Pekare (till objekt)

Objektvariabler är *pekare* – vad är det?

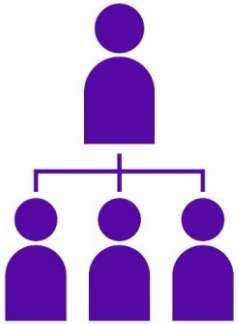
(Viktiga begrepp – inte bara inom objektorientering!)

Hur används pekare för att *sätta samman* objekt (composition)?

Variabler och pekare

Motivation

- **Programmeringsuppgift**:
  - Lagra info om *anställda*
  - Håll reda på varje anställds *närmaste chef*
    - …som också är en anställd!



- En första ansats:

```
public class Employee {
   private String name;  // Och andra egenskaper
   private Employee boss;

   public Employee(String name, Employee boss) {
     this.name = name;
     this.boss = boss;
   }
}
```

Motivation (2): Innehåll?

jonkv@ida

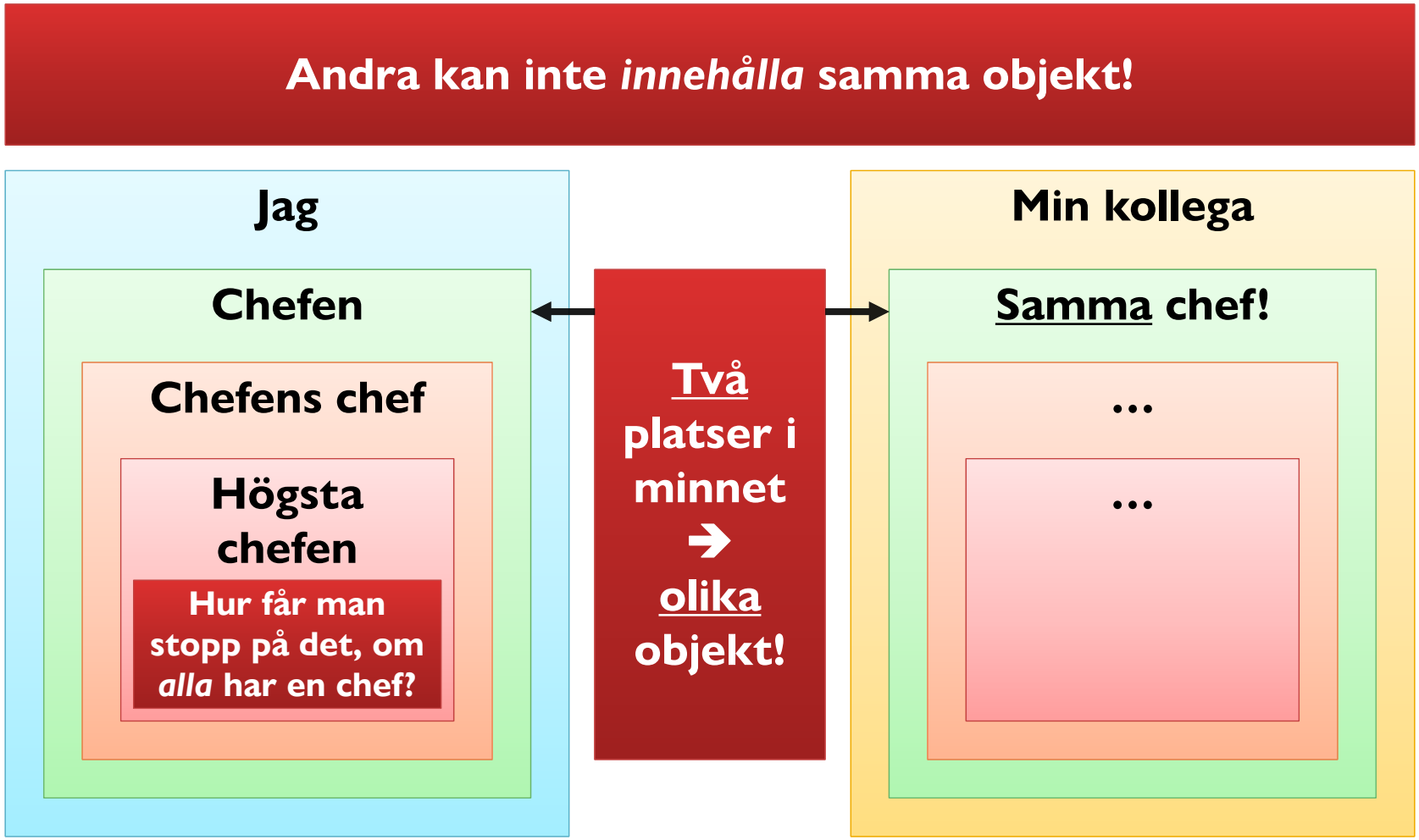
- **Ser ut** som att en **Employee** *innehåller* en annan…


```
public class Employee {
   private String name;
   private Employee boss;

   public Employee(String name, Employee boss) {
     this.name = name;
     this.boss = boss;
   }
}
```

Motivation (3): Innehåll?

jonkv@ida

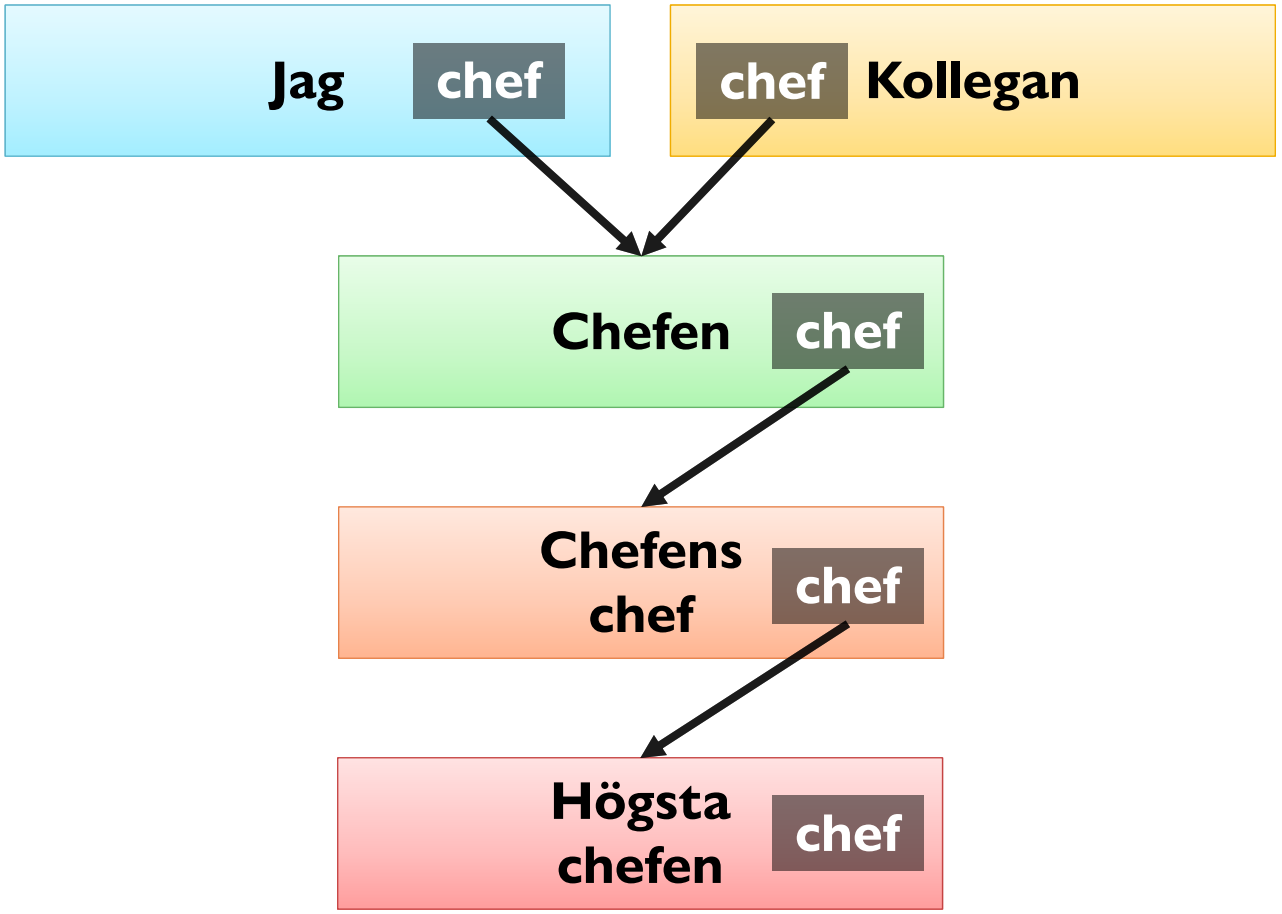
- Kan bli problematiskt!



Motivation (4): En bättre lösning

jonkv@ida

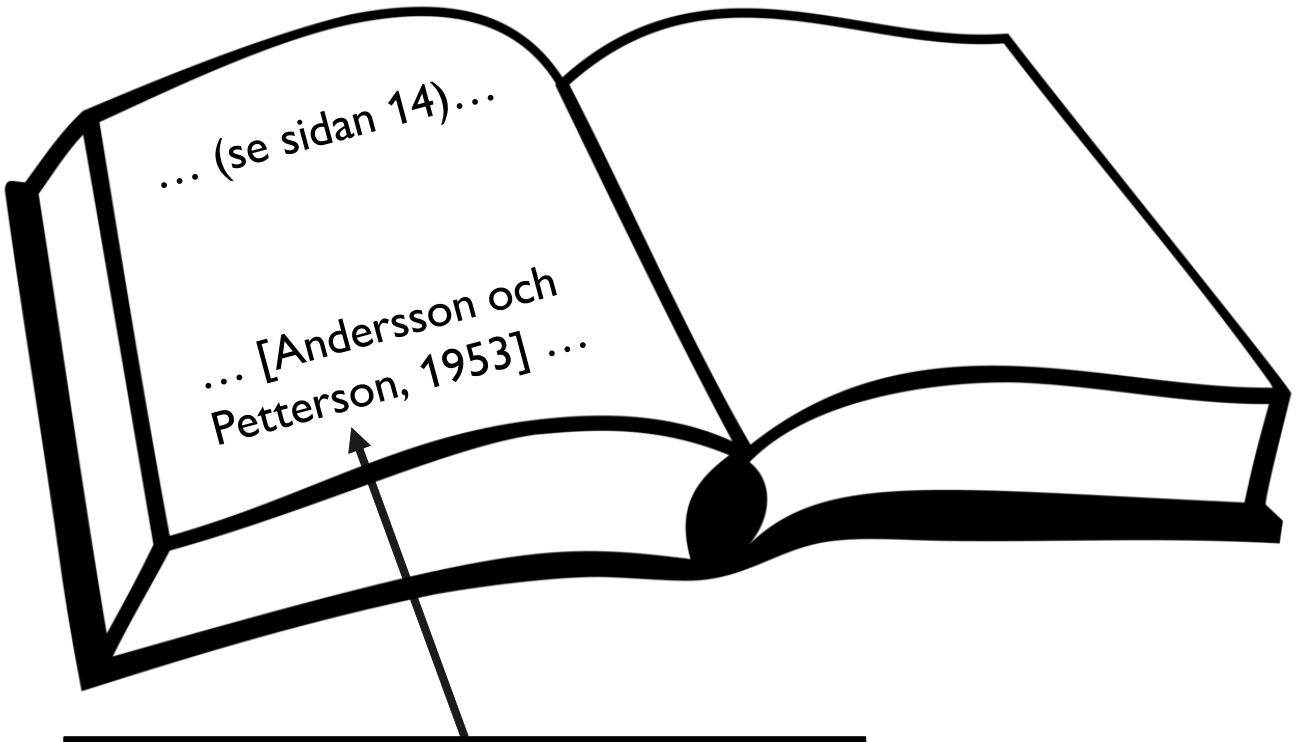
- En anställd ska inte *innehålla* chefen, utan *peka ut* vem chefen är



Fem objekt, alla på sin egen unika plats i minnet

Så: Hur *pekar* man på ett *annat* objekt?

Pekare i verkligheten



Istället för att kopiera in artikeln vi pratar om, ger vi en referens till den så att den kan hittas där

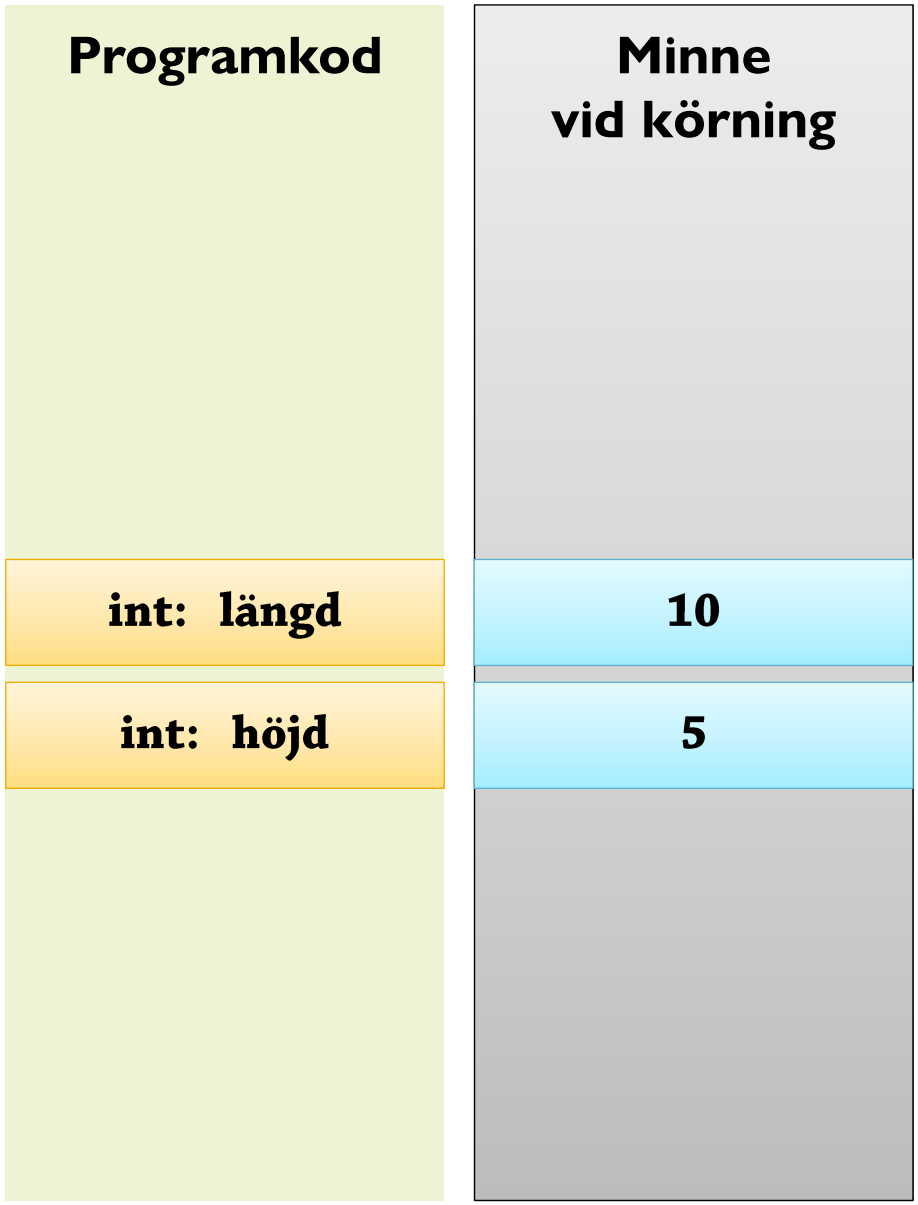
Pekare: För att "slå upp" ett objekt

- Pekare har **likheter** med många andra sätt att **slå upp** data
  - Exempel: **dict** i Python
    - people = {
      "x472": { **"name"**: **"Jonas"**, **"bossID"**: **"z208"** },
      "z208": { **"name"**: **"Patrick"**, **"bossID"**: **"z33"** },
      …
      }
    - **me** = people[**"x472"**] *# Ett "**objekt**"*
      *# Värde: **{ "name": "Jonas", "bossID": "z208" }***
    - **my_boss_id** = me[**"bossID"**] *# En "**pekare**", identifierar ett annat "objekt"*
      *# Värde: **"z208"***
    - **my_boss** = people[my_boss_id] *# Det andra "**objektet**"*
      *# Värde: **{ "name": "Patrick", "bossID": "z33" }***

Men under detta behövs något mer grundläggande…

Variabler och minnesadresser

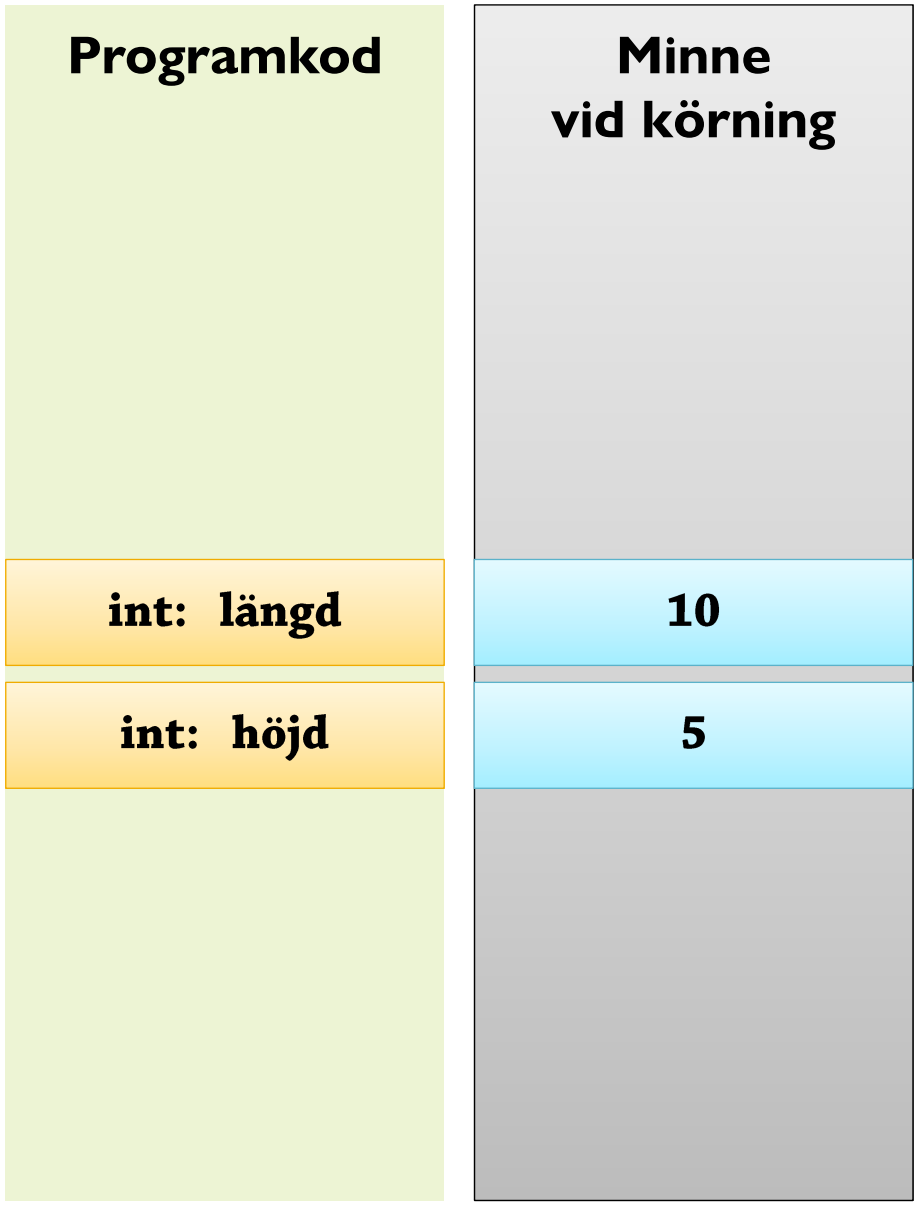
jonkv@ida



- Använd **minnesadresser**!
  - Repetition: De finns alltid där "i bakgrunden"…
    - **"int längd = 10;"** ➔ "lagra 32-bitars heltal *10* på adress *10000–10003*"

Variabler och minnesadresser (2)

jonkv@ida



- I vissa språk kan man **ta reda på** minnesadresserna
  - C:

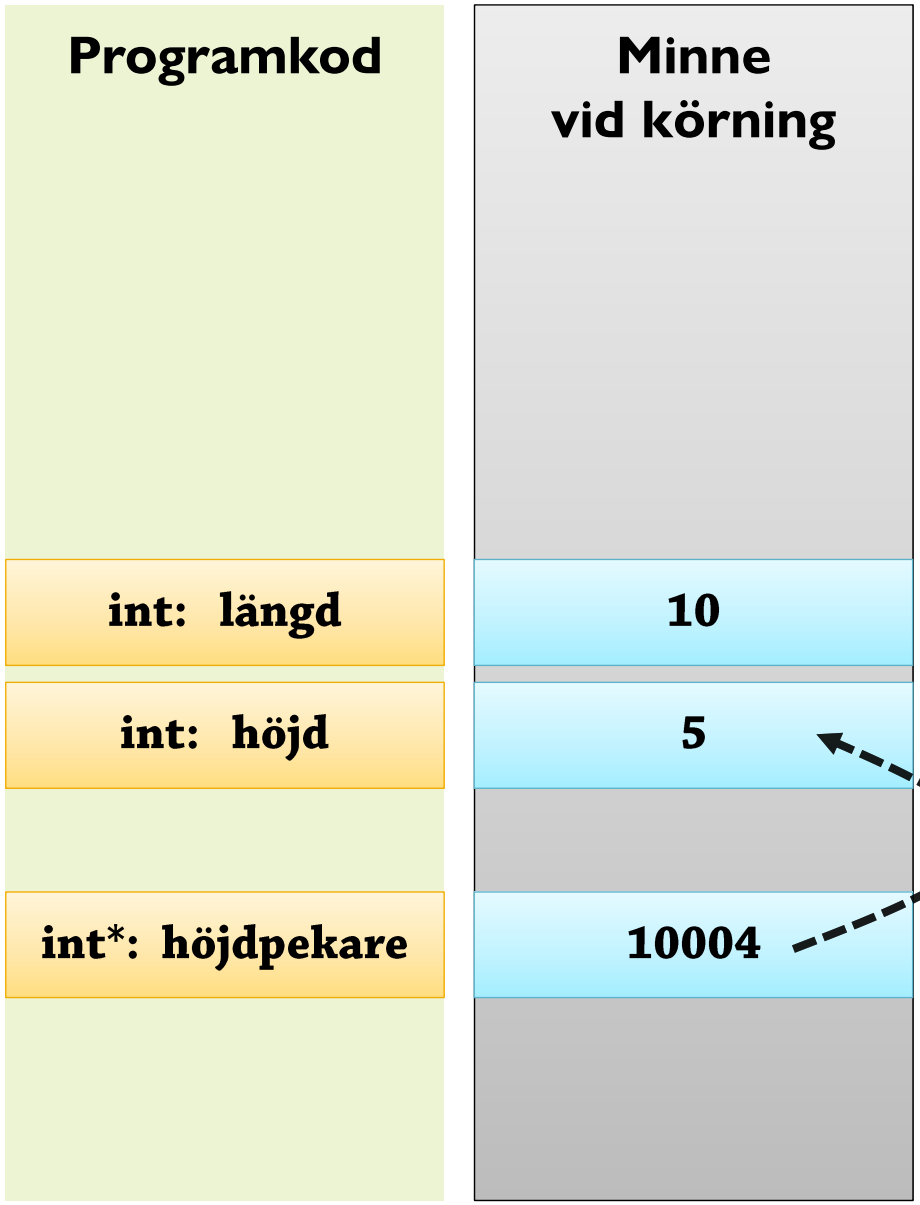
    höjd == 5

    &höjd == 10004

    ("adressen till höjd",
    "var lagras höjden?")

Pekare (1)

jonkv@ida

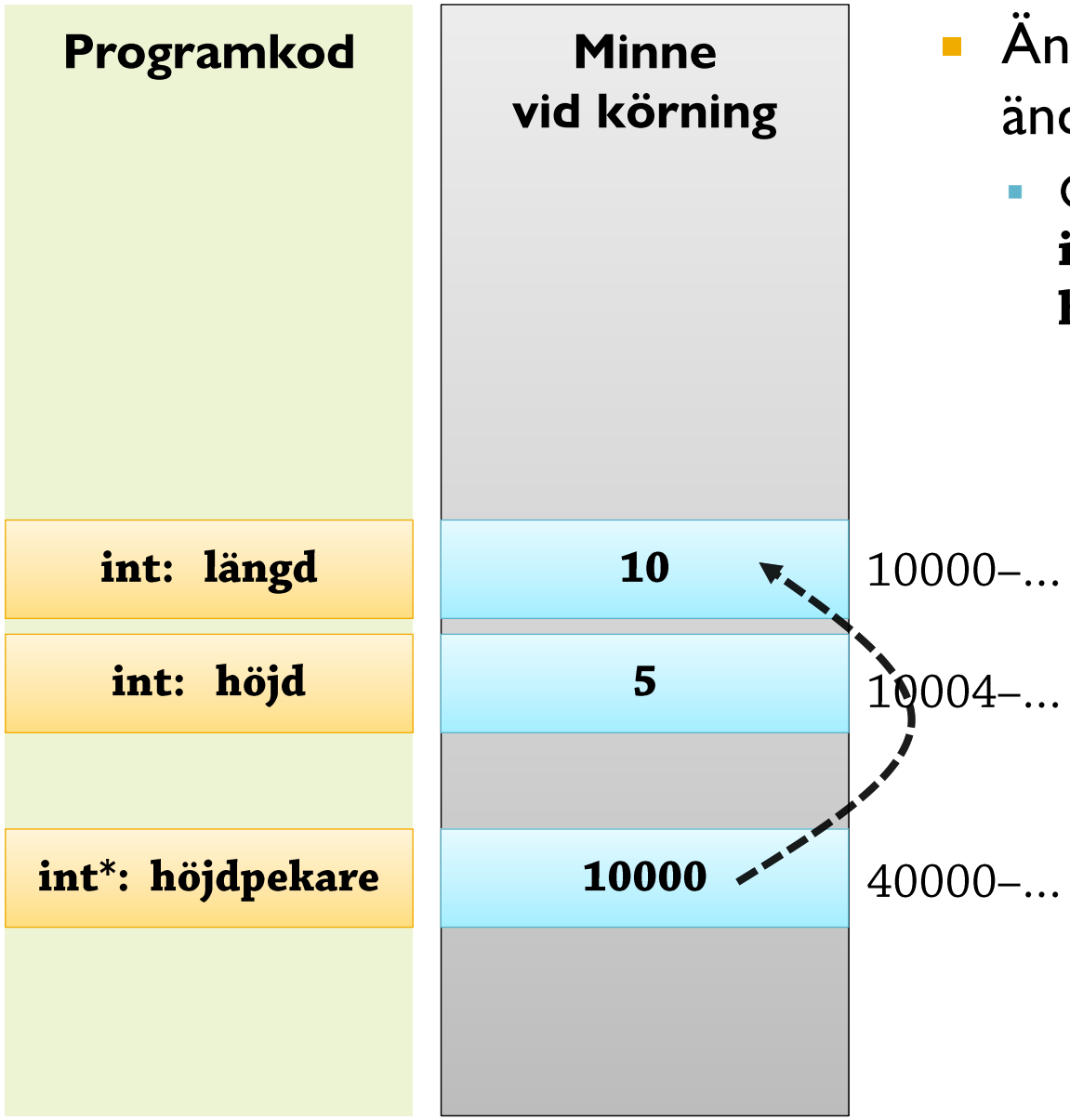


- En variabel av **pekartyp** kan *innehålla* en adress
  - C: "pekare till **int**" heter **int***
  - **int* höjdpekare = &höjd;**
  - Värdet blir adressen 10004

Värdet 10004 är *adressen* till ett *annat* värde i minnet

Pekare (2): Tilldelning

jonkv@ida



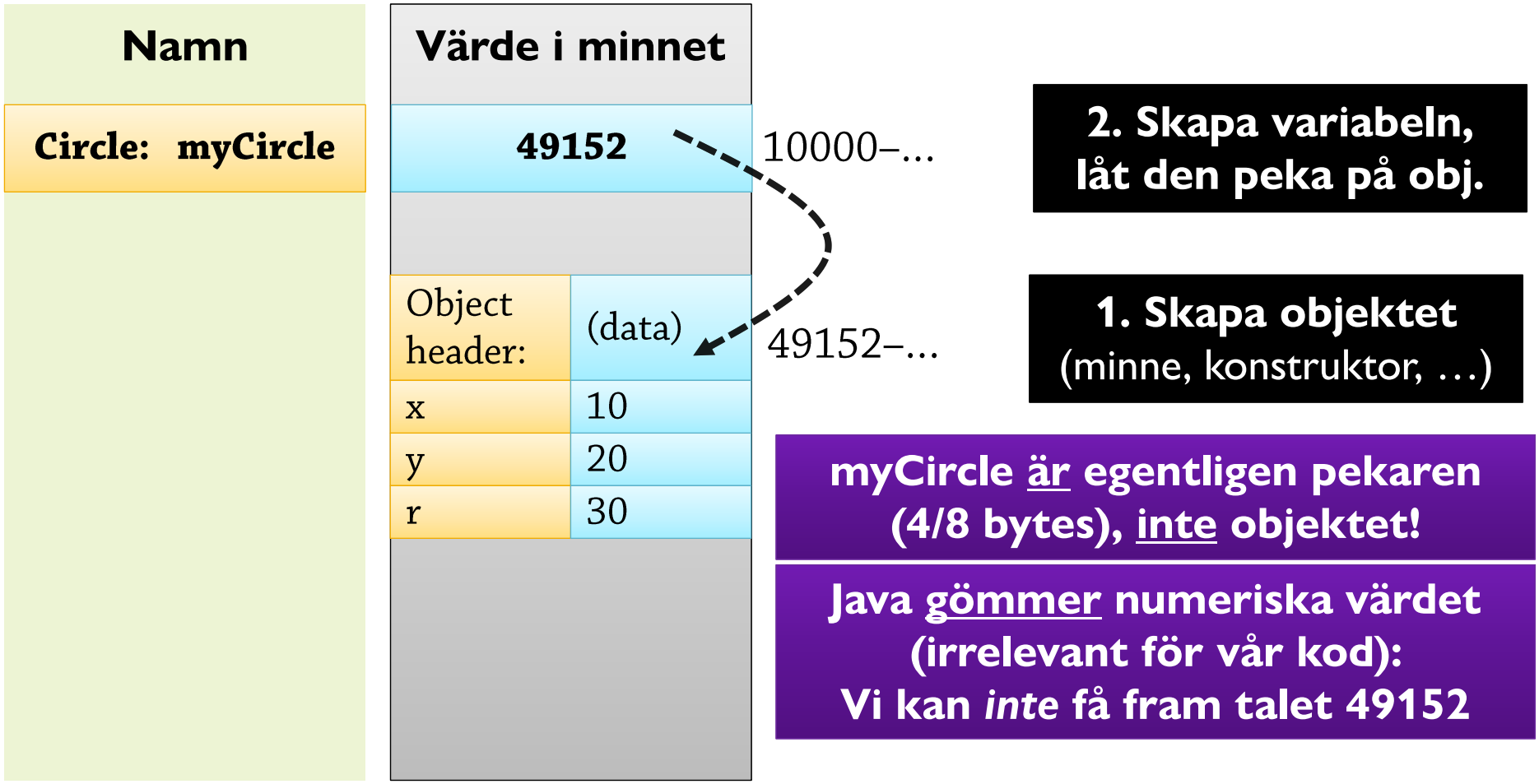
- Ändra **pekarens värde** ➔ ändra *vart den pekar*
  - C:
    int* höjdpekare = &höjd;
    höjdpekare = &längd;

Pekare i Java:
Enbart för objekt, alltid för objekt

Objektvariabler är pekare

jonkv@ida

- Java: En objektvariabel är **alltid** en pekare!
  - Circle **myCircle** = **new** Circle(10, 20, 30);



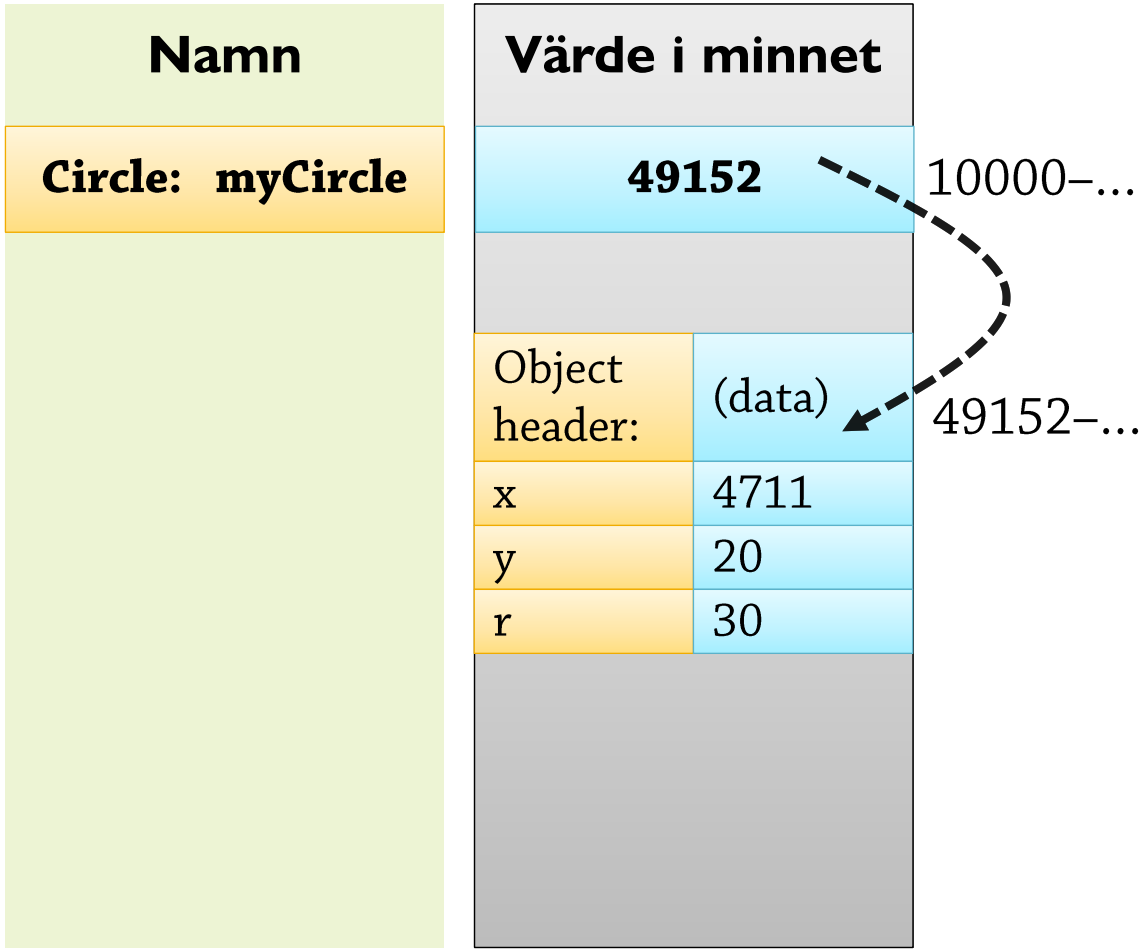
Detta gäller även Python!

(Och i Python är alla värden objekt,
så även heltalsvariabler är pekare)

Men vilka konsekvenser får detta?

Pekare i Java (1): Objektmedlemmar

- Att komma åt medlemmar: T.ex. **myCircle.x = 4711**;
  - Beräknar **myCircle** (blir **49152**);
  - "**.x**" *betyder* "följ pekaren, hitta fältet **x**" -- ändrar dess värde

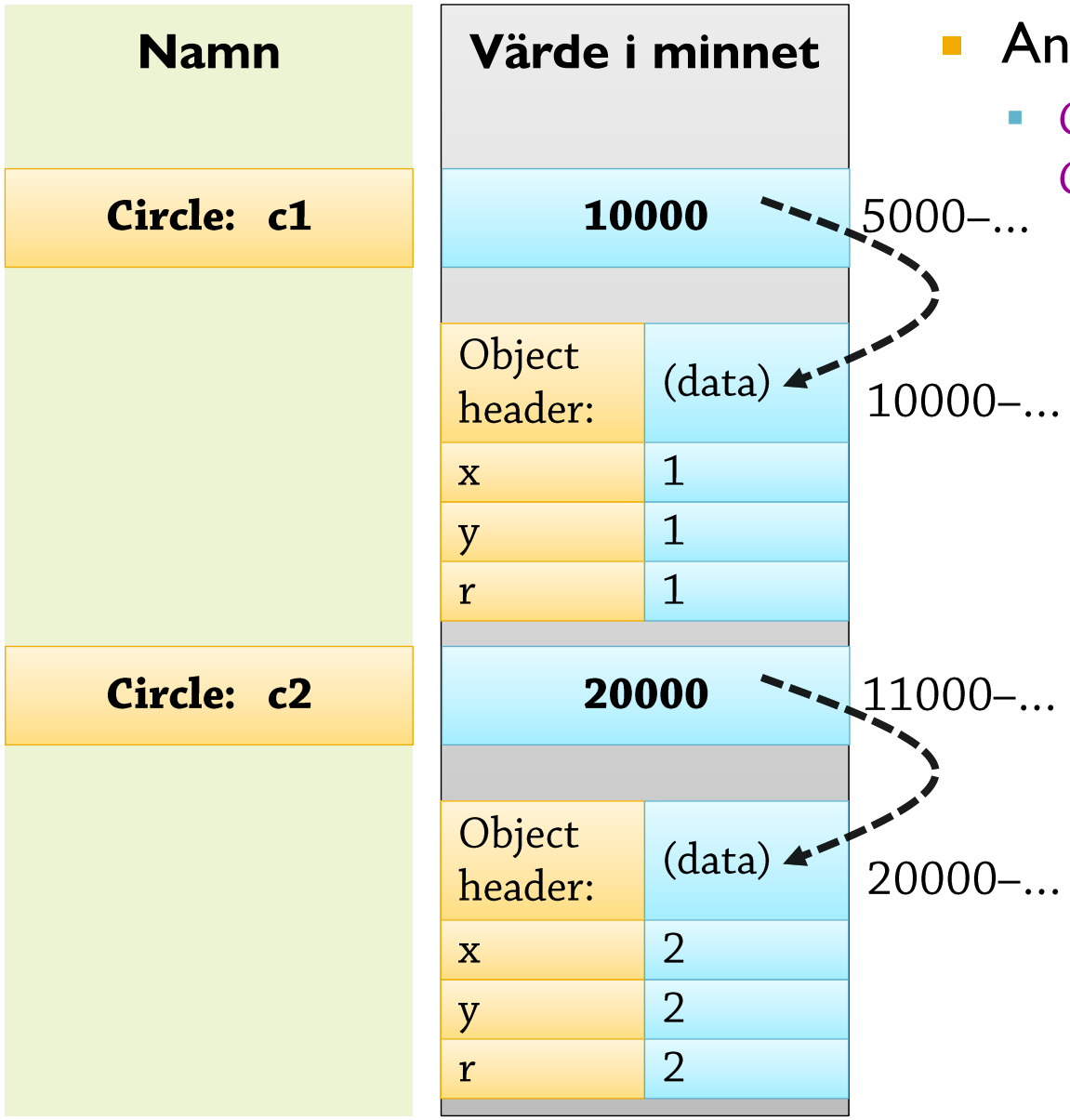


Även i Python:

```
# list1 blir pekare
list1 = [10, 20, 30]

# följ pekaren
list1[0] = 4711
```

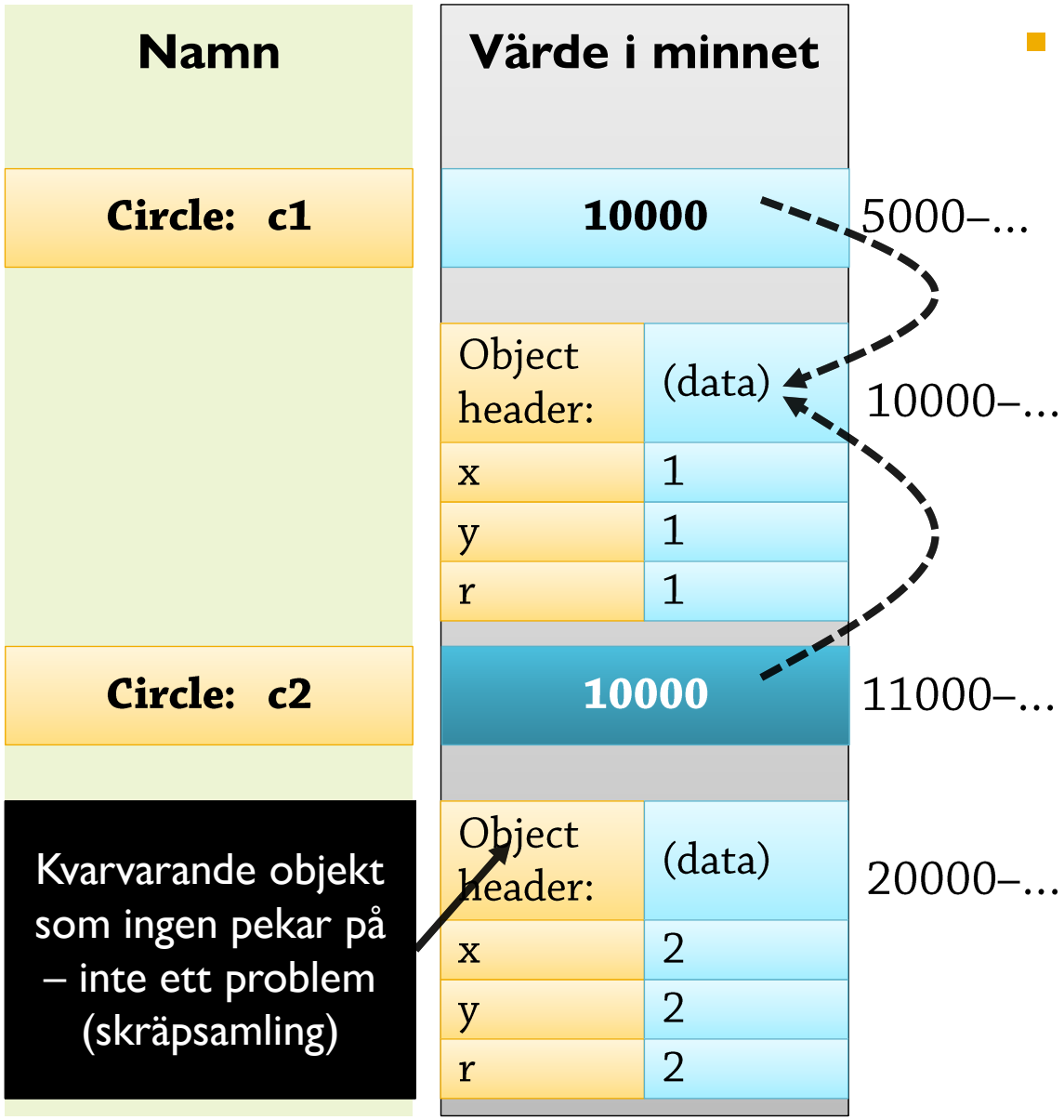
Pekare i Java (2): Exempel

jonkv@ida



- Anta två cirklar:
  - Circle c1 = **new** Circle(1,1,1);
    Circle c2 = **new** Circle(2,2,2);

Pekare i Java (3): Tilldelning

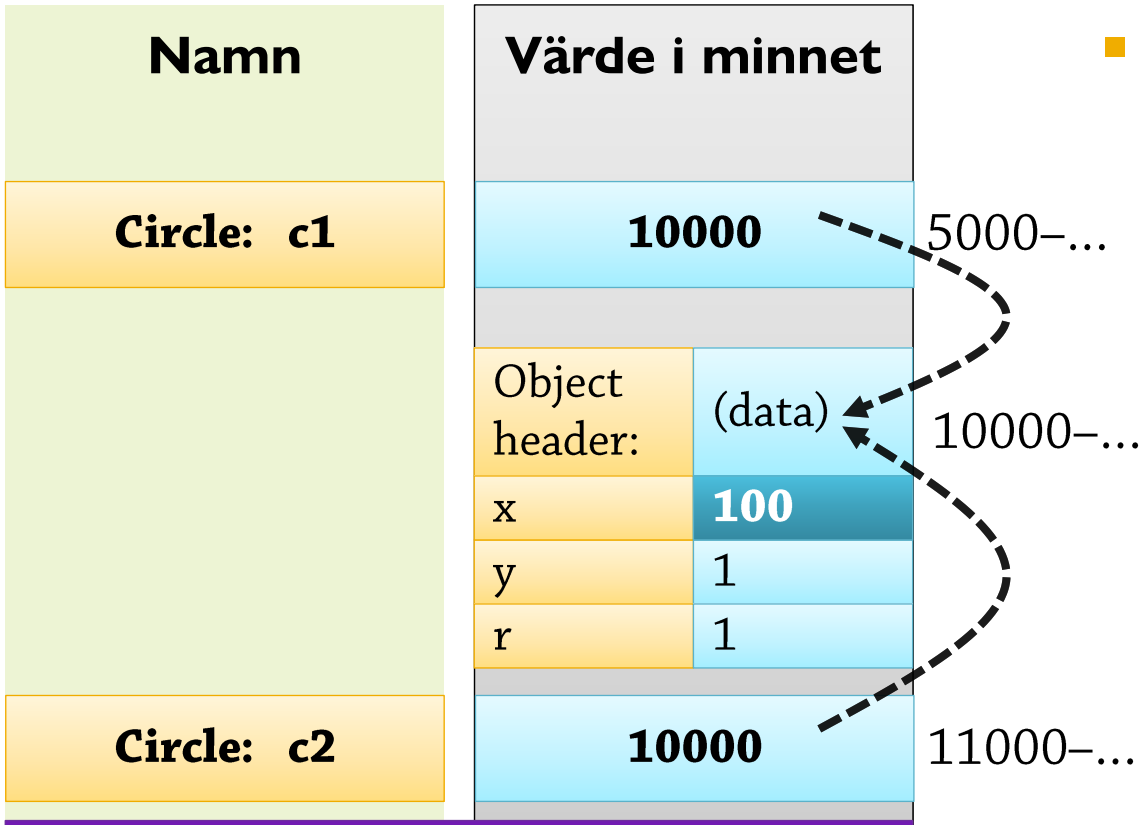


- **Tilldela** c2 = c1;
  - c1, c2 är pekare
  - Ändra **pekarens värde** → ändra *vart den pekar*
  - Sätter c2 till 10000
  - Kommer **inte** att kopiera själva cirkeln, fält för fält!

Även i Python:

```
list1 = [10, 20, 30]
list2 = list1

# Nu pekar båda variabler
# på samma listobjekt
```

Pekare i Java (4):



- c2.x = 100;
  - Nu är **c1.x** också 100!
  - **c1** och **c2** är två variabler, men pekar på *samma* objekt
  - **c1.x** och **c2.x** är *samma* variabel!

”*Jonas Kvarnström*" är en sträng
"*Kursens examinator*" är en annan
Men de *pekar ut* samma person

Om *Jonas Kvarnström* får ny chef,
får *kursens examinator* ny chef

Även i Python:

```
list1 = [10, 20, 30]
list2 = list1
list1[0] = 400
# Nu har vi list2[0] == 400
```

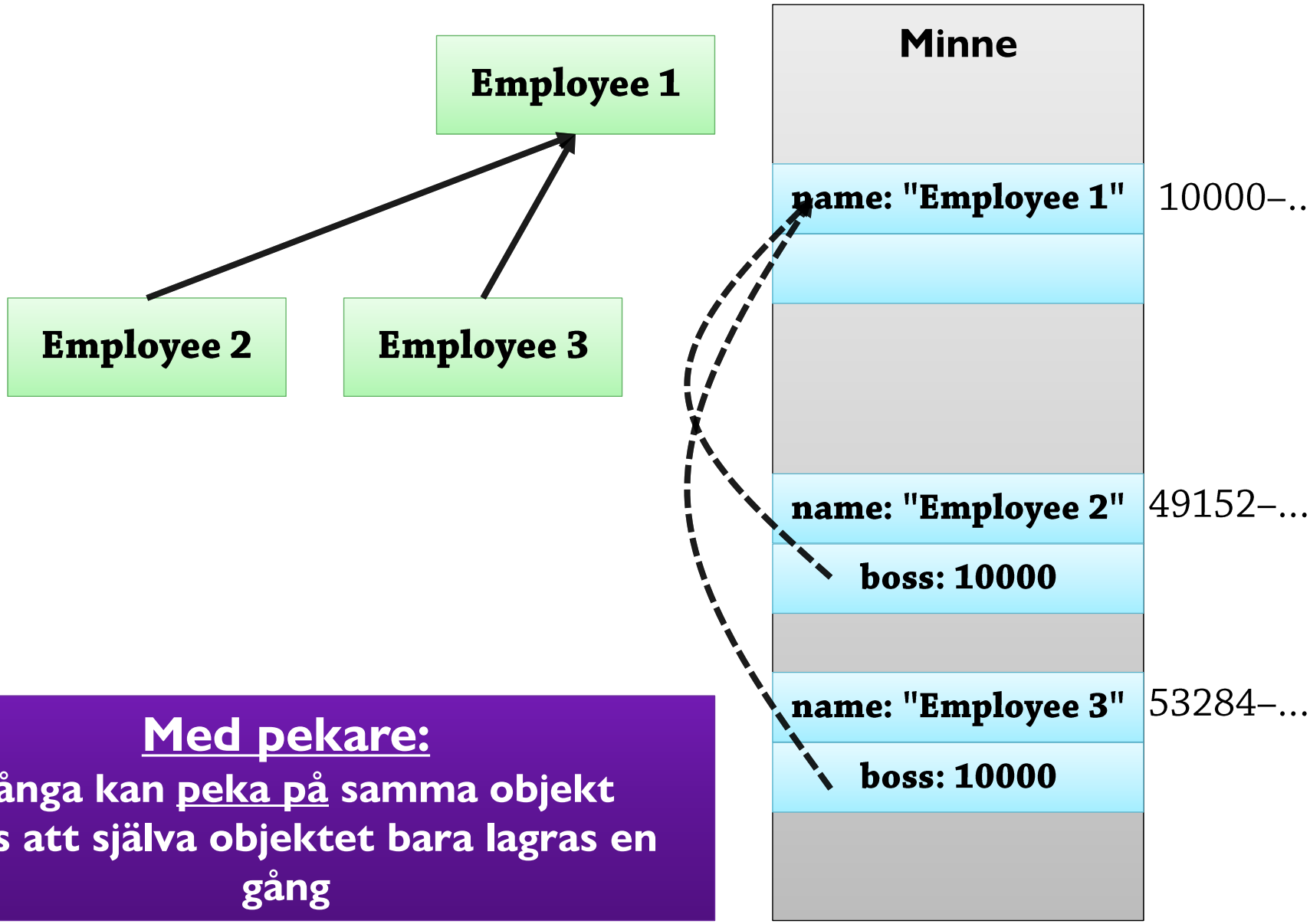
Pekarexemplet (0)

jonkv@ida

```
public class Employee {
   private String name;
   private Employee boss;

   public Employee(String name, Employee boss) {
      this.name = name;
      this.boss = boss;
   }
}
```

Pekarexemplet (1)

jonkv@ida

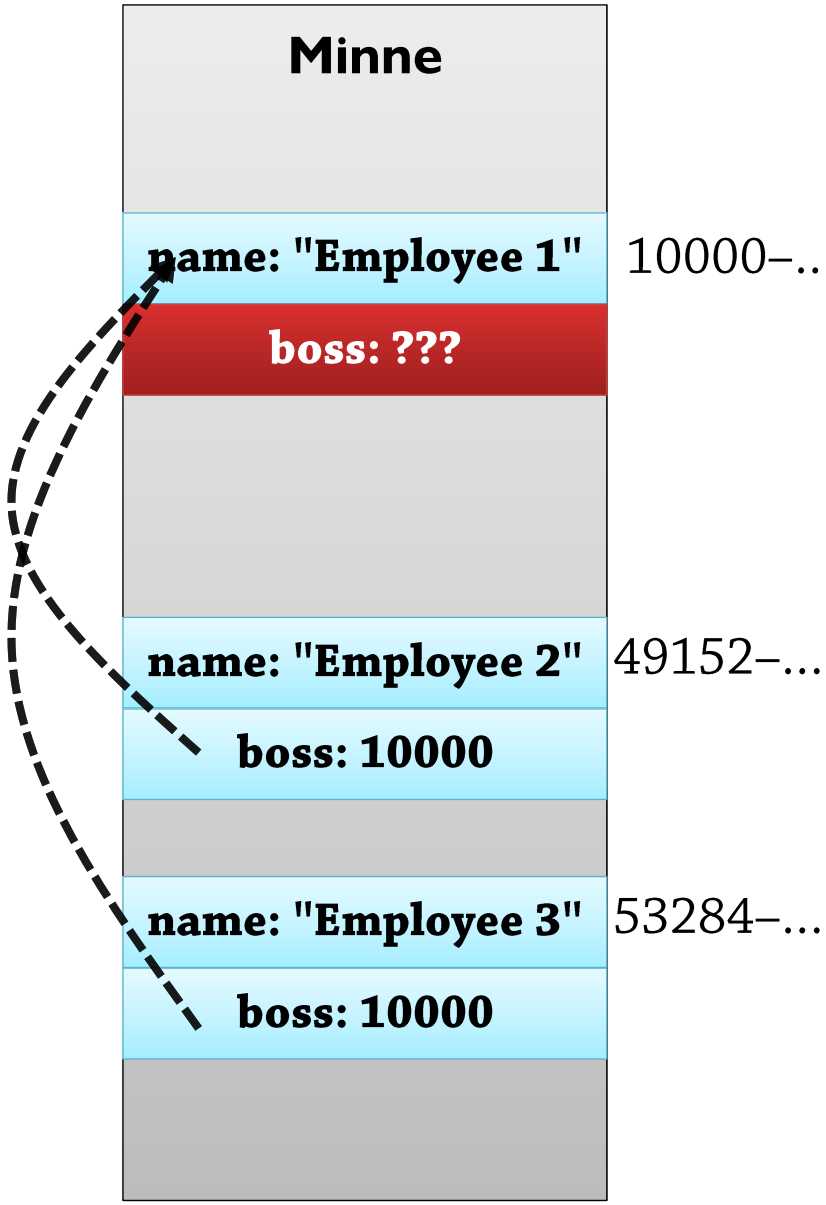


Med pekare:

Många kan peka på samma objekt trots att själva objektet bara lagras en gång

Pekarexemplet (2)

- Men anställd 1 har ingen chef!
  - Vad ska **boss**–fältet ha för värde?



Null-pekare

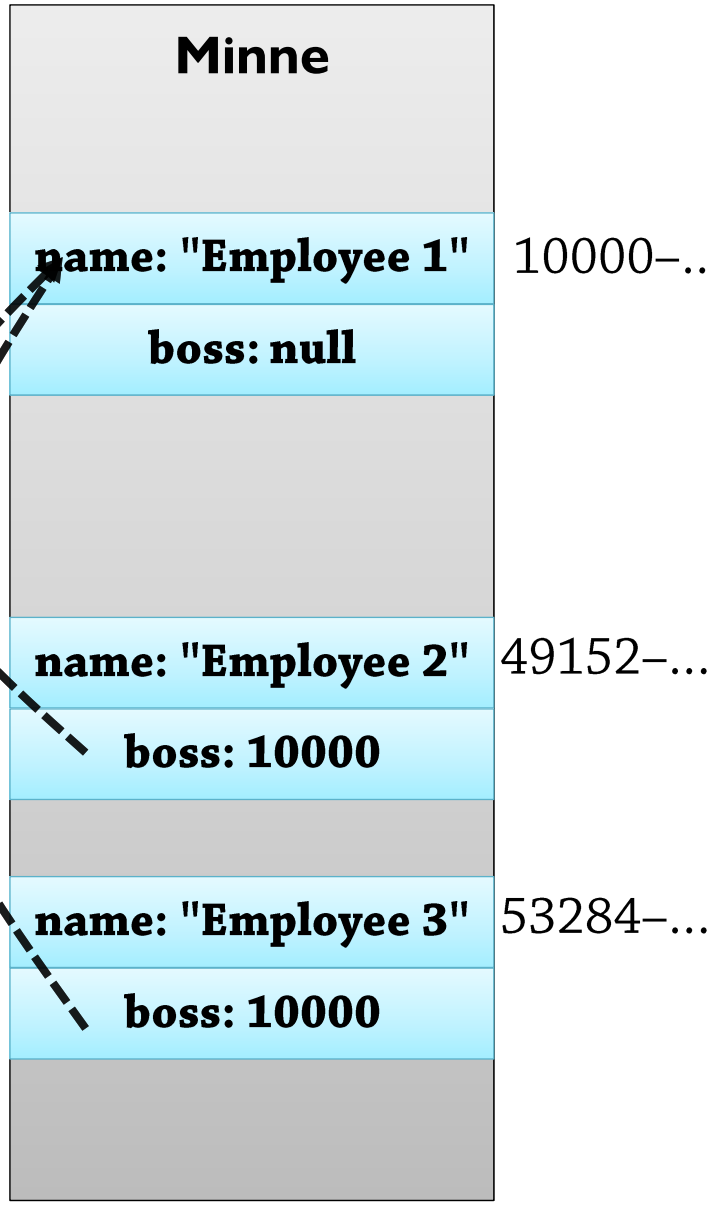
jonkv@ida

- **Objektpekare** kan ha *specialvärdet* **null**
  - Pekar "ingenstans"
    - ➔ "inte applicerbart": Noden *har* ingen chef
    - ➔ "vi vet inte än"
    - ➔ …

Med pekare:
Kan lätt ange avsaknad av värde

(Men int är inte objekt; måste ha ett numeriskt värde)

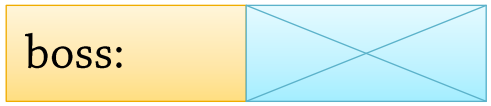
Skillnad från Python (överkurs):
None är ett *speciellt objekt*, inte en null-pekare



Null-pekare (2)

jonkv@ida

- Vad kan man **göra** om pekarens värde är null?
  - Använda själva **pekarvärdet**
    - **if** (**this**.**boss** == **other**.**boss**) … // Jämför pekarnas värden (null == 49152?),
// tittar inte efter något objekt
// (Motsvarar operatorn "**is**" i Python)
    - **painter**.draw(**circle1**) // Om circle1 == null,
// får parametern till draw också värdet null
  - **Inte** använda fälten och metoderna i **objektet den pekar på**!
    - Den pekar ju inte på något objekt!
    - **this**.**boss** = **null**; // OK
    - **this**.**boss**.**name** = "Hello"; // Fel vid körning: **NullPointerException**

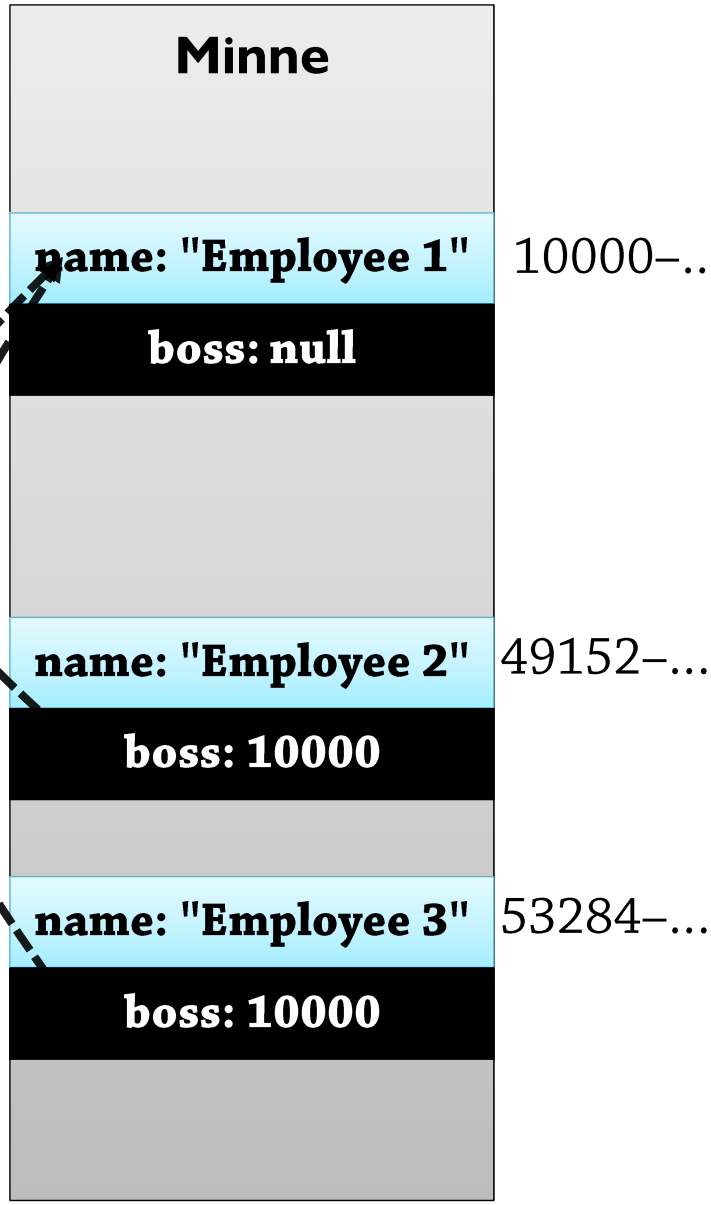


Pekare och final

Pekare och "final"

jonkv@ida

```
public class Employee {
   private String name;
   private final Employee boss;

   public Employee(String name, Employee boss) {
     this.name = name;
     this.boss = boss;
   }
}
```



- Fältet *boss* är *final* ➔ får inte ändras
  - Men *boss* är en pekare
  - Alltså får **pekaren** inte ändras
    (kan inte byta chef)
  - Men **det vi pekar på** (chefen)
    blir inte oföränderligt (chefen kan byta namn)

Modellering:
Sammansättning av objekt
(med hjälp av pekare)

Sammansättning 1

jonkv@ida

- Anta en klass för 2D-positioner:

```
public class Point {
  private double x, y;
  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }
  public double getX() {
      return x;
   }
  public double getY() {
      return y;
   }
   public double getDistFromOrigin() {
      return Math.sqrt(x*x + y*y);
   }
}
```

Sammansättning 2

jonkv@ida

- Skapa en cirkelklass – två alternativ:

Implementera allt från början

```
public class Circle {
  // Alla fält är primitiva typer, som tidigare
  private double x, y, r;
  public Circle(double x, double y, double r) {
    this.x = x;
    this.y = y;
    this.r = r;
  }
  // ...
}
```

Använd existerande punktklassen!

```
public class Circle {
  // Fält kan vara objekt(pekare)
  private Point center;
  private double r;
  public Circle(Point center, double r) {
    this.center = center;
    this.r = r;
  }
  ...
}
```

Sammansättning = composition

Sammansättning 3

jonkv@ida

- Med sammansättning:
  - En cirkel **har** en punkt, eller **består av** en punkt (och en radie)
    - Modellering – stämmer detta med vår syn?
  - Slipper skriva om existerande kod
    - **Point** kunde vara komplicerad, ha metoder som ”avstånd från origo”, …
  - Mindre upprepning – bra!

- Exempel i labb:
  - Listor finns redan
  - En kö **har** en lista där den kan lagra sina element

```
public class Circle {
  // Fält kan vara objekt(pekare)
  private Point center;
  private double r;
  public Circle(Point center, double r) {
    this.center = center;
    this.r = r;
  }
  …
}
```

Sammansättning och delegering

jonkv@ida

- Om man vill ge tillgång till "komponentens" funktionalitet: **Delegera**!

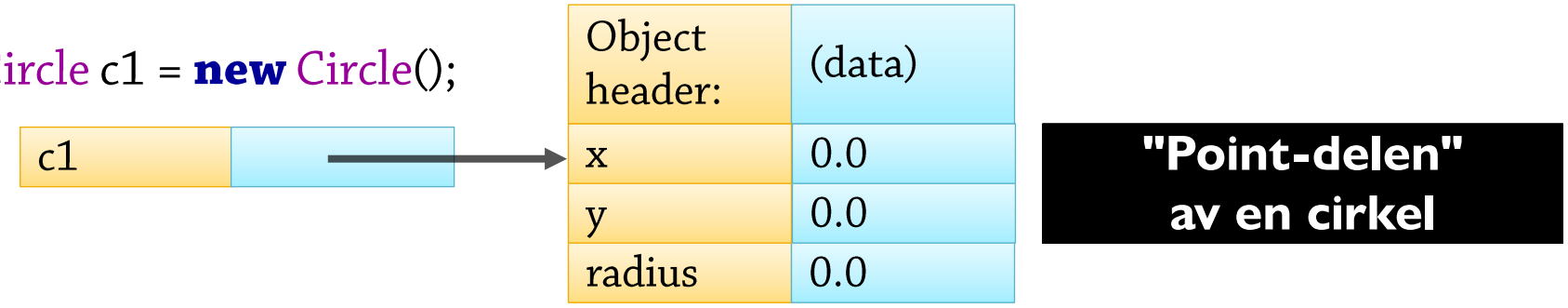
```
public class Circle {
  private Point center;
  private double r;

  public Circle(Point center, double r) {
    this.center = center;
    this.r = r;
  }
  public double getDistFromOrigin() {
     return center.getDistFromOrigin();
  }
  …
}
```

Vad är cirkelns avstånd till origo?
➔ Samma som punktens, så delegera frågan till den!

Sammansatt objektstruktur 1

jonkv@ida

- I vissa språk: Sammansatta objekt **är** sammansatta i minnet
  - Circle c1 = **new** Circle();

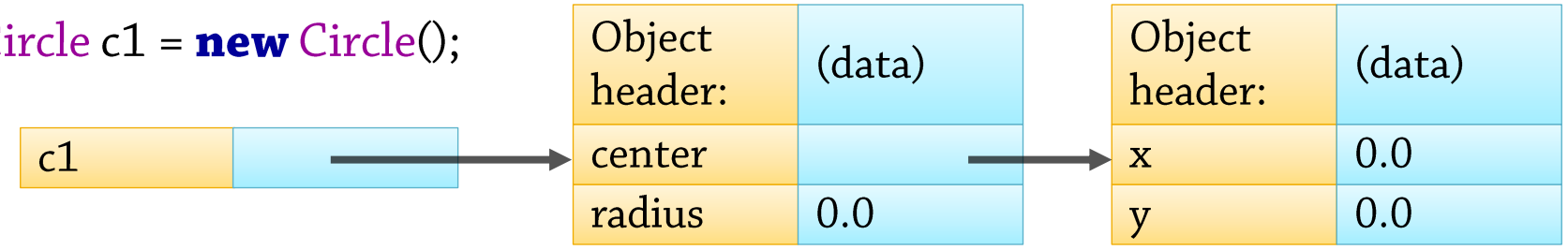



```
public class Circle {
  private Point center;
  private double r;

  public Circle(Point center, double r) {
    this.center = center;
    this.r = r;
  }
  ...
}
```

Sammansatt objektstruktur 2

jonkv@ida

- Men Javas objektvariabler är ju alltid **pekare**!
  - "Sammansatt" objekt = flera **länkade** objekt
  - `Circle c1 = new Circle();`



En **Circle** består av
– en **Point**-pekare (*inte* en **Point**)
– en double

```
public class Circle {
  private Point center;
  private double r;

  public Circle(Point center, double r) {
    this.center = center;
    this.r = r;
  }
  ...
}
```

Sammansatt objektstruktur 3

jonkv@ida

- Konsekvenser: Som vi såg för pekare tidigare
  - Exempel: Två cirklar *kan* ha **samma centrumobjekt**
    - Point center = **new** Point(10, 20);
      Circle c1 = **new** Circle(center, 7);
      Circle c2 = **new** Circle(center, 12);
  - Två listor kan innehålla (pekare till) **samma** cirkel…