UNIVERSITETET I OSLO

Institutt for informatikk

INF1010, 9. februar 2017

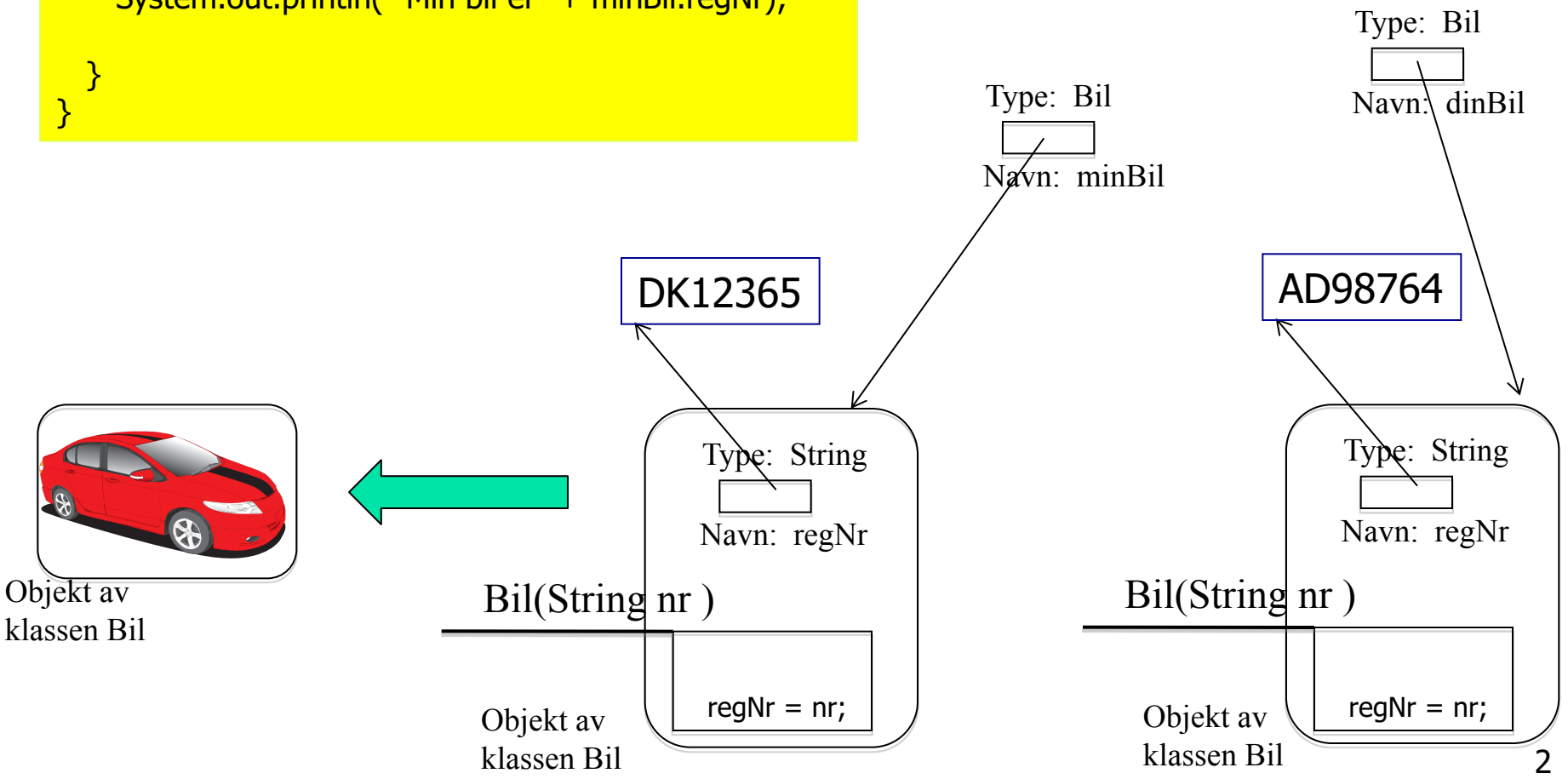
Litt om generiske klasser (mye mer neste uke)

Stein Gjessing
Inst. for Informatikk
Universitetet i Oslo

Først litt repetisjon


```
class LagBiler {
  public static void main(String[] args)  {
      Bil dinBil  = new Bil("AD98764");
      Bil minBil = new Bil("DK12365");
      System.out.println(" Din bil er " + dinBil.regNr);
      System.out.println(" Min bil er "+ minBil.regNr);

   }
}
```

```
class Bil {
   String regNr;
   Bil (String nr) { regNr = nr;}
}
```



Repetisjon fra gamle dager: ***Metoder*** *med parametre*

*En **metode** uten parameter:*

```
int kvadratAv5 ( ) {
        return 25;
}
```

*En **metode** med parameter:*

```
int kvadrat (int tall) {
        return tall * tall;
}
```

tall er **formell** parameter

Definerende forekomst av den formelle parameteren tall

Bruks-forekomst av den formelle parameteren tall

```
int svar = kvadrat(7);
```

7 er **aktuell** parameter

Klasser med parametre – generiske klasser

- Nytt for ca. 10 år siden (Java 1.5)
  - Metoder med parametre har eksistert "alltid"
- Klasser med parametre kalles også
  - Generiske klasser
  - Generiske typer
- Ikke 100% vellykket i Java
  - Pga. at at det ikke var lett å få Java Virtual Machine til å kjøre dette.
- Derfor: Bare enkel bruk av generiske klasser er pensum i INF1010.
- Kapittel 18.1 og 18.2 i Horstmann

Vi har alt sett klassen HashMap med parametre

```
HashMap <String, Bil> register;
register = new HashMap <String,Bil> ( );

Bil bl  = new Bil(”DE25132”);

register.put (”DE25132”, bl);
```

```
class Bil {
   String regNr;
   Bil (String nr) { regNr = nr;}
}
```

String og Bil er **aktuelle** parametre

K og V er **formelle** parametre

I Java-biblioteket:

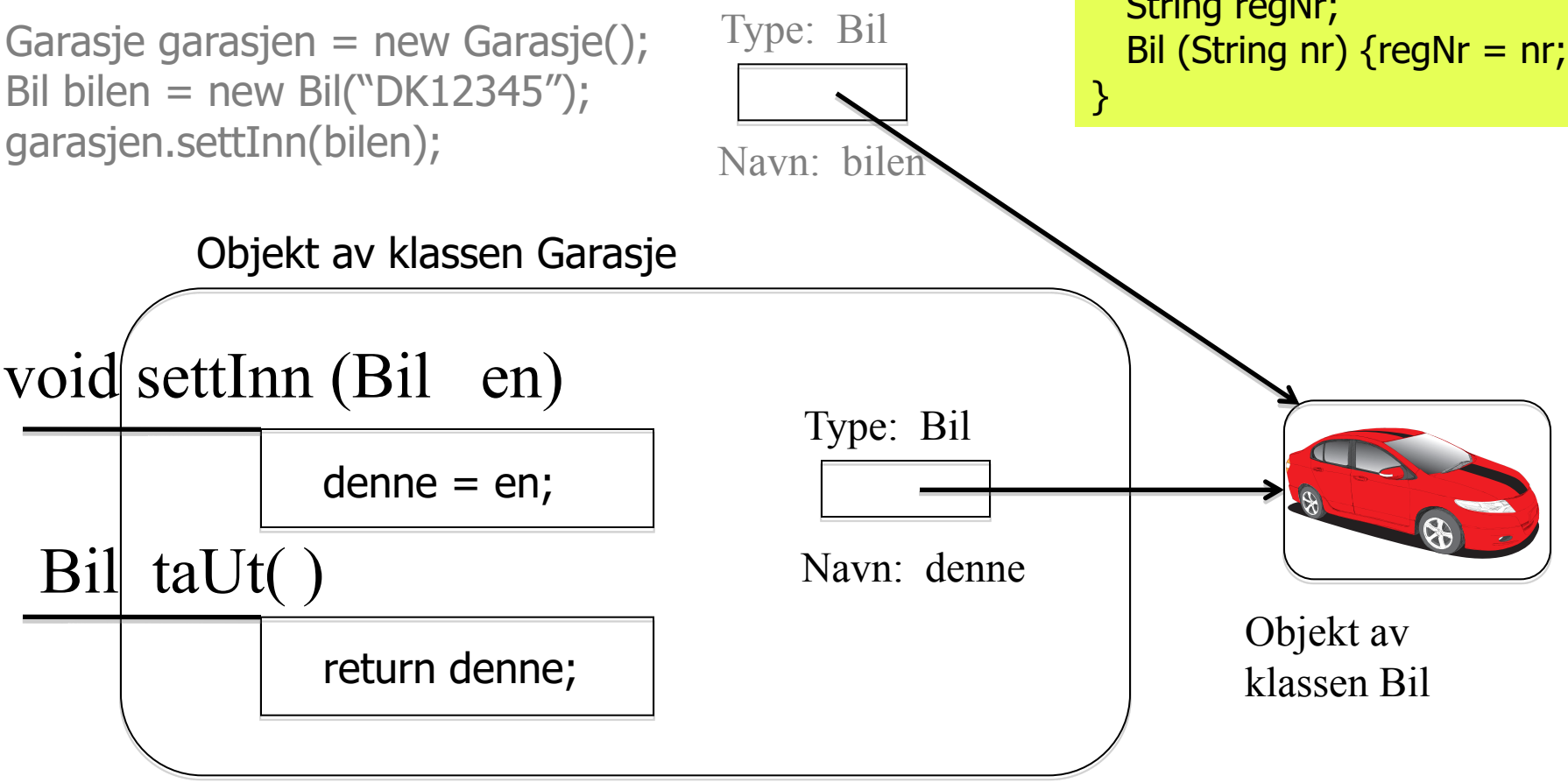
```
class HashMap<K,V> {  . . . }
```

Ikke noe nytt her


```
class Garasje {
        private Bil denne;
        public void settInn (Bil en) { denne = en;}
        public Bil taUt ( ) {return denne;}
}
```

```
class Bil {
   String regNr;
   Bil (String nr) {regNr = nr;}
}
```

```
Garasje garasjen = new Garasje();
Bil bilen = new Bil("DK12345");
garasjen.settInn(bilen);
```

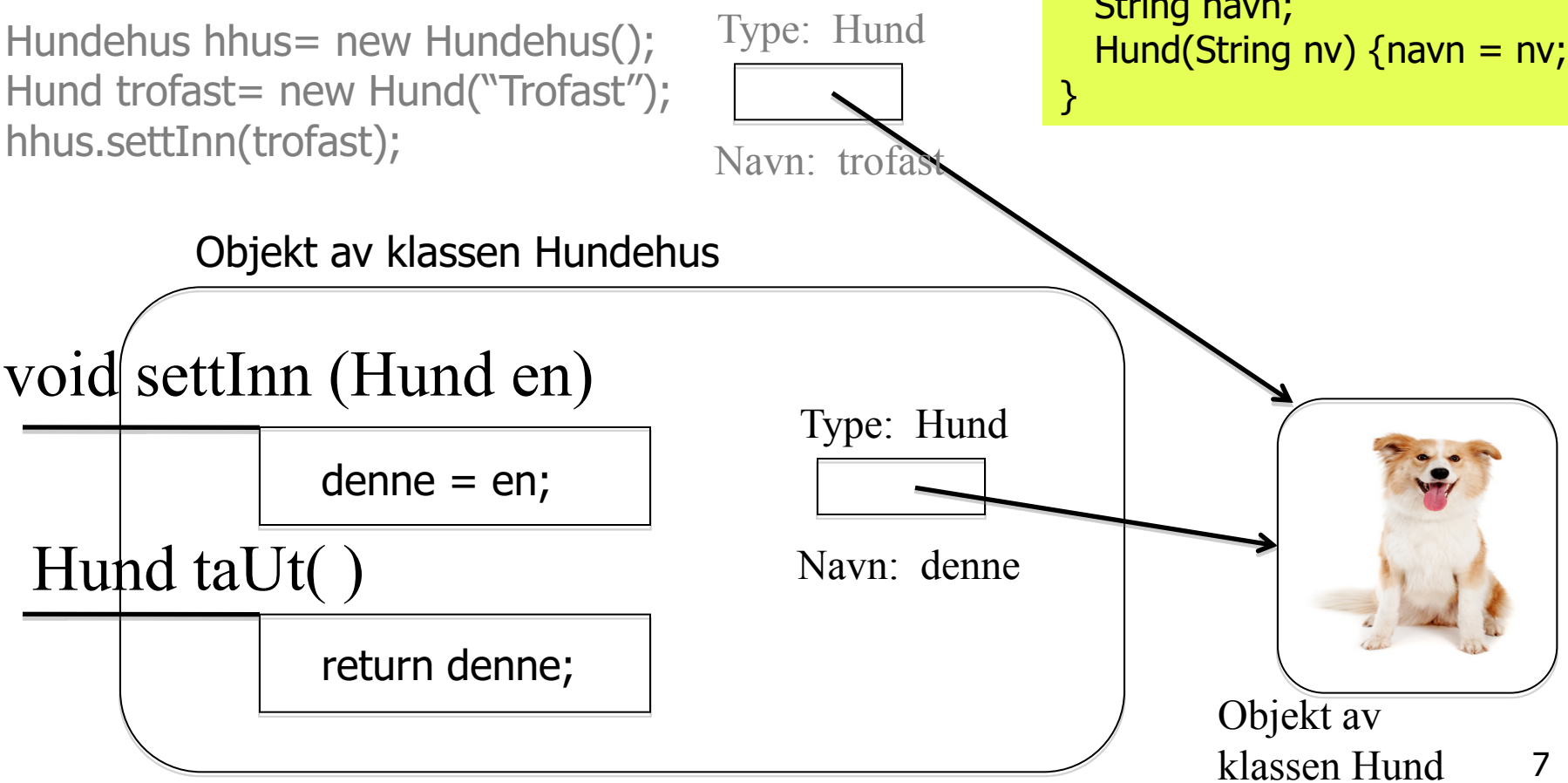


Ikke noe nytt her

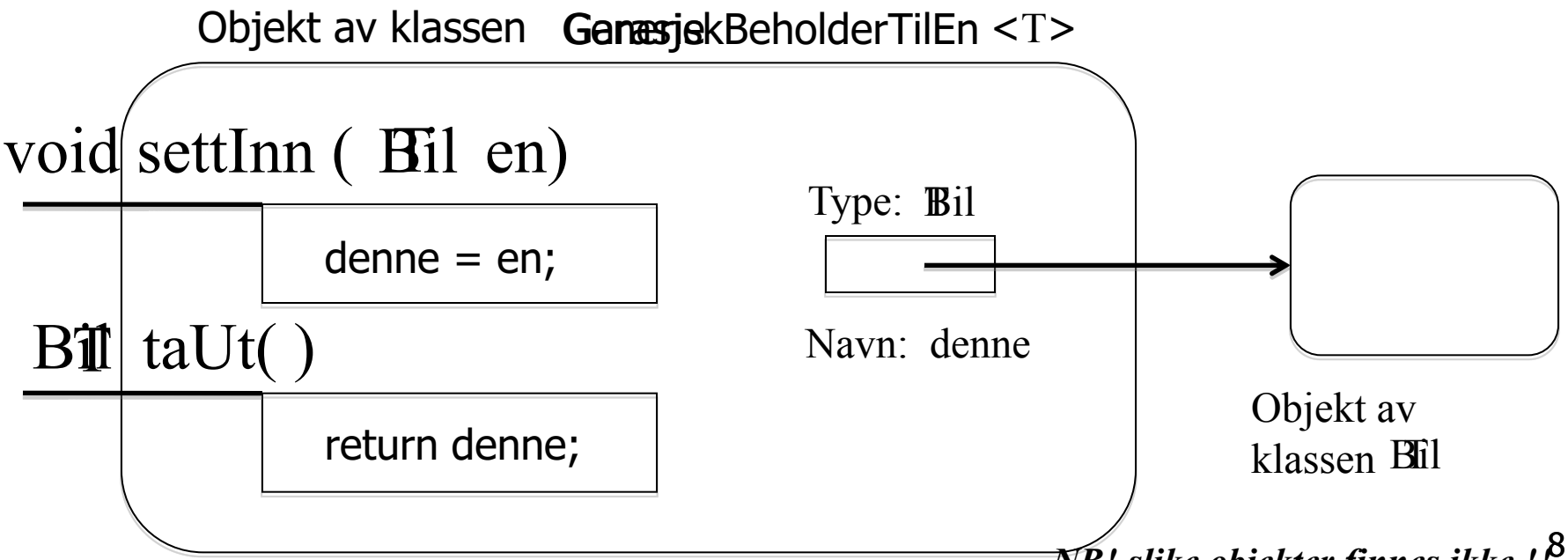

```
class Hundehus {
        private Hund denne;
        public void settInn (Hund en) { denne = en;}
        public Hund taUt ( ) {return denne;}
}
```

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

```
Hundehus hhus= new Hundehus();
Hund trofast= new Hund("Trofast");
hhus.settInn(trofast);
```



```
class GarasjeBeholderTilEn <T> {
        private Bil denne;
        public void settInn (Bil en){denne = en;}
        public Bil taUt(){return denne;}
}
```

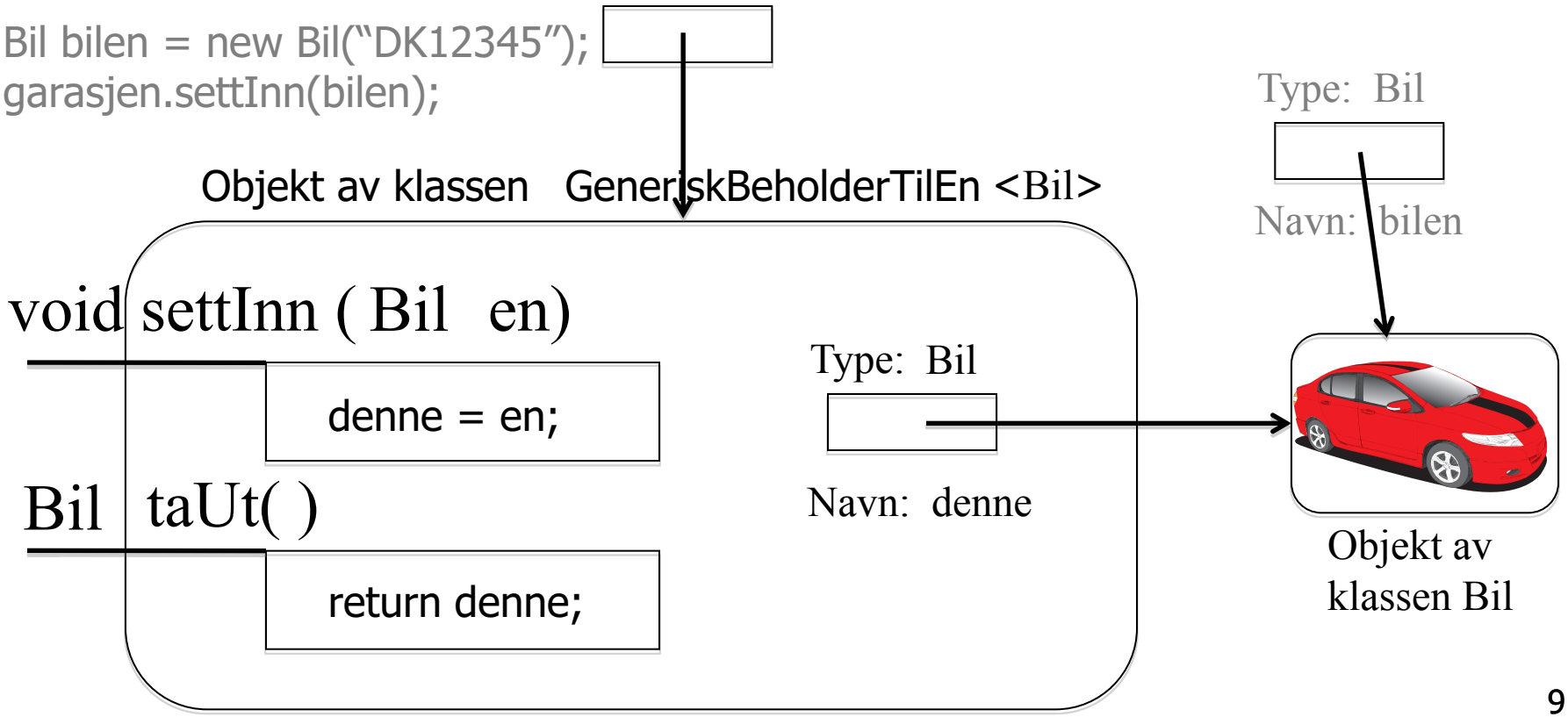


NB! slike objekter finnes ikke !!

```
class GeneriskBeholderTilEn <T> {
        private T   denne;
        public void settInn (T  en) { denne = en;}
        public T  taUt ( ) {return denne;}
}
```

```
class Bil {
   String regNr;
   Bil (String nr) {regNr = nr;}
}
```

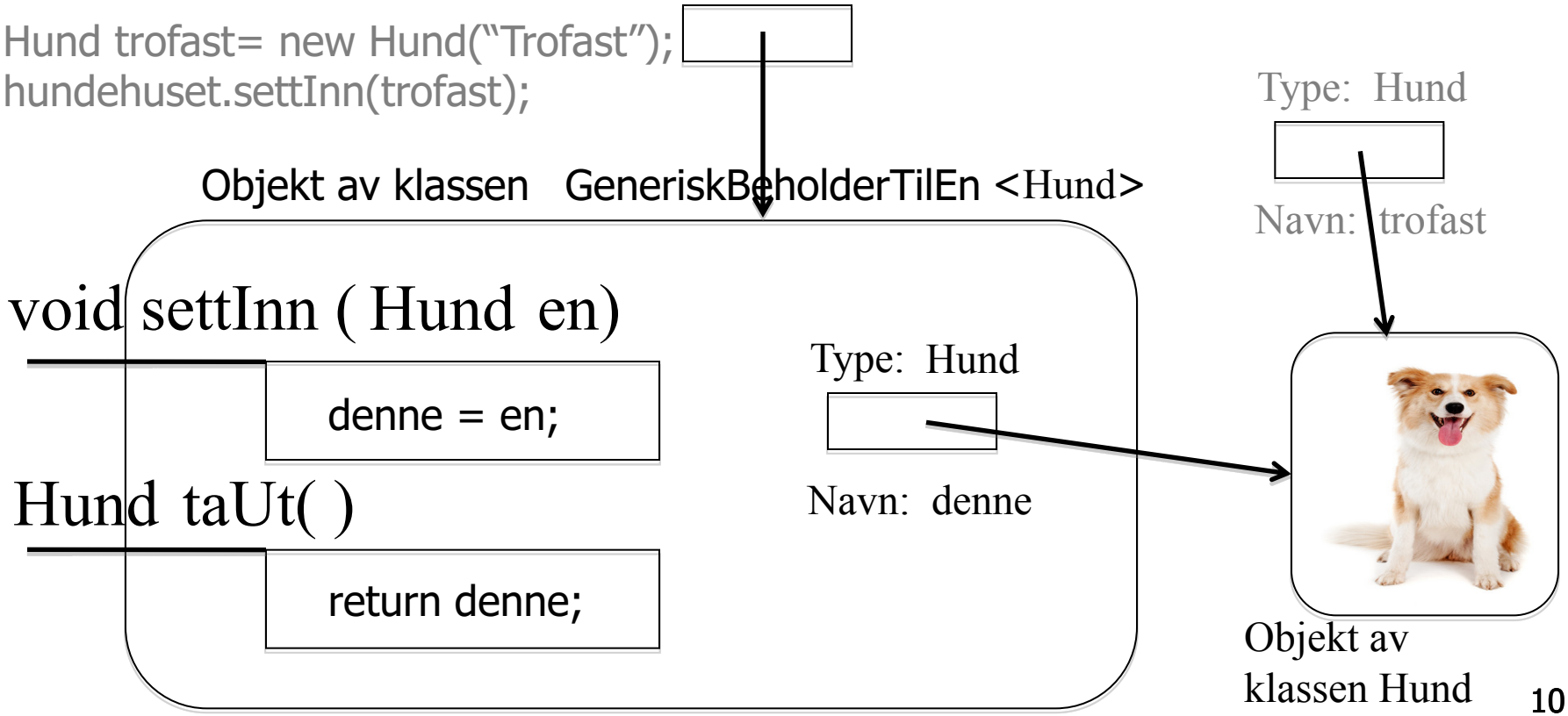
```
GeneriskBeholderTilEn<Bil> garasjen = new GeneriskBeholderTilEn<Bil>();
Bil bilen = new Bil("DK12345");
garasjen.settInn(bilen);
```



```
class GeneriskBeholderTilEn <T> {
        private T   denne;
        public void settInn (T  en) { denne = en;}
        public T  taUt ( ) {return denne;}
}
```

```
class Hund {
   String navn;
   Bil (String nv) {navn = nv;}
}
```

```
GeneriskBeholderTilEn<Hund> hundehuset= new GeneriskBeholderTilEn<Hund>();
Hund trofast= new Hund("Trofast");
hundehuset.settInn(trofast);
```



Generisk eller generell eller parametrisert

versjon av den enkle beholderen

(T er et navn vi velger selv (T for "Type"))

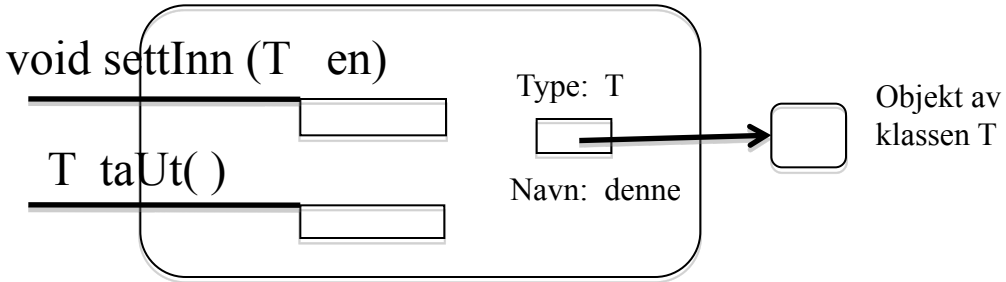
T er formell parameter

Bruks-forekomster av den formelle parameteren T

Definerende forekomst av den formelle parameteren T

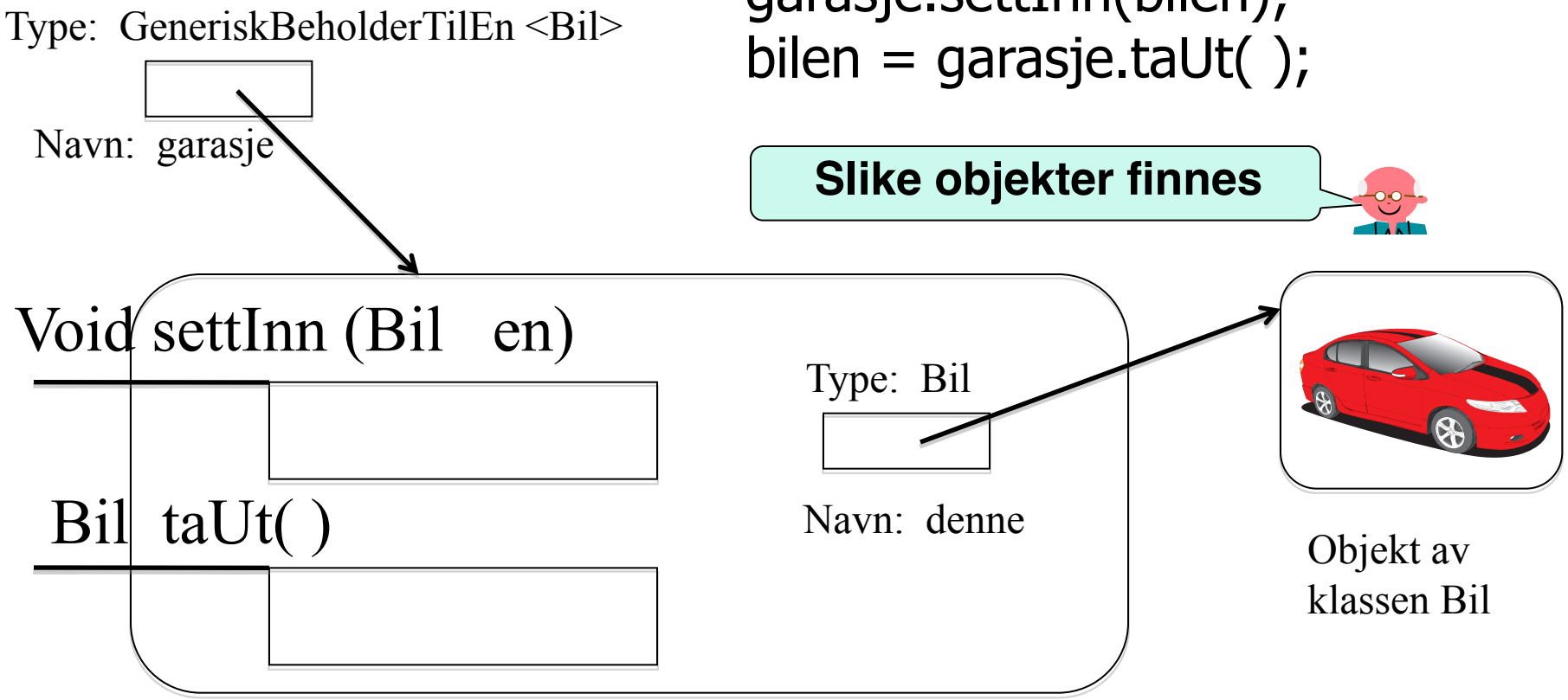
```
public class  GeneriskBeholderTilEn  <T>  {
              private T denne;
              public void settInn (T en)  {  denne = en;  }
              public T taUt ( ) {  return denne;   }
        }
```

NB! slike objekter finnes ikke !!



GeneriskBeholderTilEn <Bil> garasje = new GeneriskBeholderTilEn <Bil> ( );

```
Bil bilen  = new Bil();
garasje.settInn(bilen);
bilen = garasje.taUt( );
```

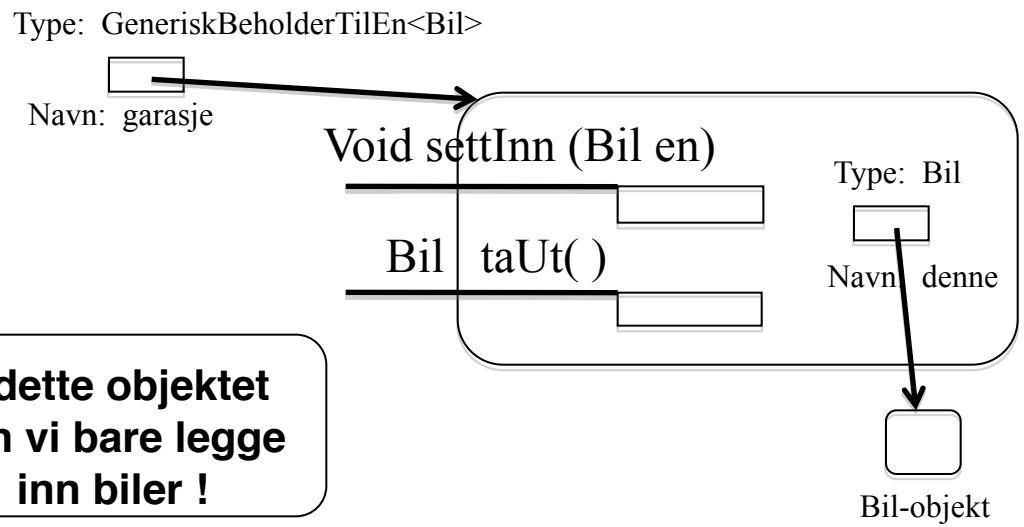


Formell parameter

```
public class  GeneriskBeholderTilEn  <T>  {
        private T denne;
        public void leggInn (T en)  { denne= en; }
        public T hent () {    return denne;  }
}
```

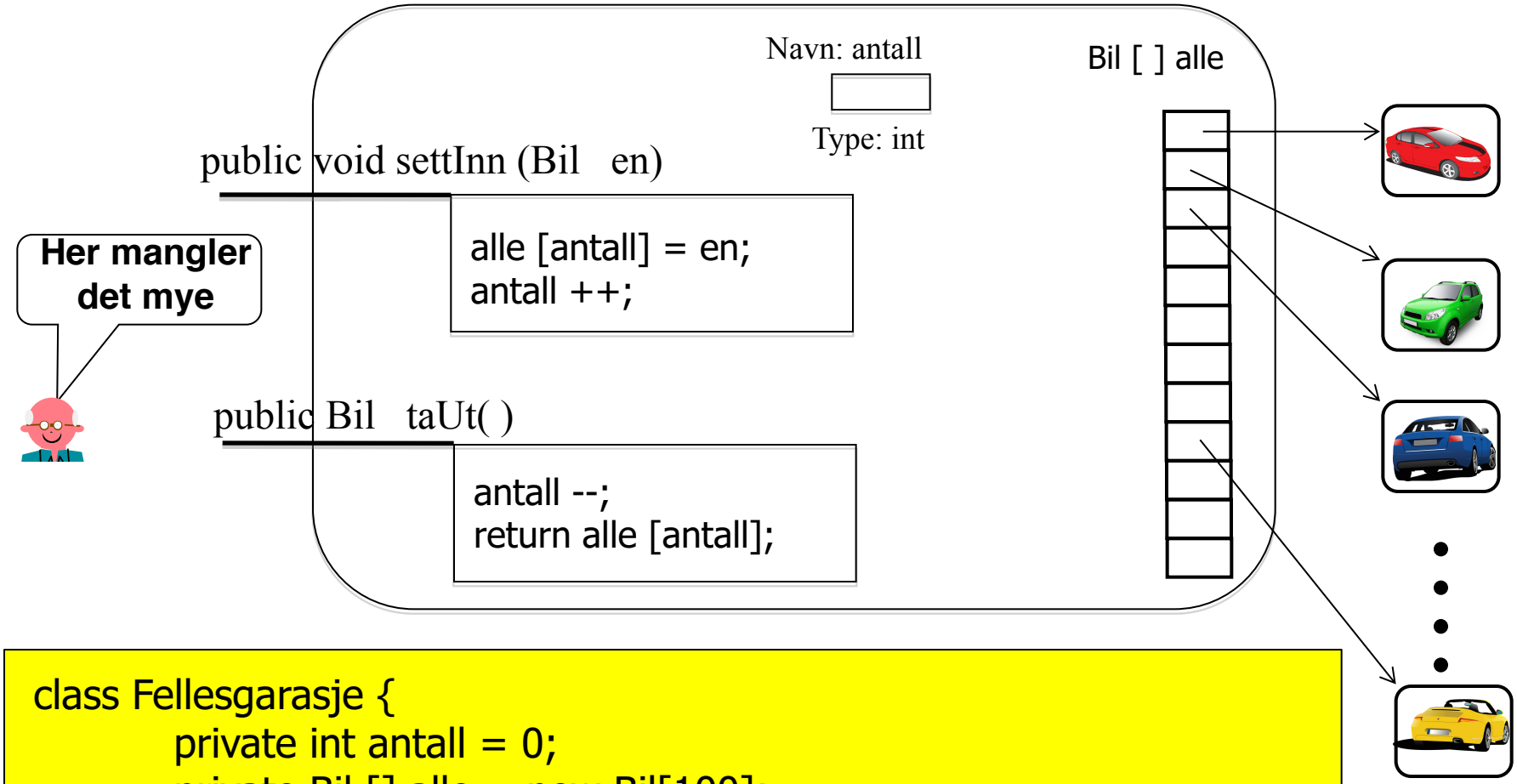
Aktuell parameter

```
class Bil {. . .}

public class EnkelGenDemo {
  public static void main(String[] args) {
     GeneriskBeholderTilEn <Bil> garasje = new GeneriskBeholderTilEn <Bil> ();

     Bil bilen  = new Bil();
     garasje.settInn(bilen);

     Bil  bilTo = garasje.taUt();

  }
}
```



Ikke noe nytt her

En stor beholder: En Fellesgarasje

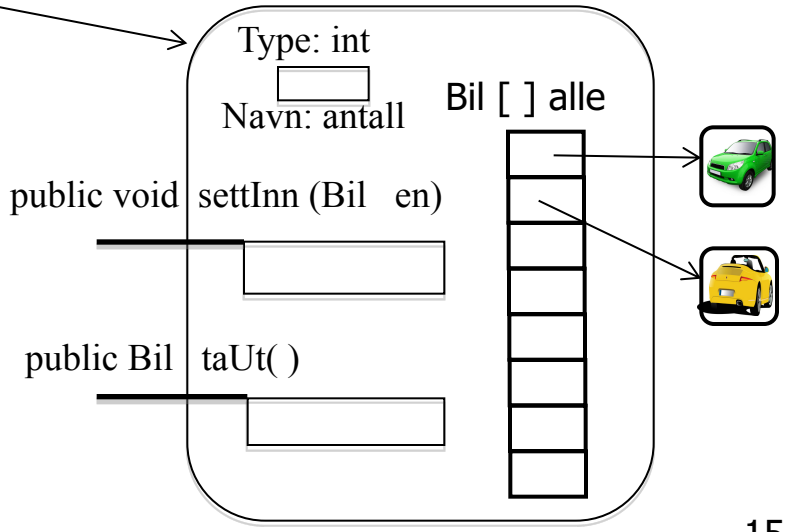



```
class Fellesgarasje {
        private int antall = 0;
        private Bil [] alle = new Bil[100];

        public void settInn(Bil peker) {alle[antall] = peker; antall ++; }

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

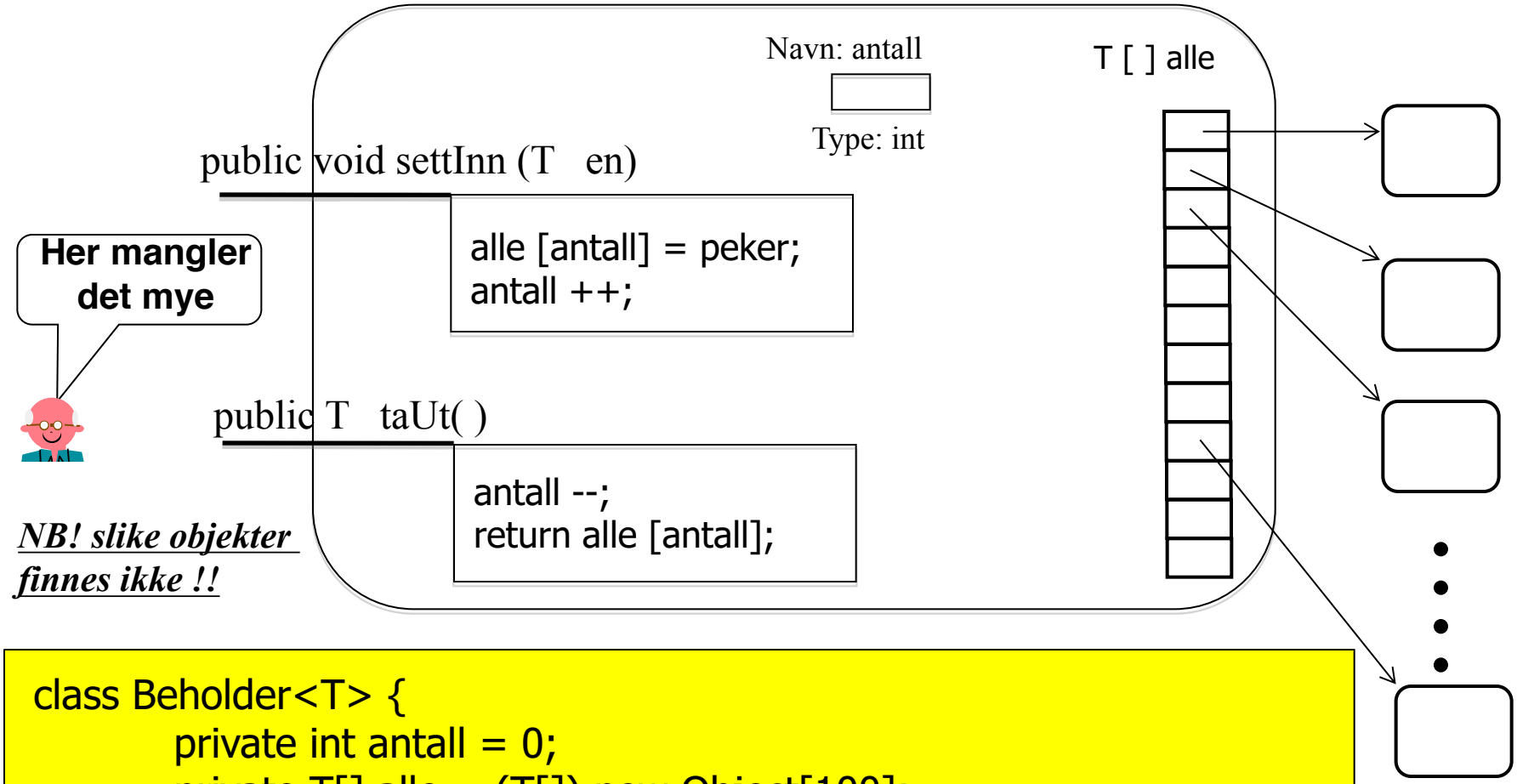
```
class Fellesgarasje {
        private int antall = 0;
        private Bil [] alle = new Bil[100];

        public void settInn(Bil peker) {alle[antall] = peker; antall ++; }

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

```
class Bil {
   String regNr;
   Bil (String nr) {
        regNr = nr;
    }
}
```

```
class BrukGarasje {
  public static void main(String[] args)  {
     Fellesgarasje  fellesGarasjen =
                          new Fellesgarasje( );

     Bil dinBil  = new Bil("AD98764");
     Bil minBil = new Bil("DK12365");
     fellesGarasjen.settInn(minBil);
     fellesGarasjen.settInn(dinBil);
     fellesGarasjen.taUt();
     Bilen denBilen = fellesGarasjen.taUt();
     System.out.println(" Bilen som kjørte sist ut var: "
                    + denBilen.regNr);
   }
}
```



En generell stor beholder:




```
class Beholder<T> {
        private int antall = 0;
        private T[] alle = (T[]) new Object[100];

        public void settInn(T peker) {alle [antall] = peker; antall ++; }

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

"unchecked cast"

Terminal — bash — 83×11

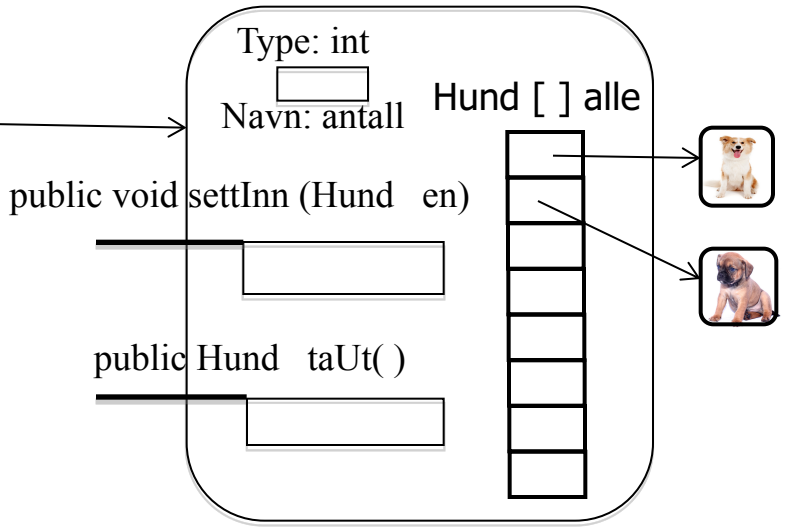
```
ammoniake:programmer steing$ javac Hoved.java
Note: Hoved.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
ammoniake:programmer steing$ javac -Xlint:unchecked Hoved.java
Hoved.java:23: warning: [unchecked] unchecked cast
found   : java.lang.Object[]
required: T[]
          T[] alle = (T[]) new Object[100];
                           ^
1 warning
ammoniake:programmer steing$
```

Ikke bry deg om dette.
Grunnen er at under kjøring vet Java ikke hva slags klasse som brukes inne i objekter av den generiske typen.
Dette er en "feil" i Javas kjøresystem.

Litt mer om dette senere i semesteret.

Da kan vi f.eks. skrive:

```
Beholder<Hund> minHundegard = new Beholder<Hund> ( );

Hund passopp = new Hund("Passopp");
Hund trofast   = new Hund("Trofast");
minHundegard.settInn(passopp);
minHundegard.settInn(trofast);
Hund drittbikkje;
drittbikkje = minHundegard.taUt();
System.out.println(" Drittbikkja heter: "
                        + drittbikkje.navn);
```




```
Beholder<Bil> fellesGarasjen = new Beholder<Bil> ( );

Bil dinBil  = new Bil("AD98764");
Bil minBil = new Bil("DK12365");
fellesGarasjen.settInn(minBil);
fellesGarasjen.settInn(dinBil);
fellesGarasjen.taUt();
Bilen denBilen = fellesGarasjen.taUt();
System.out.println(" Bilen som kjørte sist ut var: "
                        + denBilen.regNr);
```