
```



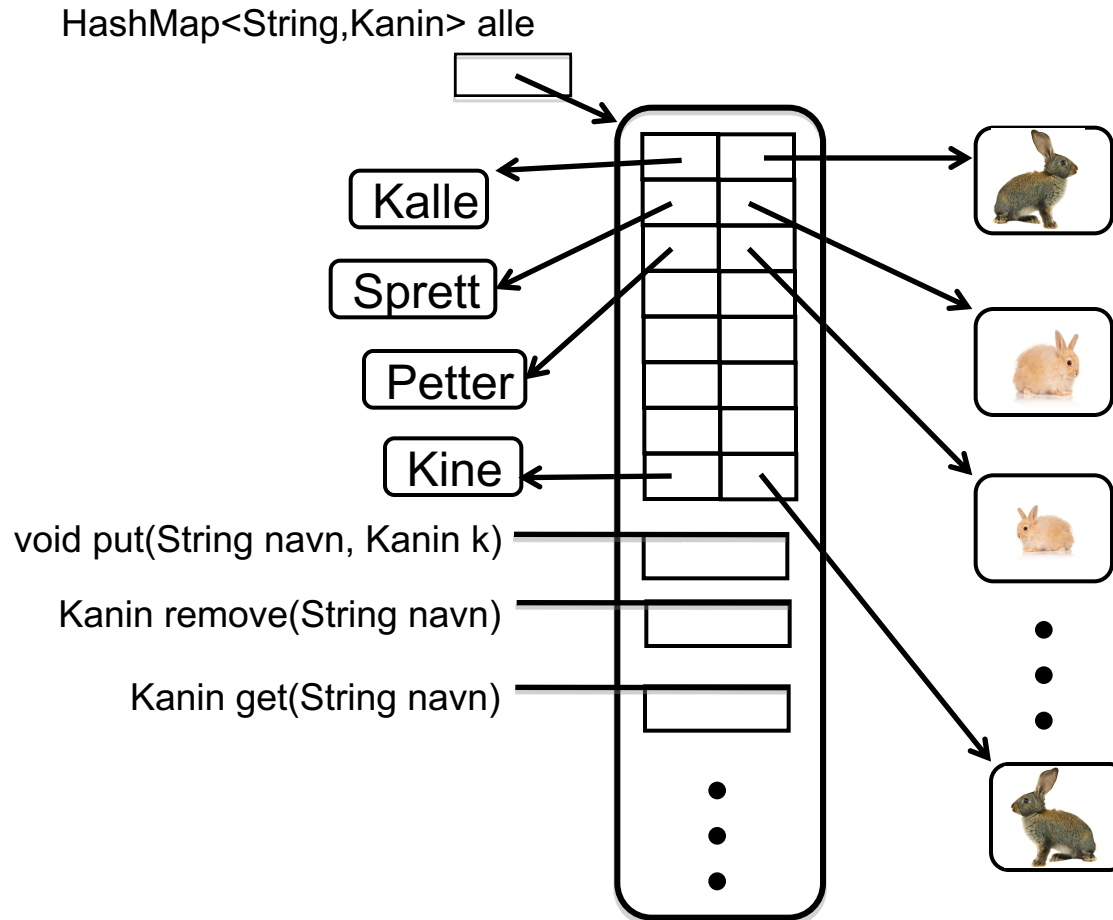
```
class KaningardTestArrayList {
    public static void main (String [ ] args) {
        ForenkletKaninGardAL mittKaninbur = new ForenkletKaninGardAL( );
        Kanin kalle = new Kanin("Kalle");
        mittKaninbur.settInn(kalle);
        Kanin sprett = new Kanin("Sprett");
        mittKaninbur.settInn(sprett);
        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);
        }
    }
}
```

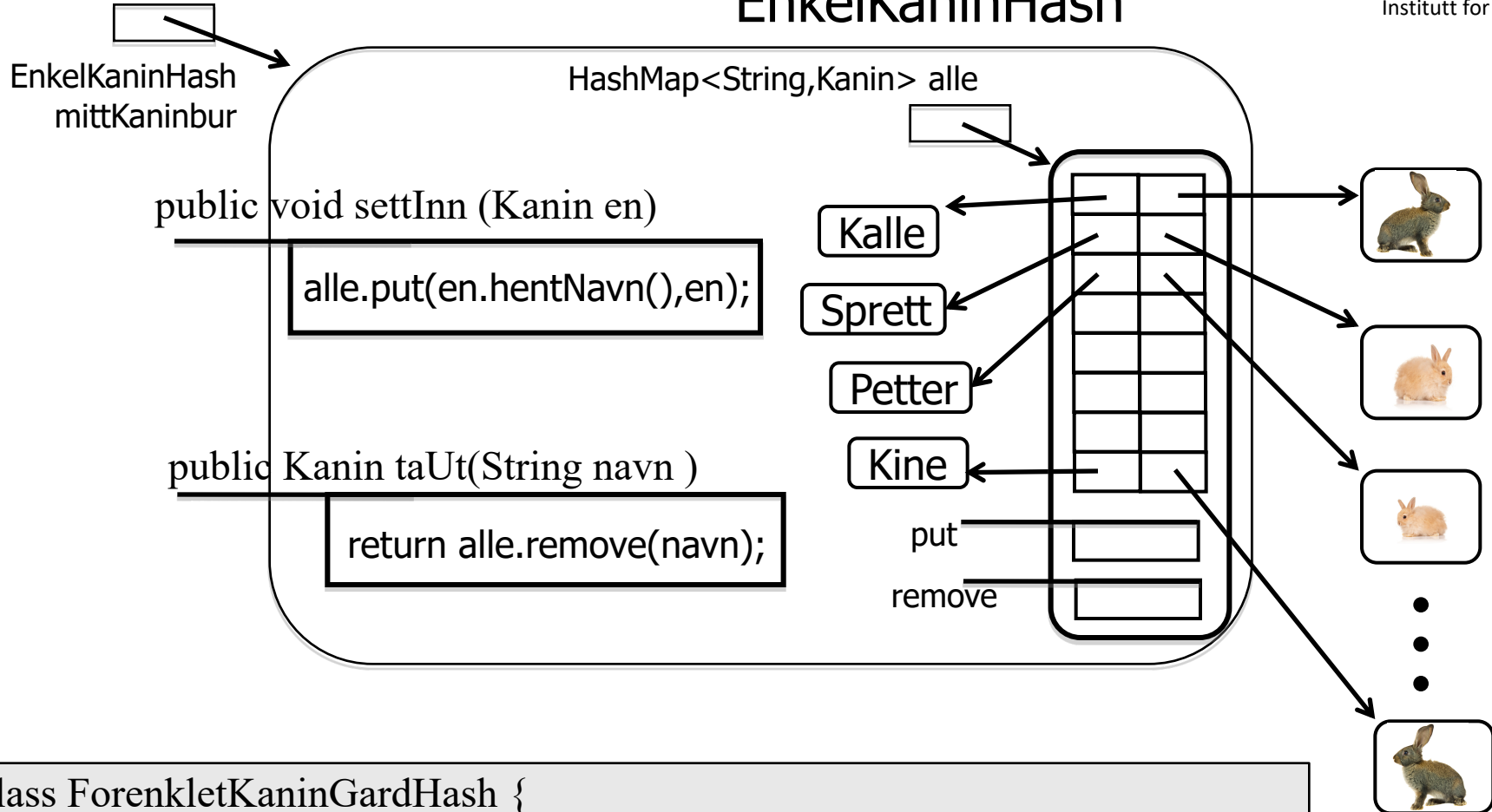
HashMap

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);
        }
    }
}
```