UNIVERSITETET I OSLO

Institutt for informatikk

# INF1010 våren 2017
# 25. januar

# Objektorientering i Java

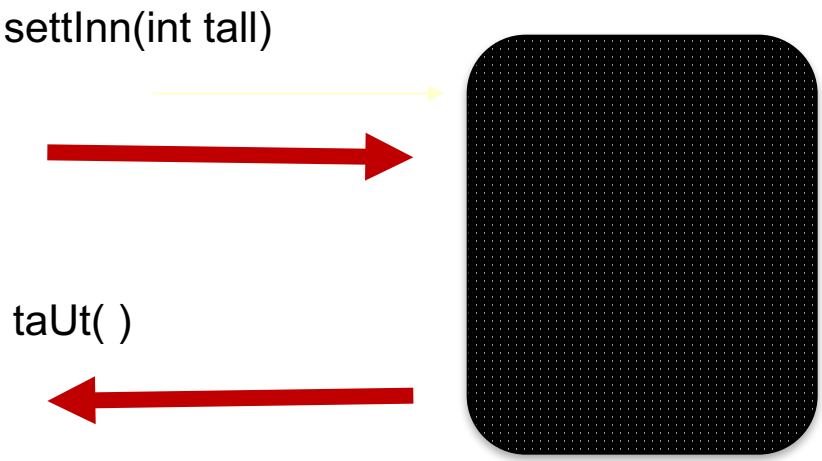
## Om enhetstesting

(Repetisjon av INF1000 og ”lær deg Java” for INF1001 og INF1100)

Stein Gjessing
Institutt for informatikk

# Hva er objektorientert programmering ?

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



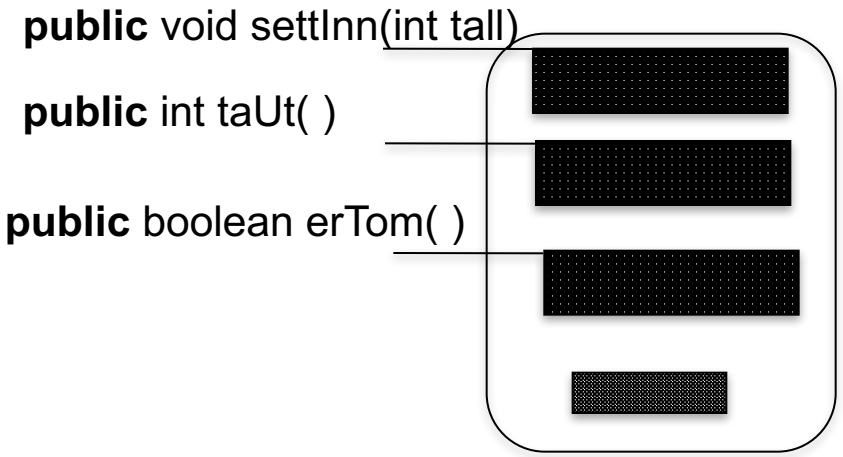

Masse “problemer”, selv med et så lite eksempel:

Hvordan virker settInn 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.

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




Ukjent implementasjon av metode

Ukjent implementasjon av metode

Ukjent implementasjon av metode

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

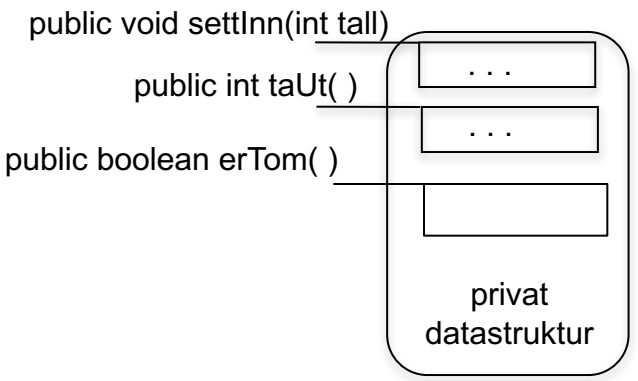


Dette er IKKE nytt.

Vi lager en klasse som vi kan lage objekter av:

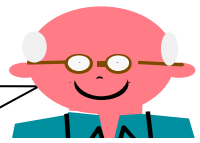
```
class EnkelHeltallsbeholder {

        private . . .

        public void settInn(int tall) {
                . . .
        }

        public int taUt( ) {
                . . .
        }

        public boolean erTom ( ) {
                . . .
        }
}
```

**new EnkelHeltallsbeholder()**
gir dette objektet:

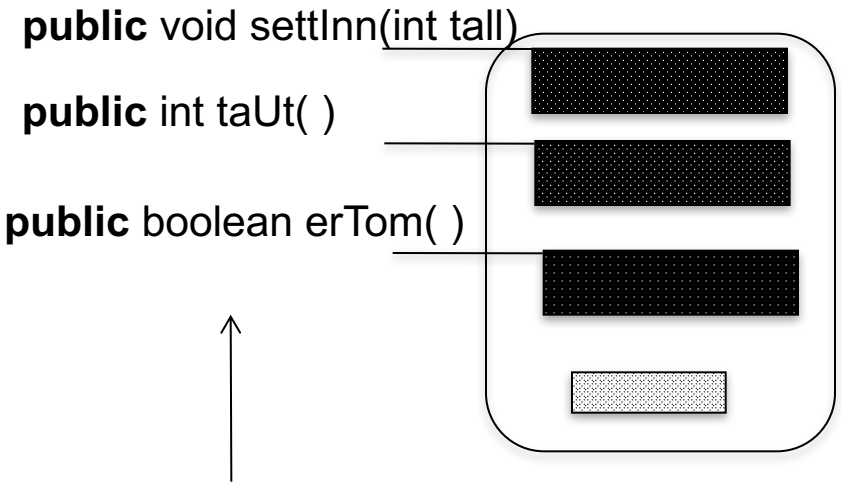



**Objekt** av klassen EnkelHeltallsbeholder

Dette er en historie fra virkeligheten !



# Metodenes **signaturer**

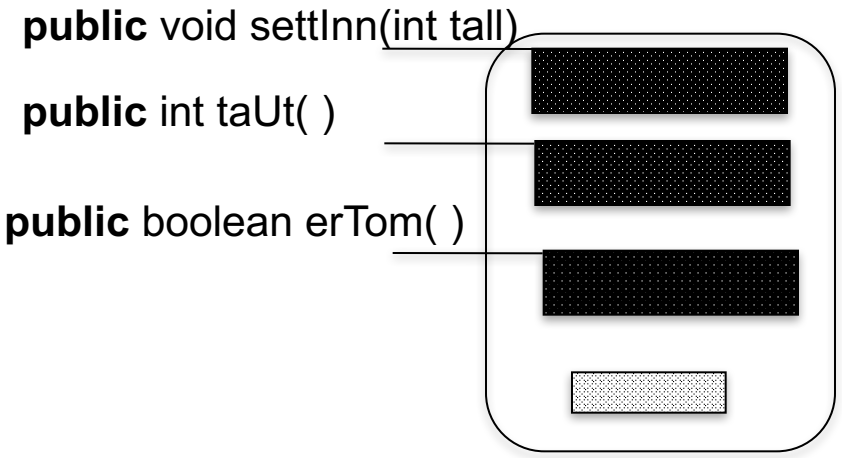



Dette kaller vi metodenes **signaturer** (skrivemåte, syntaks)

**Signaturen** til en metode er

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

# Metodenes **semantikk**

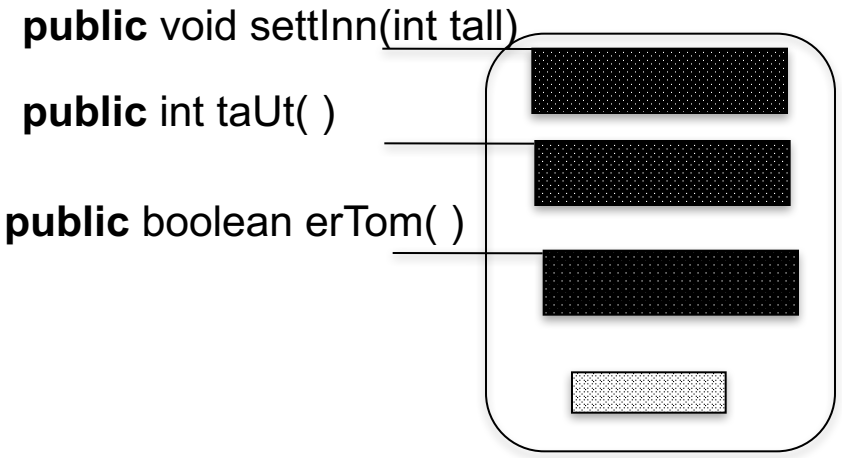



Hva gjør disse metodene? Hvordan virker de? Hva er sematikken til metodene?

**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 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 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, det er ikke Java-spesifikt.

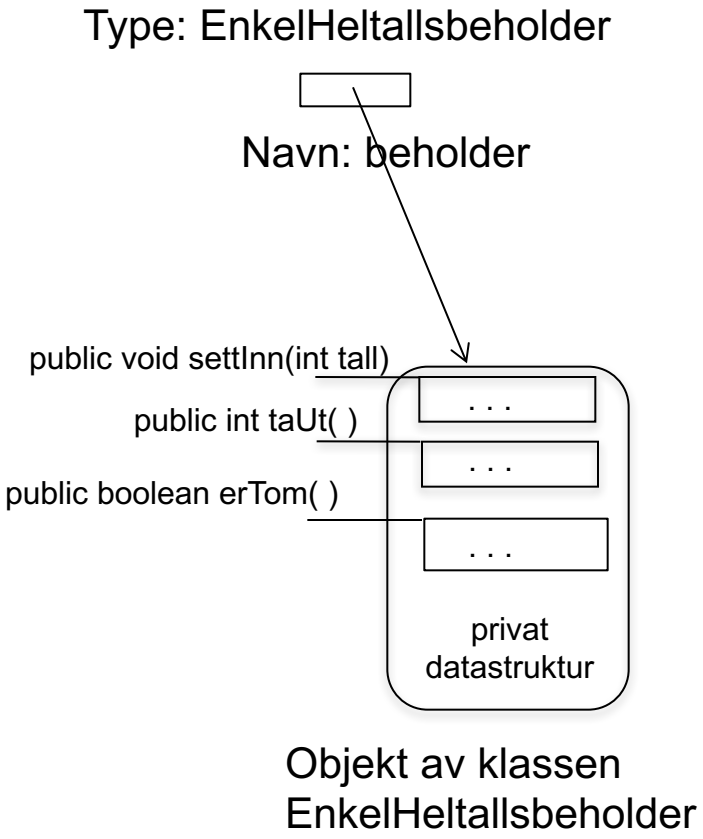
☺ Dette er en spøk. Informatikken har ikke nummererte lover

# 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();
              . . .
              . . .
              . . .
      }
}
```



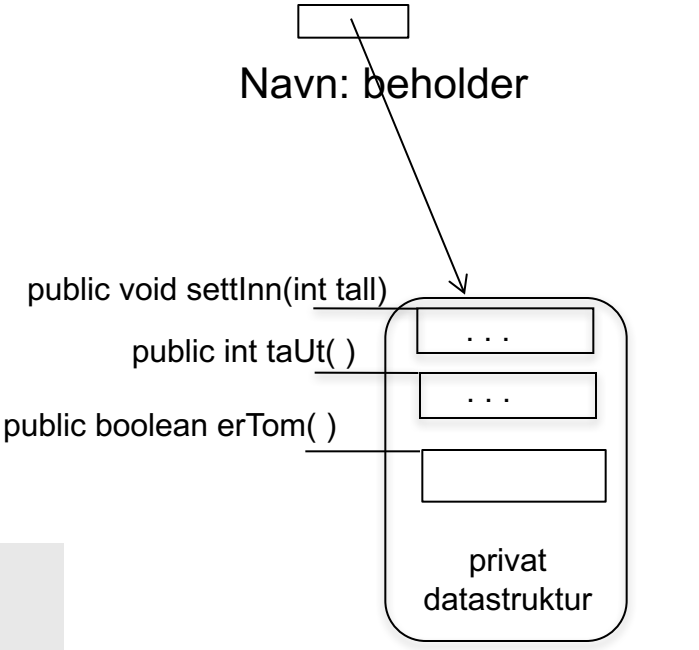

Objekt av klassen EnkelHeltallsbeholder

# 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




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”);}
      }
}
```

- mos:programmer steing$ java VeldigEnkelTestAvBeholder
- Riktig
- mos:programmer steing$

# Enkel test 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");}
    }
}
```

- Metoden "settInn" gjør at objektet tar vare på tallet som er parameter til metoden. Hvis det er et tall 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

```
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;
    }
}
```

```
mos:programmer steing$ java EnkelTestAvHeltallsbeholder
Riktig 1
Feil 2
Riktig 3
Feil 4
mos:programmer steing$
```

**Hvor er feilen ??**

# Med Javadoc (og uten ? feil)



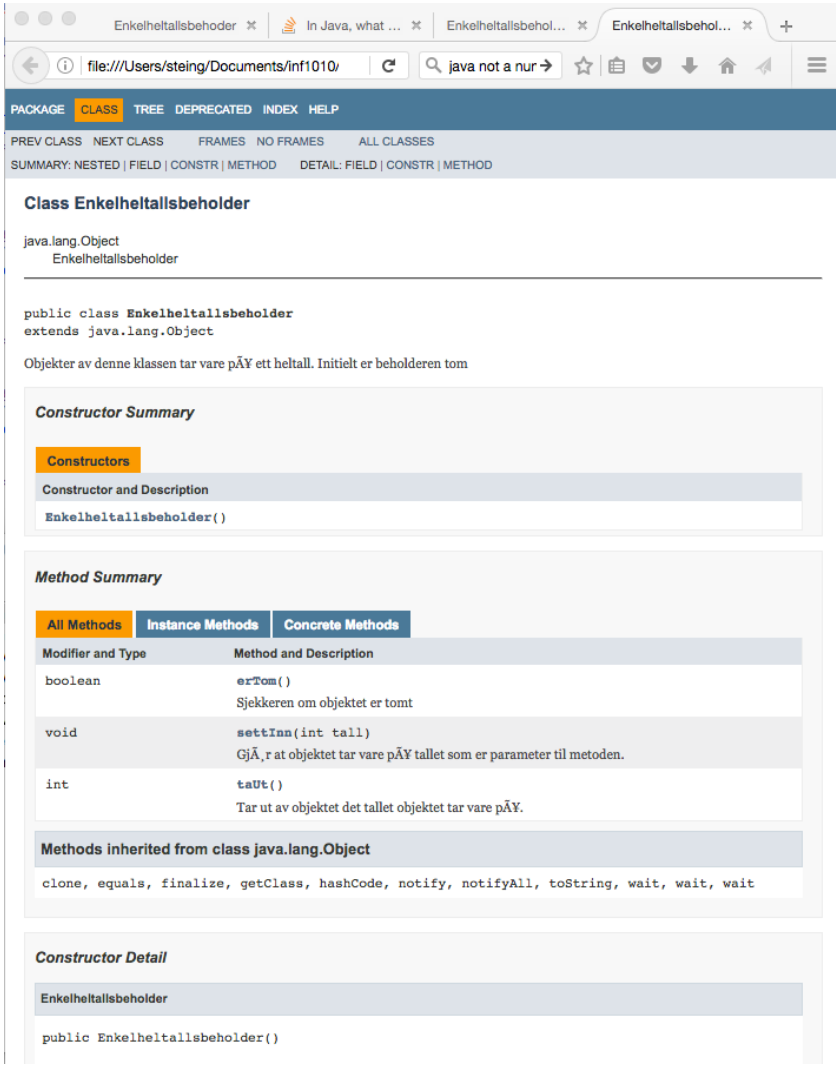
```
/** Objekter av denne klassen tar vare på
 *  ett  heltall.
 *  Initielt er beholderen tom
 *
 * @author Stein Gjessing
 * versjon 5. januar 2017
 */
public class Enkelheltallsbeholder {
     private boolean tom = true;
     private int tallet;
    /**
     * Gjør at objektet tar vare på tallet som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

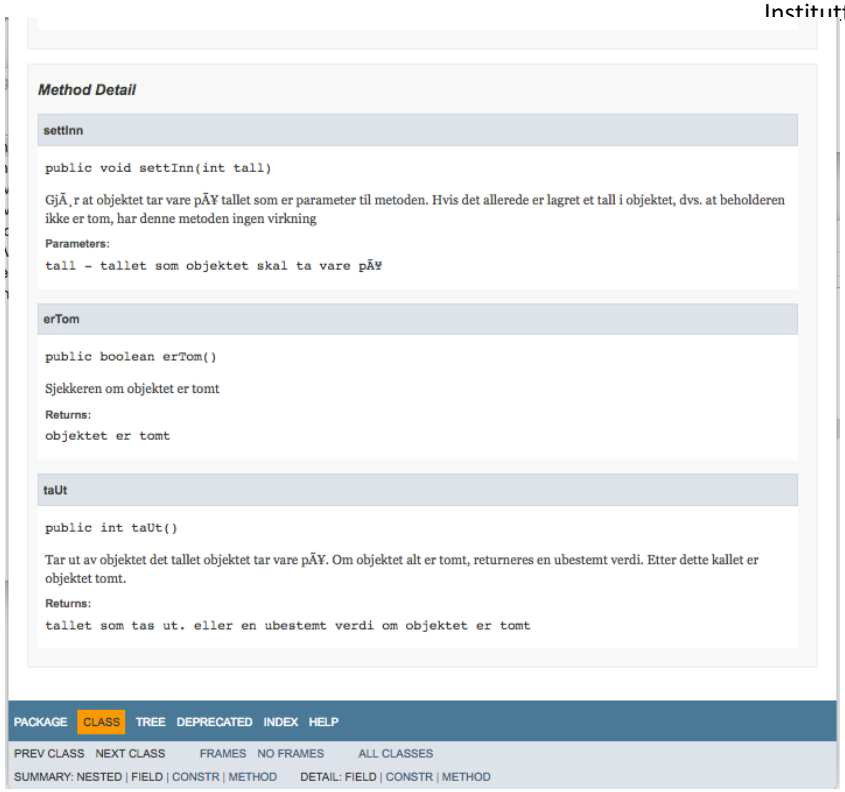
```
    /**
    * Sjekkeren om objektet er tomt
    *
    *@return objektet er tomt
    */
     public boolean erTom ( ) {
          return tom;
     }
    /**
     * Tar ut av objektet det tallet objektet
     *   tar vare på.
     * Om objektet alt er tomt, returneres en
     *  ubestemt verdi.
  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

UNIVERSITETET I OSLO

# Javadoc resultat

Institutt for informatikk




```
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;
      }
```

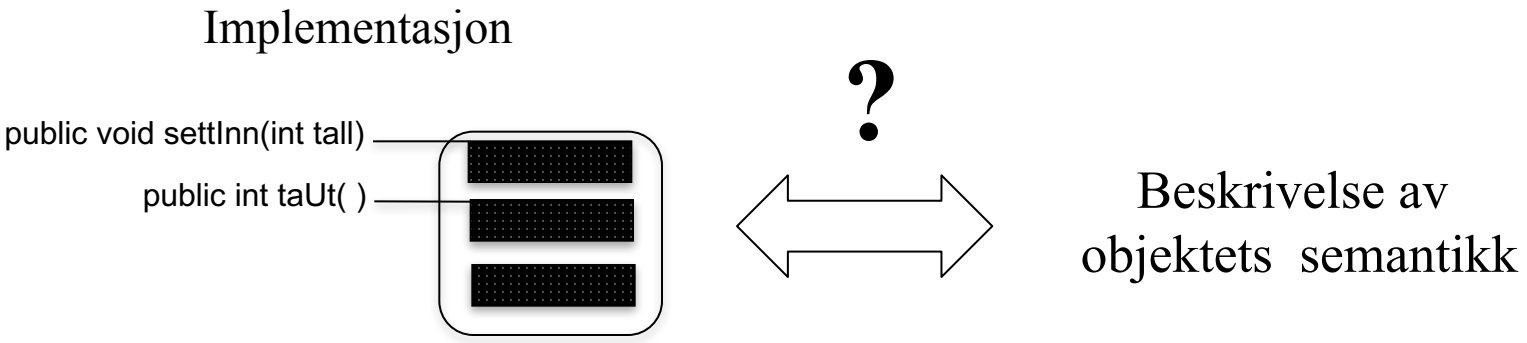
```
     /**
      * Sjekkeren om objektet er tomt
      *
      *@return objektet er tomt
      */
      public boolean erTom ( ) {
            return tom;
      }
     /**
      * Tar ut av objektet det tallet objektet
      *   tar vare på.
      * Om objektet alt er tomt, returneres en
      *  ubestemt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

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.

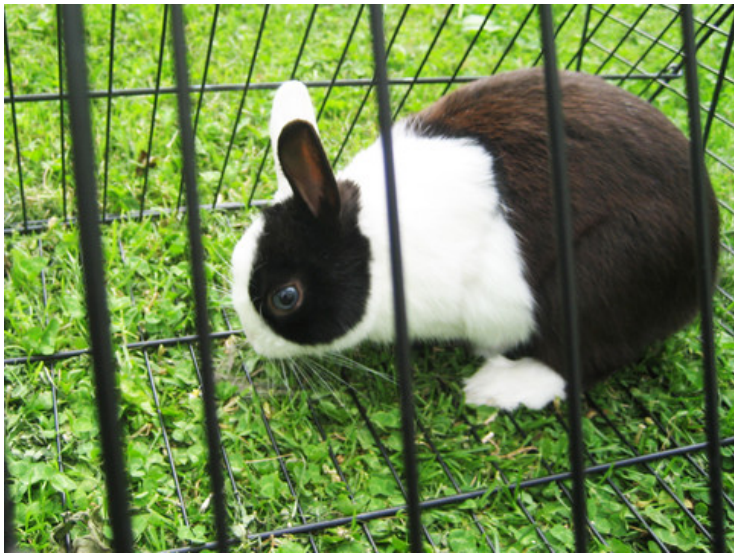



***De sematiske kravene kalles også en “kontrakt” (mellom brukerene av objektet og objektet selv)***

# Flere eksempler: Kaniner og kaninbur



```
class Kanin{
        String navn;
        Kanin(String nv) {navn = nv;}
}
```



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

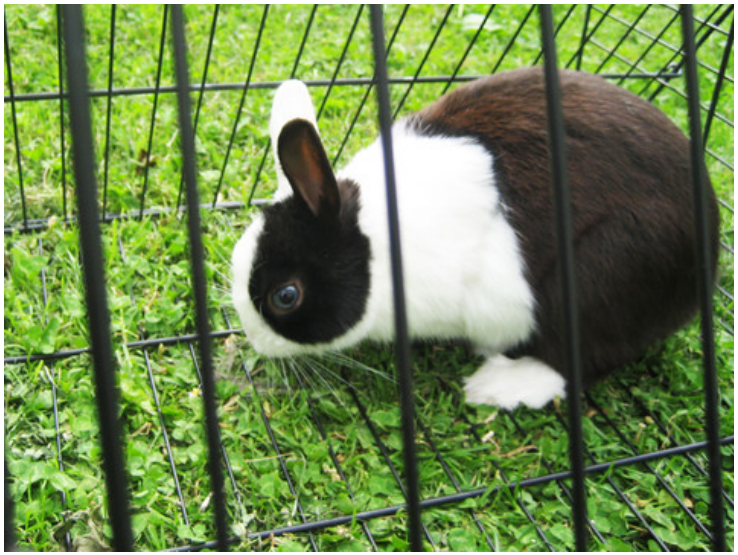
Semantikk:

- 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.**

# Kaniner og kaninbur: Full kode



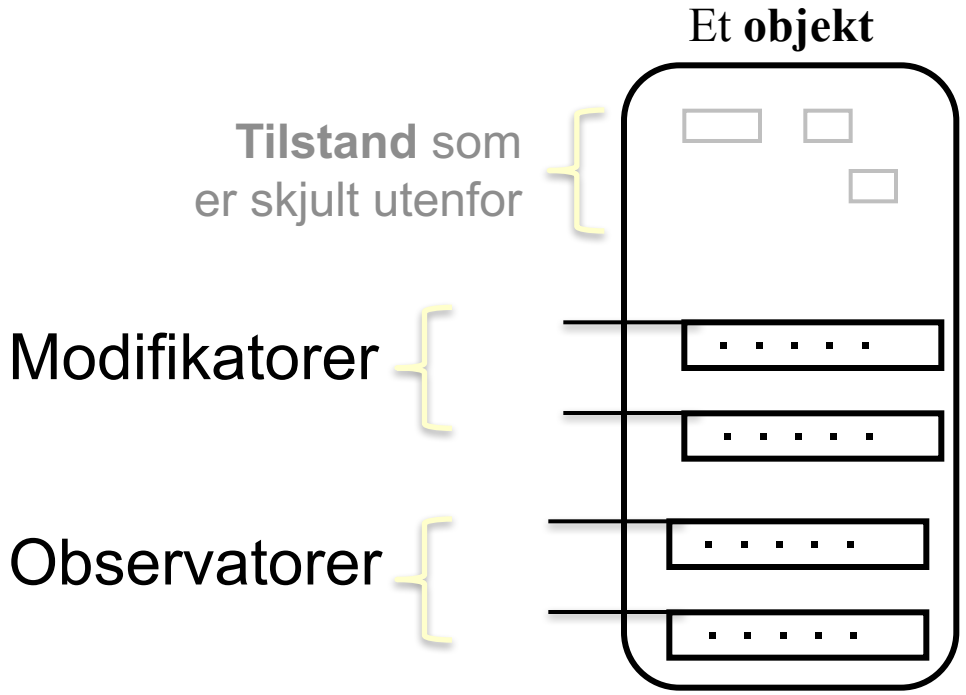
```
class Kanin{
        String navn;
        Kanin(String nv) {navn = nv;}
}
```



```
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



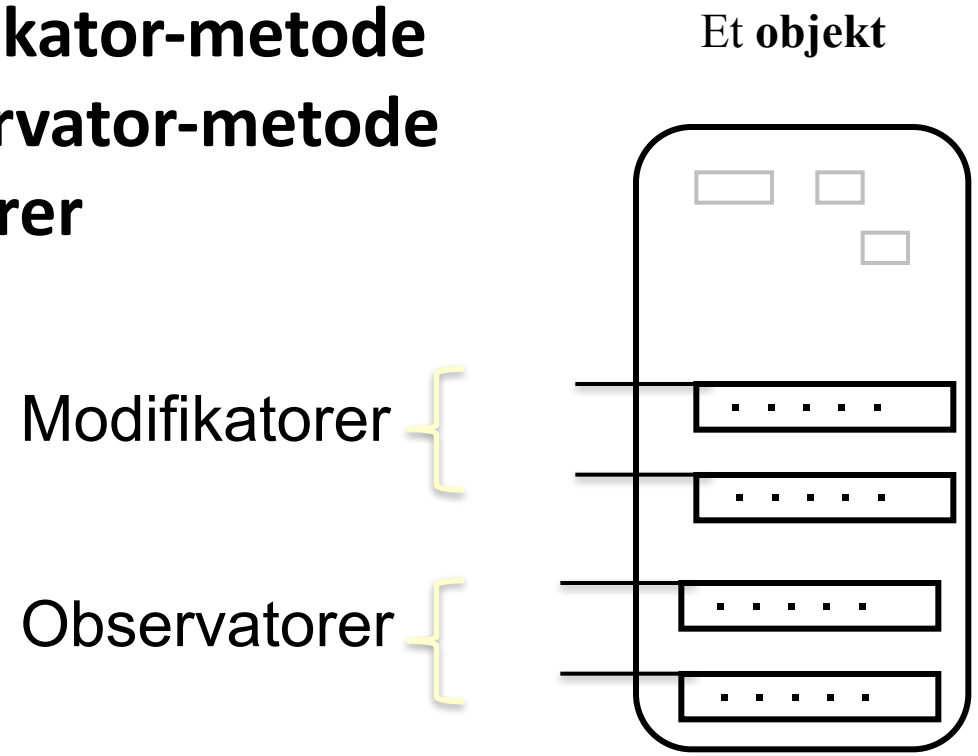

Modifikator, f.eks. set(),

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**




Modifikator, f.eks. set().

Observator, f.eks. les()

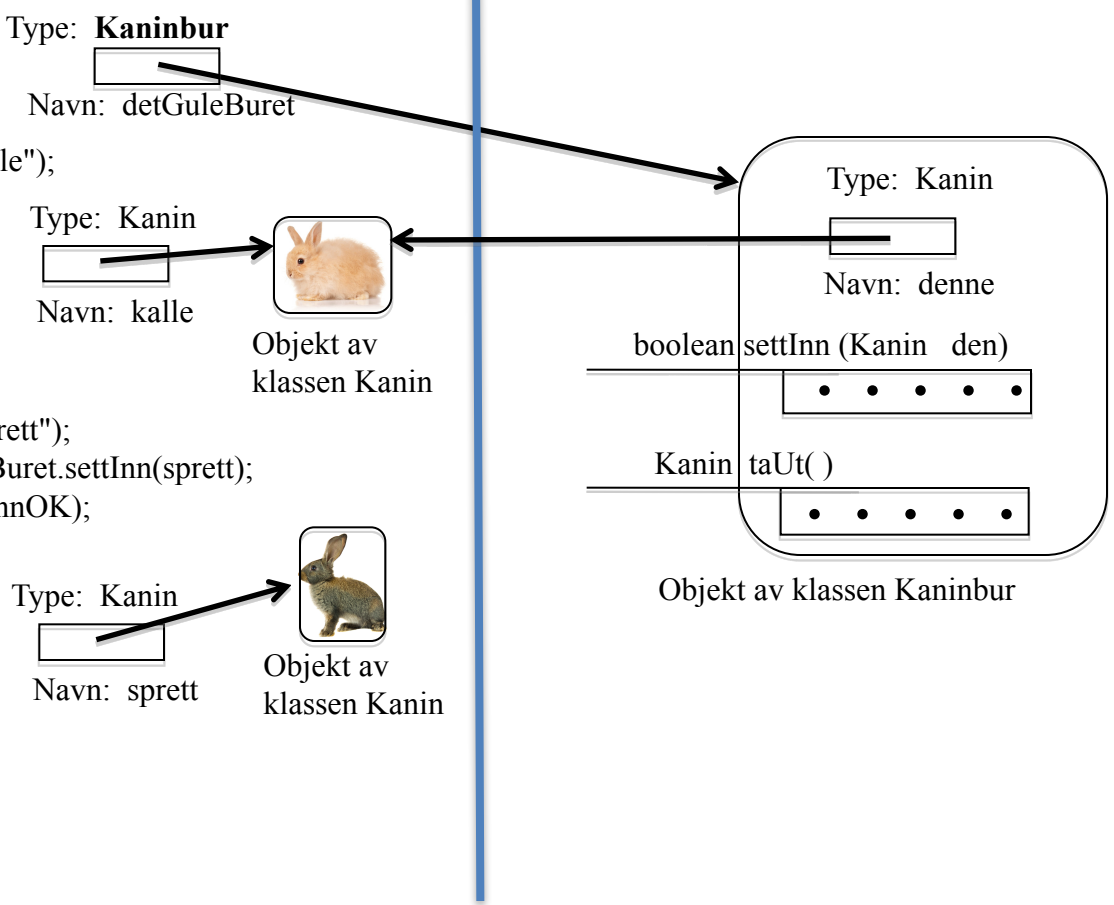
# Enhetstesting av class Kaninbur



```
class Kanin{
        String navn;
        Kanin(String nv) {navn = nv;}
}
```

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

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

    Kanin kalle = new Kanin("Kalle");

    detGuleBuret.settInn(kalle)

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



# Enhetstest av class Kaninbur



## Lag egne testmetoder (IKKE slik jeg har gjort her)

Kalle

```
class  LittenTestKaninbur {
      public static void main (String [ ] args) {
            Kaninbur detGuleBuret = new Kaninbur( );
            Kanin kalle = new Kanin("Kalle");
            detGuleBuret.settInn(kalle);
            Kanin sprett = new Kanin(”Sprett”);
            // tester at andre kanin ikke kommer inn i buret:
            booelan settInnOK = detGuleBuret.settInn(sprett);
            if (settInnOK)
                  { System.out.prinln(“Feil sett inn”); }
                  else {System.out.prinln(“Riktig sett inn”); }
            // tester at den som først ble satt inn nå tas ut:
            Kanin enKanin = detGuleBuret.taUt( );
            if (enKanin.navn.equals(“Kalle”)
                  { System.out.prinln(“Riktig ta ut”); }
                  else {System.out.prinln(“Feil  ta ut”); }
             }
}
```

Sprett

Pelle

Enhetstesting bør vi gjøre av alle klasser

# Eksempel på kaninbur til mange kaniner

3 observatorer

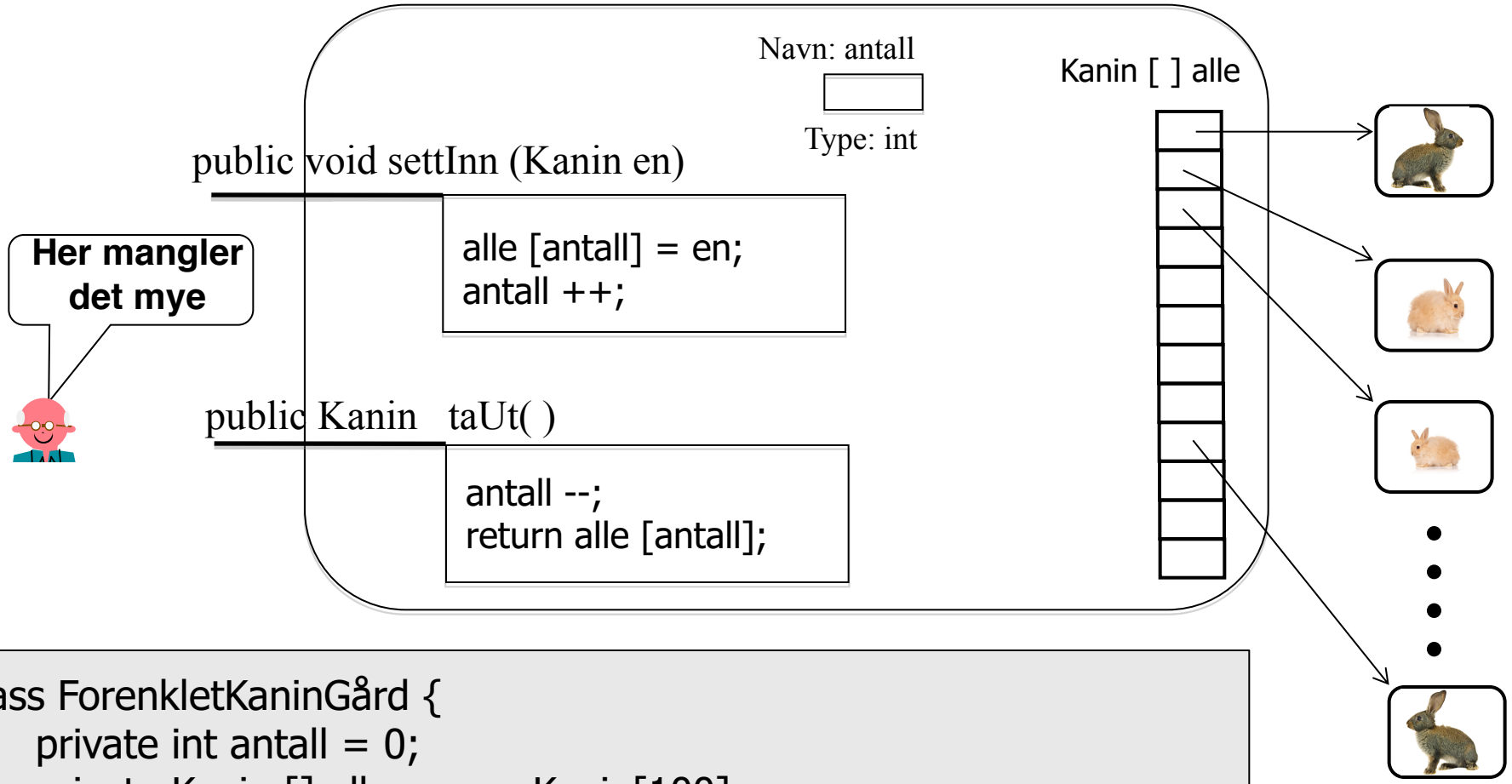
2 modifikatorer

```
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 peker) {alle[antall] = peker; antall ++; }

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

Oppgave:
skriv ferdig klassen KaninGård fra forrige side