Prøveeksamen

- Hva slags oppgaver ble gitt?
- Hvordan løser man dem?

Oppgavetype 1: *Hva skjer?*

Du får se et kort Java-program og må finne ut hva som skjer.

Løsningsmåte 1: Forstå programmet

Les programkoden og forstå meningen.

```
class ArrayFyll {
  public static void main (String[] args) {
    int[] tall = new int[8];
    tall[0] = 0;
    tall[1] = 1;
    for (int i=2; i < 8; i++) {
        tall[i] = tall[i-2] + tall[i-1];
    }
    for (int t: tall) {
      System.out.println(t);
    }
  }
}
```

Oppgavetype 1: *Hva skjer?*

Du får se et kort Java-program og må finne ut hva som skjer.

Løsningsmåte 1: Forstå programmet

Les programkoden og forstå meningen.

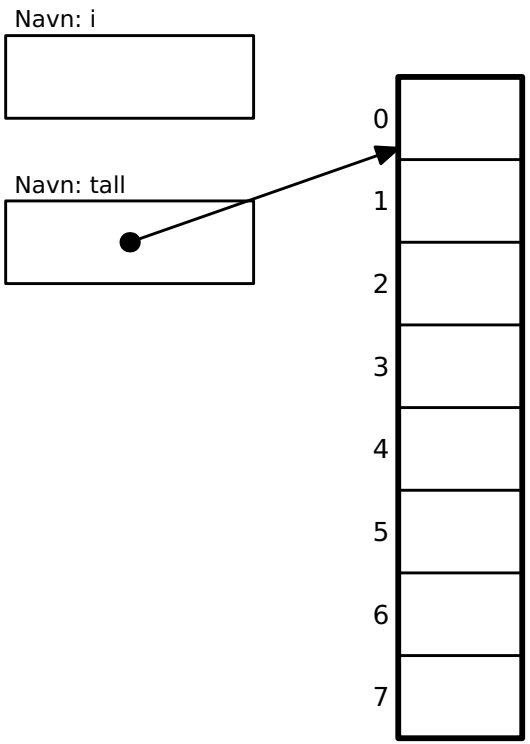
NB!

Les hele koden *nøye* selv om du tror du har skjønt hovedtanken. Det kan være fallgruber underveis.

```
class ArrayFyll {
  public static void main (String[] args) {
    int[] tall = new int[8];
    tall[0] = 0;
    tall[1] = 1;
    for (int i=2; i < 8; i++) {
        tall[i] = tall[i-2] + tall[i-1];
    }
    for (int t: tall) {
      System.out.println(t);
    }
  }
}
```

Løsningsmåte 2: Utfør programmet

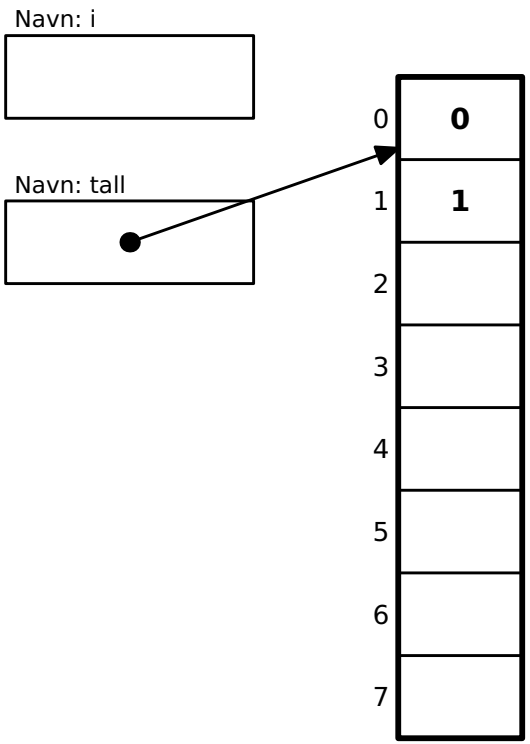
«Lek» datamaskin og gjør det programkoden sier.




```java
class ArrayFyll {
  public static void main (String[] args) {
    int[] tall = new int[8];
    tall[0] = 0;
    tall[1] = 1;
    for (int i=2; i < 8; i++) {
        tall[i] = tall[i-2] + tall[i-1];
    }
    for (int t: tall) {
      System.out.println(t);
    }
  }
}
```

Løsningsmåte 2: Utfør programmet

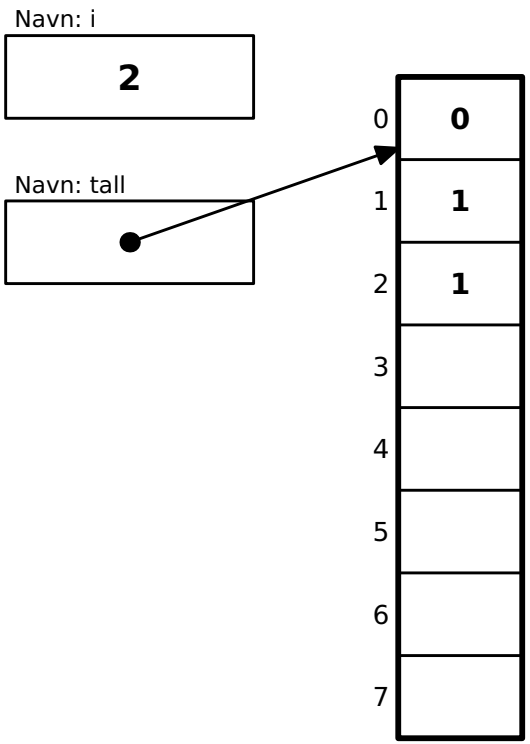
«Lek» datamaskin og gjør det programkoden sier.




```
class ArrayFyll {
  public static void main (String[] args) {
    int[] tall = new int[8];
    tall[0] = 0;
    tall[1] = 1;
    for (int i=2; i < 8; i++) {
        tall[i] = tall[i-2] + tall[i-1];
    }
    for (int t: tall) {
      System.out.println(t);
    }
  }
}
```

Løsningsmåte 2: Utfør programmet

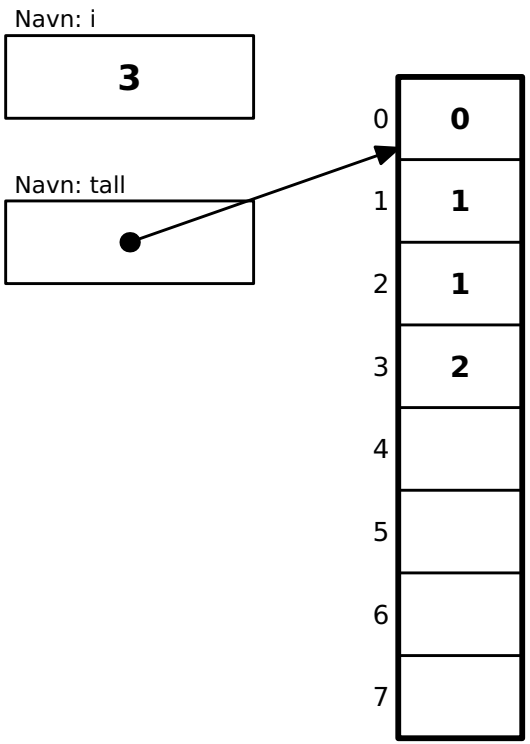
«Lek» datamaskin og gjør det programkoden sier.




```java
class ArrayFyll {
  public static void main (String[] args) {
    int[] tall = new int[8];
    tall[0] = 0;
    tall[1] = 1;
    for (int i=2; i < 8; i++) {
        tall[i] = tall[i-2] + tall[i-1];
    }
    for (int t: tall) {
      System.out.println(t);
    }
  }
}
```

Løsningsmåte 2: Utfør programmet

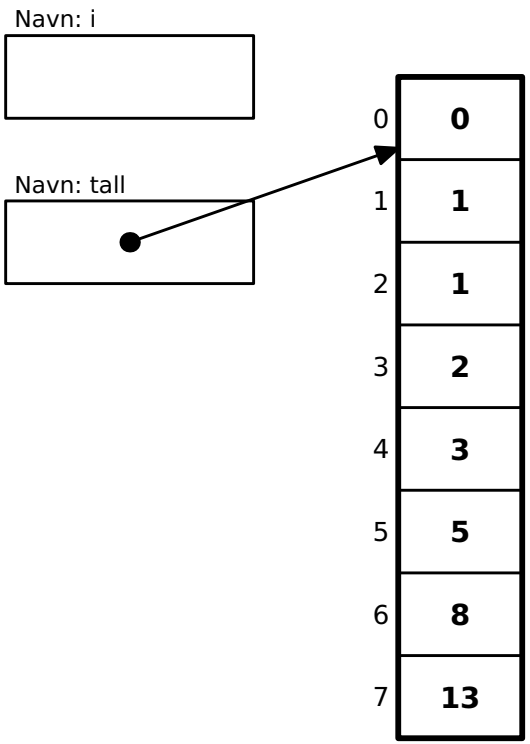
«Lek» datamaskin og gjør det programkoden sier.




```java
class ArrayFyll {
  public static void main (String[] args) {
    int[] tall = new int[8];
    tall[0] = 0;
    tall[1] = 1;
    for (int i=2; i < 8; i++) {
        tall[i] = tall[i-2] + tall[i-1];
    }
    for (int t: tall) {
      System.out.println(t);
    }
  }
}
```

Løsningsmåte 2: Utfør programmet

«Lek» datamaskin og gjør det programkoden sier.




```
class ArrayFyll {
  public static void main (String[] args) {
    int[] tall = new int[8];
    tall[0] = 0;
    tall[1] = 1;
    for (int i=2; i < 8; i++) {
        tall[i] = tall[i-2] + tall[i-1];
    }
    for (int t: tall) {
      System.out.println(t);
    }
  }
}
```

Oppgavetype 2: *«Programmering»*

Du får presentert en programmeringsoppgave:

> Her er to metoder:
>
> **settInnIArray** setter verdien **v** inn i arrayen **a** i posisjonen med indeks **ix**, men bare hvis **ix** er en lovlig indeks.
>
> **hentFraArray** henter ut verdien lagret i arrayen **a** i posisjonen med indeks **ix**, men hvis **ix** er mindre enn minste lovlige indeks, skal metoden returnere arrayelementet med lavest lovlig indeks; tilsvarende, hvis **ix** er større enn største lovlige indeks, skal metoden returnere elementet med høyest lovlig indeks.

Oppgave 2

Så kommer en løsning der det mangler diverse biter:

```
public static void                    ( int [ ] a, int ix, int v ) {
    if (                              ) {
                                      ;
    }
}

public static int                     (int [ ] a, int ix) {
    if (ix < 0) {
        return a [                    ];
    } else if (ix >=                  ) {
        return                        ;
    } else {
        return                        ;
    }
}
```

Til sist kommer bitene som skal settes inn:

a.legalIndex(ix)	hentFraArray	a[a.length]	a.length	a[ix] = v
a[a.length-1]	0 <= ix < a.length	a[v] = ix	1	a[ix]++
ix >= 0 && ix < a.length	0	ix >= 0 || ix < a.length	settInnIArray	a[ix]
a.length()				

Merk!

- Noen ganger skal man bruke alle bitene, andre ganger bare noen.
- Noen ganger må man bruke samme bit flere steder i koden.

Hint til å løse oppgaven

- Noen biter er ganske innlysende – sett disse på plass først.
- Manglende navn på metoder kan man ofte skjønne utifra parametrene og returtypen (f eks at `hentFraArray` må ha en returtype mens `settInnIArray` kan være **`void`**).
- Husk at variabler må deklareres før de kan brukes.
- Så er det å sette inn kodebiter så programmet blir
  1. lovlig og
  2. gjør det det skal.

  Det vil bare være én riktig løsning.

Oppgavetype 3: Teori

Her får du diverse spørsmål du må besvare utifra det du har lært om objektorientering.

(På slike oppgaver kan du få minuspoeng om du svarer galt, men aldri mindre enn 0 poeng totalt.)

	Arv	Komposisjon
Person – student	◯	◯
Bibliotek – bok	◯	◯
Bil – hjul	◯	◯
Ulv – Rovdyr	◯	◯

Ett typisk eksempel er spørsmål om hvordan man bør strukturere ulike objekter i et program. Skal vi bruke *arv* eller *komposisjon*?

Løsningsmåte

Hvis det føles naturlig å si

$$a \text{ er en } b$$

bør man bruke arv (dvs a **`extends`** b).


En student er en person


~~En bok er et bibliotek~~

Hvis det derimot føles naturlig å si

a har/inneholder en b

bør man bruke komposisjon (dvs at a inneholder en instansvariabel (f eks en **`array`**) som refererer til b-ene).



~~En student har eller inneholder en person~~



Et bibliotek har eller inneholder en bok (eller flere)

Oppgavetype 4: *Er dette lov?*

I slike oppgaver får du se noe Java-kode:

```
public class A {
    public static void main (String[] args) {
        /* kodelinje */
    }
}
```

som skal kompileres og kjøres slik:

```
javac A.java
java A hallo
```

Spørsmålet er da om gitte kodelinjer er lov å bruke på angitt sted eller ikke.

Kompileringsfeil

er feil som kompilatoren (`javac`) kan oppdage:

- syntaksfeil (f eks manglende parentes)
- navnefeil (ukjent navn fordi det er skrevet galt)
- typefeil (f eks «`int x = true;`»)

Kjørefeil

er feil som først kan oppdages under kjøringen (`java`):

- ulovlig indeks i array
- følge en **`null`**-peker («`p.f()`» når `p` er **`null`**)
- ulovlig typekonvertering («`(Lastebil)bilPeker`»)
- regnefeil (som divisjon med 0)
- unntak (f eks fordi en fil ikke lar seg lese)

Oppgaven er altså å velge:

- Er linjen OK å bruke?
- Hvis ikke, kan `javac` oppdage feilen?
- Ellers er dette en feil som `java` oppdager under kjøring.

Eksempler

`return;`	OK
`return 42;`	kompileringsfeil (siden `main` er **`void`**)
`args[5];`	kjørefeil (siden `args` bare har ett element, nemlig `"hallo"`)

Refransetilordninger

Spesielt viktig er å kunne vurdere tilordninger av objektreferanser. Er dette lovlig?

```
Bil bilPeker = new Lastebil();
Lastebil minLastebil = bilPeker;
```

Hvordan avgjør man slikt?

Refransetilordninger

Spesielt viktig er å kunne vurdere tilordninger av objektreferanser. Er dette lovlig?

```
Bil bilPeker = new Lastebil();
Lastebil minLastebil = bilPeker;
```

Hvordan avgjør man slikt?

Hovedregel

Ved tilordning må venstresiden ha samme type som høyresiden eller være av en mer generell type (dvs en type som høyresiden **`extends`**-er eller **`implements`**-er).

Løsningsmåte

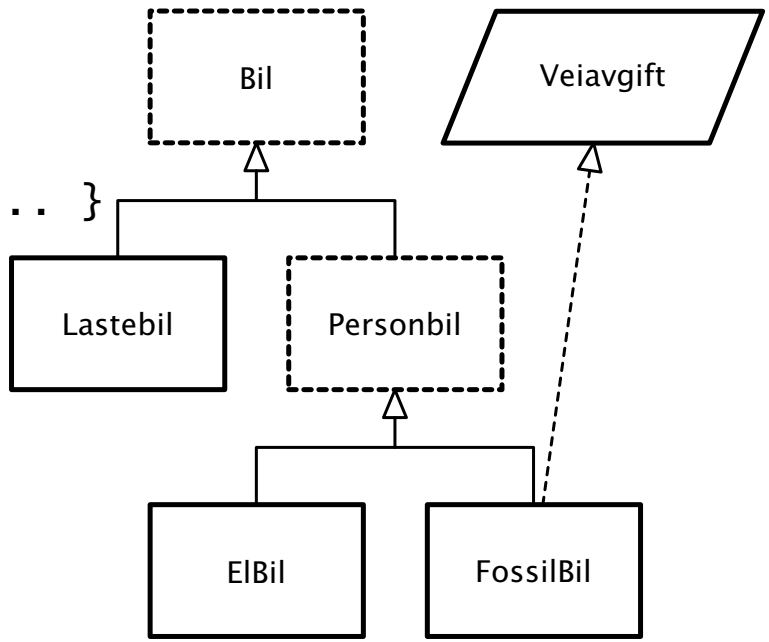
Anta at vi har

```
abstract class Bil { ... }
class Lastebil extends Bil { ... }
abstract class Personbil extends Bil { ... }
class ElBil extends Personbil { ... }
class FossilBil extends Personbil
    implements Veiavgift { ... }
interface Veiavgift { ... }
```

Løsningsmåte

Anta at vi har

```
abstract class Bil { ... }
class Lastebil extends Bil { ... }
abstract class Personbil extends Bil { ... }
class ElBil extends Personbil { ... }
class FossilBil extends Personbil
    implements Veiavgift { ... }
interface Veiavgift { ... }
```



Det første vi bør gjøre er å tegne klassestrukturdiagrammet:

Så må vi finne typene til venstre- og høyresiden.

Venstresiden

Dette vil være en variabel. Se på variabelens *deklarasjon*.

Så må vi finne typene til venstre- og høyresiden.

Venstresiden

Dette vil være en variabel. Se på variabelens *deklarasjon*.

Høyresiden

Her finnes tre muligheter:

`new Klasse()` har typen **`Klasse`**.

Så må vi finne typene til venstre- og høyresiden.

Venstresiden

Dette vil være en variabel. Se på variabelens *deklarasjon*.

Høyresiden

Her finnes tre muligheter:

`new Klasse()` har typen **`Klasse`**.

`variabel` Se på variabelens *deklarasjon*.

Så må vi finne typene til venstre- og høyresiden.

Venstresiden

Dette vil være en variabel. Se på variabelens *deklarasjon*.

Høyresiden

Her finnes tre muligheter:

`new Klasse()` har typen **`Klasse`**.

`variabel` Se på variabelens *deklarasjon*.

`(Klasse)variabel` (typekonvertering, «casting») gir en ny sjekk:

Så må vi finne typene til venstre- og høyresiden.

Venstresiden

Dette vil være en variabel. Se på variabelens *deklarasjon*.

Høyresiden

Her finnes tre muligheter:

`new Klasse()` har typen **`Klasse`**.

`variabel` Se på variabelens *deklarasjon*.

`(Klasse)variabel` (typekonvertering, «casting») gir en ny sjekk:

1. Se på selve *objektet* variabelen peker på.

Så må vi finne typene til venstre- og høyresiden.

Venstresiden

Dette vil være en variabel. Se på variabelens *deklarasjon*.

Høyresiden

Her finnes tre muligheter:

`new Klasse()` har typen **`Klasse`**.

`variabel` Se på variabelens *deklarasjon*.

`(Klasse)variabel` (typekonvertering, «casting») gir en ny sjekk:

1. Se på selve *objektet* variabelen peker på.
2. Kan det tilordnes til typen **`Klasse`**? Hvis ikke, er det en kjørefeil.

Så må vi finne typene til venstre- og høyresiden.

Venstresiden

Dette vil være en variabel. Se på variabelens *deklarasjon*.

Høyresiden

Her finnes tre muligheter:

`new Klasse()` har typen **`Klasse`**.

`variabel` Se på variabelens *deklarasjon*.

`(Klasse)variabel` (typekonvertering, «casting») gir en ny sjekk:

1. Se på selve *objektet* variabelen peker på.
2. Kan det tilordnes til typen **`Klasse`**? Hvis ikke, er det en kjørefeil.
3. Resultatet har typen **`Klasse`**.

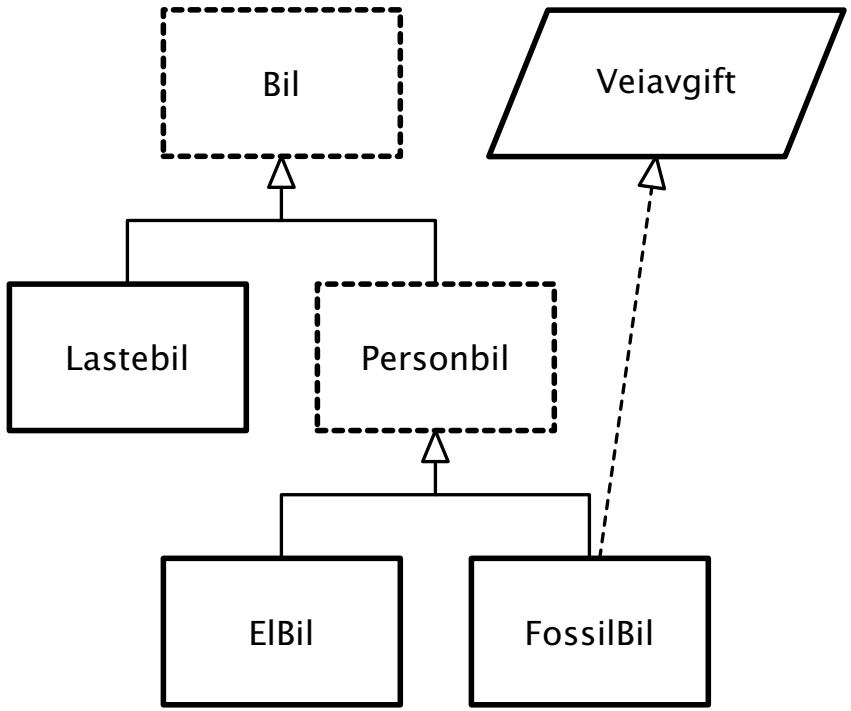
Hovedregel (i ny form)

En tilordning er lovlig når man kan komme fra høyresidens type til venstresidens type ved å følge piler i klassestrukturdiagrammet.

Eksempler

```
Bil bil1 = new Lastebil();
Bil bil2 = new ElBil();
Lastebil bil3 = new Lastebil();

Personbil bil4;
bil4 = bil3;  // Feil
bil4 = bil2;  // Feil
bil4 = (Personbil)bil1;  // Kjørefeil
bil4 = (Personbil)bil2;  // OK
```



Oppgavetype 5: *Pekersjonglering*

Du får se et Java-program som oppretter noen objekter og lar diverse pekere peke på dem. Hva er status til slutt?

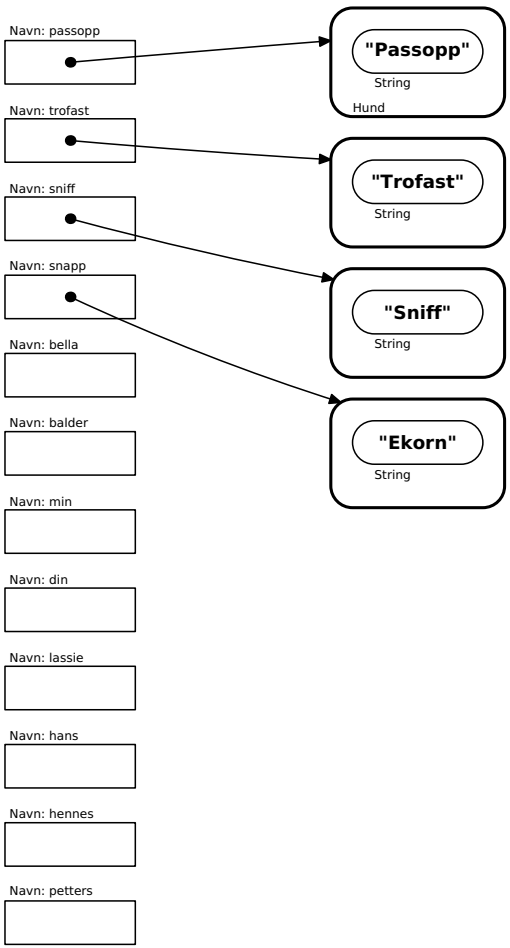
```
class Hunder {
    public static void main (String[] a) {
        Hund passopp = new Hund("Passopp");
        Hund trofast = new Hund("Trofast");
        Hund sniff = new Hund("Sniff");
        Hund snapp = new Hund("Ekorn");

        Hund bella, balder, min, din, lassie, hans, hennes, petters;

        min = snapp;
        din = passopp;
        petters = din;
        hans = min;
        din = snapp;
        hennes = trofast;
        trofast = sniff;
        lassie = hennes;
        petters = din;
        bella = lassie;
```

Hvilket objekt peker alle pekerne på?

Her vil jeg anbefale å tegne objektene og pekerne:



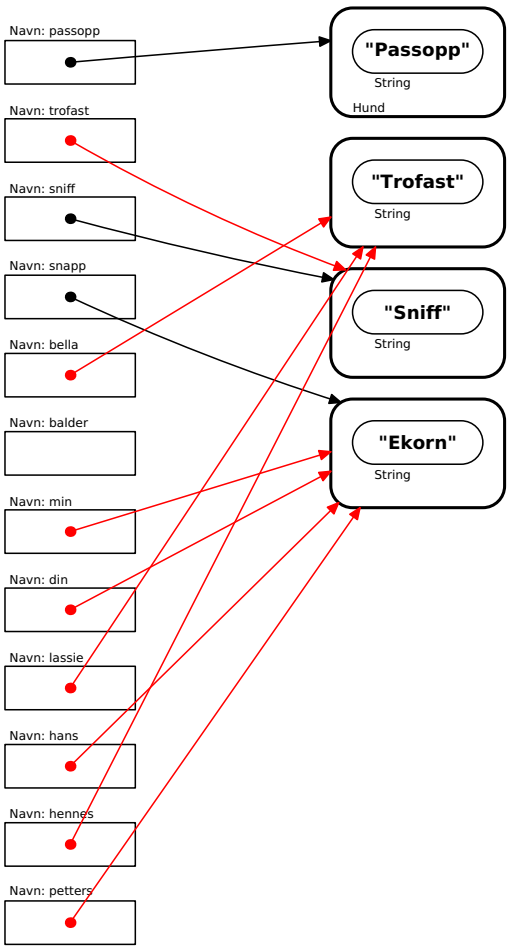

```
class Hunder {
    public static void main (String[] a) {
        Hund passopp = new Hund("Passopp");
        Hund trofast = new Hund("Trofast");
        Hund sniff = new Hund("Sniff");
        Hund snapp = new Hund("Ekorn");

        Hund bella, balder, min, din, lassie, hans, hennes, petters;

        min = snapp;
        din = passopp;
        petters = din;
        hans = min;
        din = snapp;
        hennes = trofast;
        trofast = sniff;
        lassie = hennes;
        petters = din;
        bella = lassie;
```

Hvilket objekt peker alle pekerne på?

Her vil jeg anbefale å tegne objektene og pekerne:




```
class Hunder {
    public static void main (String[] a) {
        Hund passopp = new Hund("Passopp");
        Hund trofast = new Hund("Trofast");
        Hund sniff = new Hund("Sniff");
        Hund snapp = new Hund("Ekorn");

        Hund bella, balder, min, din, lassie, hans, hennes, petters;

        min = snapp;
        din = passopp;
        petters = din;
        hans = min;
        din = snapp;
        hennes = trofast;
        trofast = sniff;
        lassie = hennes;
        petters = din;
        bella = lassie;
```

Polymorfi

Her vil du få se noen klasser og subklasser og skal avgjøre hvilken (re)definisjon som velges.

```
class Bok { double pris;
    double internpris () { return pris; }
}

class Laerebok extends Bok {
    double internpris () { return pris*0.8; }
}

class IN1010bok extends Laerebok {
    double internpris () { return super.internpris()*0.5; }
}

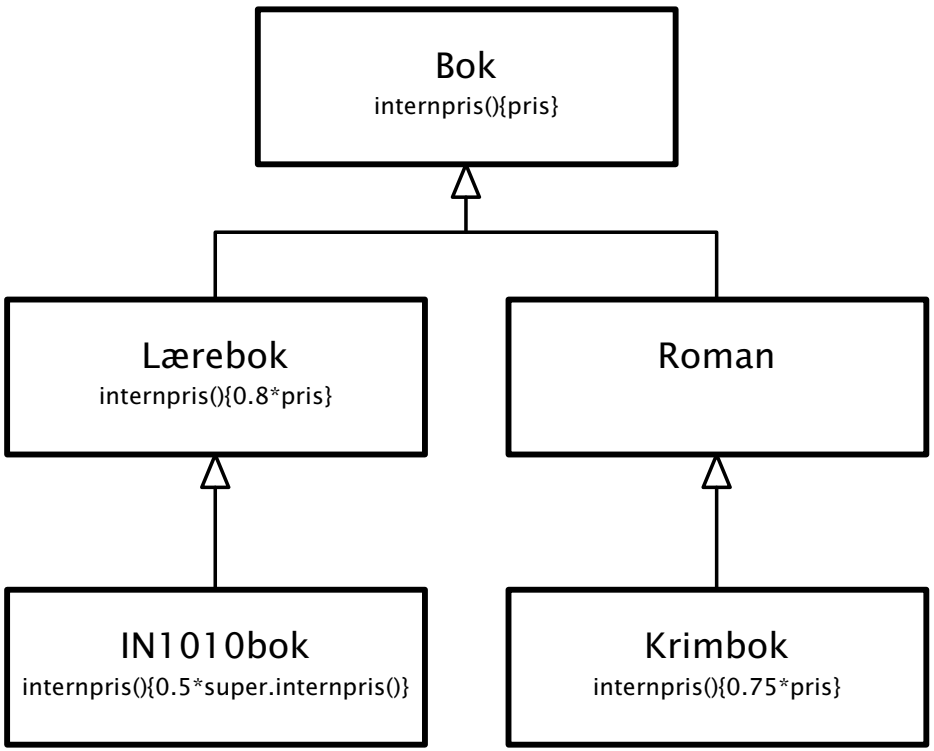
class Roman extends Bok {
}

class Krimbok extends Roman {
    double internpris () { return pris*0.75; }
}
```

Husk!

Når vi skal velge blant polymorfe metoder, må vi se på hvilket *objekt* som er opprettet.

Ofte er det en god hjelp å tegne klassediagrammet og legge inn litt ekstra informasjon.

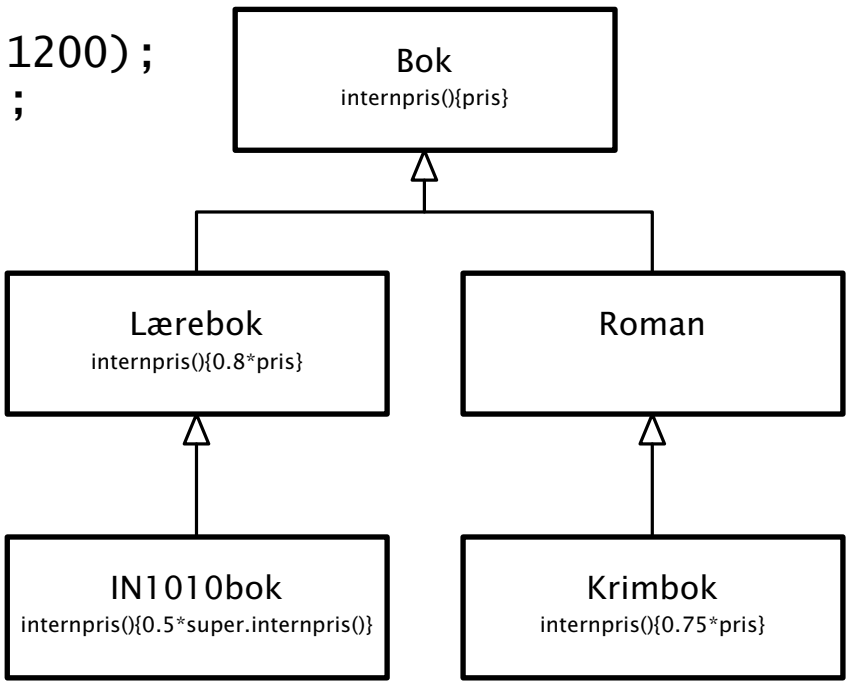


Eksempler

Anta at vi har

```
Bok bok3 = new IN1010bok("Big Java", 1200);
System.out.println(bok3.internpris());
```

Vi ser at bok3 er opprettet som en IN1010bok. (At variabelen bok3 er deklarert som en Bok er uvesentlig i denne sammenhengen.)



Svaret blir derfor
0.5 * `Lærebok.internpris()` =
0.5 * 0.8 * `pris` =
0.4 * 1200 = 480