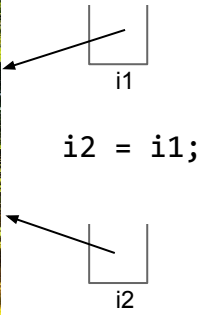
Referenser

DIT012/Joachim von Hacht

Effektivitet

Bildstorlek : 3,2 MB



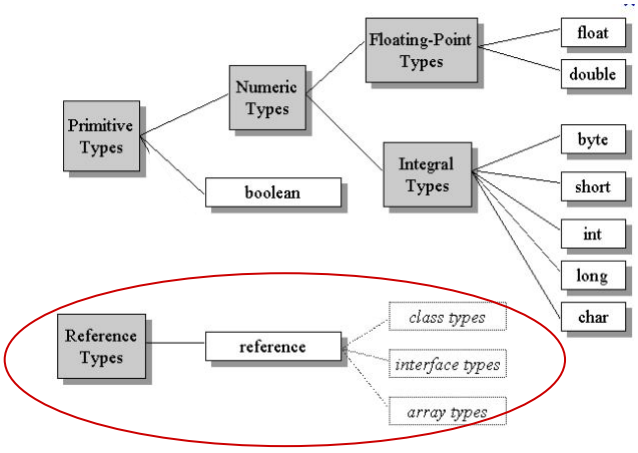
Vi har sett att vid tilldelning och metodanrop/återhopp sker det en kopiering av data

- Innebär att data kopieras från en plats i minnet till en annan plats
- Tilldelar vi en int till en heltalsvariabel kopierar vi 32 bitar/4 bytes

Antag att det som skall kopieras är mycket stort (i bytes) t.ex. en bild eller en video, kan röra sig om många MB eller GB

- En ren kopiering skulle i detta fall bli väldigt resurskrävande och ineffektiv.
- Genom att istället använda en referens och kopiera referensen blir allting väldigt mycket effektivare!
    - En referens i Java är 4 eller 8 bytes (beroende på om datorn är en 32-bitars eller 64-bitars)! Alla referenser har samma storlek i minnet.
- M.h.a. referens kan vi indirekt komma åt datan
    - Dock får vi se upp, … referenserna pekar på samma objekt!

Referenstyper



3

Referenstyper kan vara array-typer (t.ex. int[]), klasstyper (t.ex. String) eller gränssnittstyper (interface)

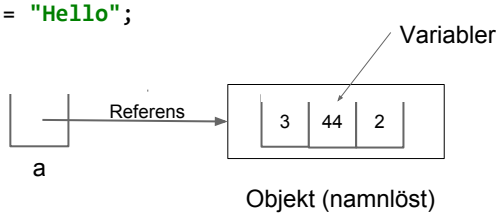
Man kan skapa nya referenstyper!

- Typsystemet är utbyggbart för array-, klass- och gränssnittstyper
  - Kallas också **egendefinierade typer** (vi definierar dem)
  - Genom att deklarera arrayer skapas nya typer utifrån en grundtyp.
  - Genom att deklarera klasser och gränssnitt skapas nya klass- respektive gränssnittstyper.

Referens kontra Primitiva Variabler

```
// Primitive variable
int points = 52;
```


```
// Reference variables
int[] a = { 3, 44, 2 };
String s = "Hello";
```



Variabler med primitiv typ, **primitiva variabler**, innehåller värdet, värdet finns i variabeln (t.ex. värdet 52 en int)

Variabler deklarerade med referenstyper innehåller inte värdet …

- … de innehåller en **referens** till ett namnlöst objekt som innehåller variabler med värdet/värdena
  - Vi säger också att referensen "pekare" på ett objekt.
- Variabler med referenstyper, kallas för **referensvariabler**
  - Enda sättet att komma åt det namnlösa objektet är via referensen (genom att skriva variabelnamnet)
  - Tappar vi referensen i referensvariabeln är objektet oåtkomligt.
    - Objektet kommer då automatiskt att tas bort ur minnet, **skräpsamlas (garbage collect)**

En mer teknisk förklaring är att en referens är en minnesadress.

- En referensvariabel innehåller en adress

Avreferering

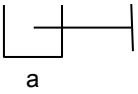
```
class Player {
  String name;
  int pts;
}

Player p = new Player();  // Reference variables
int[] a = { 3, 44, 2 };


out.println( p.pts );  // Implicit dereferencing
out.println( a[0] );   // Implicit dereferencing
```

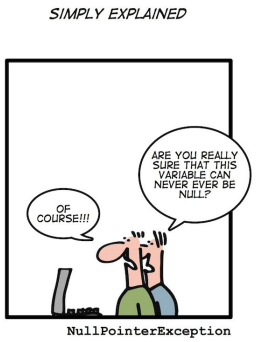
Avreferering innebär (i Java) att om vi har en referensvariabel så kommer programmet att implicit (automatiskt) "följa" referensen till objektet och därefter välja variabel eller metod.

- Så sker alltid i Java (men inte i t.ex. C/C++)!
- [ ]-operatorn och .-notationen gäller alltså för objektet, inte variabeln!

null

```
int[] a = null;

// Exception!!!
a[4] = 5;

// Will print null
out.println( a )

// Ok, b also null
int[] b = a;
```





För att beteckna att en referensvariabel inte refererar något används det speciella värdet null

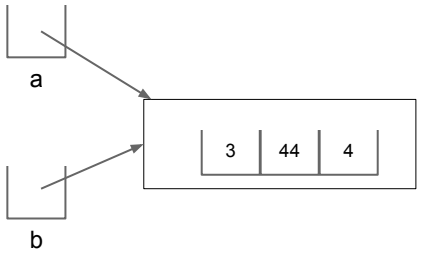
- Ett undantag uppstår om man försöker gör något med en null-referens
  - **NullPointerException, NPE!**
  - Ett ständigt och stort problem är att hålla reda på om referenser är null.
- Att tilldela en variabel null eller att skriva ut ett null-värde går dock bra.

För alla objekt, o, skall gälla att o.equals(null) är false!

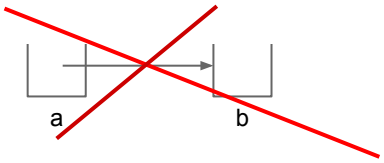
- Se bildserie klasser.

Referenser och Tilldelning

```
int[] a = { 3, 44, 2 };

int[] b = null;

b = a;
```



Detta kommer aldrig att ske i Java!
En referens pekar alltid på ett objekt, inte en annan variabel.



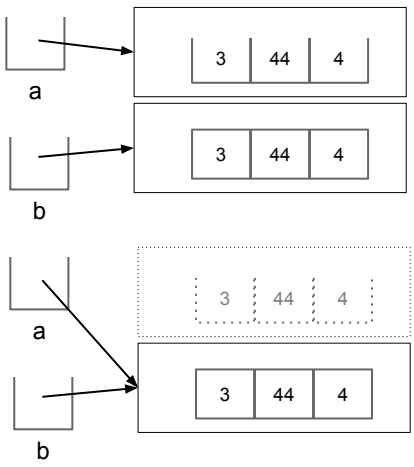
Tilldelning för referensvariabler fungerar som för primitiva typer

- Värde kopieras från vänster till höger MEN …
- .. "värdet" är en referens, det är referensen som kopieras!
- Effekten blir att två referenser pekar på samma objekt!

En referens kan aldrig direkt referera en variabel, alltid ett objekt!

Referenser och Likhet

```
int[] a = { 3, 44, 4 };

int[] b = { 3, 44, 4 };


// False
out.println(a == b);


a = b;

// True
out.println(a == b);
```



Likhet för referensvariabler fungerar som för primitiva variabler

- Innehållet i variablerna jämförs MEN …
- … innehåller är nu referenser!
- Likhet innebär att referenserna refererar samma sak (pekar på samma objekt)
    - Innehåller samma adress!

Efter tilldelningen, a = b ovan, pekar referenserna på samma objekt.

Att tilldelning och likhet för referenser får en speciell betydelse sammanfattas som **referenssematik**

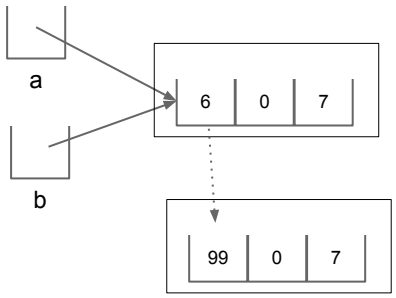
- Semantiken (betydelsen) blir annorlunda pga av vi använder referenser
- För primitiva typer säger man **värdesematik**

OBS! String är en referenstyp, innebär att == inte "fungerar" för strängar!

- Normalt är de ju själva texten vi vill jämföra men == jämför referenserna.
- Se bildserie om strängar.

Alias-problem

```
a = b;


a[0] = 99;


// b changed!!!
if( b[0] == 99 ){
    // true
}
```

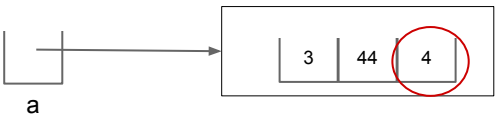


Att flera referenser kan peka på samma objekt innebär att det finns flera sätt att ändra variablerna i objektet.

- Ändringen kan ske “bakom ryggen” på oss.
- Kallas **alias-problem,** den ena referensen är ett alias för den andra
- Kan leda till svårlösta fel i program (går inte att undvika i imperativa språk med referenser)

Konstanta referenser

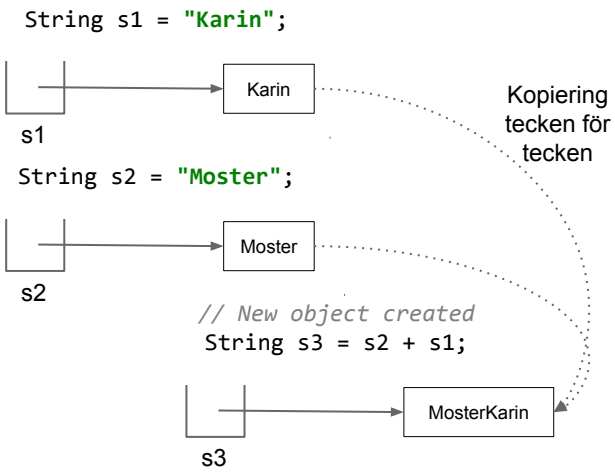
```
// Constant reference variable a
final int[] a = { 3, 44, 2 };

// Error! a is final
a = new int[5];

// Ok, variables in object not final
a[2] = 4;
```



En konstant referensvariabel kan inte ändras.

- Objektet den referera kan däremot ändras!

Strängar och +-operatorn



Konkatenering med +-operatorn innebär att ett nytt strängobjekt skapas och en referens till detta returneras.

- Eftersom strängar inte kan ändras (tecknen kan inte ändras)
- Tecknen från operanderna kopieras till det nya objektet.
- +-operatorn kan vara ineffektiv t.ex. i en loop med många varv (kopierar samma sak och mer och mer för varje varv)

Referenser som returvärden

```
class Accumulator {
   int n;
   Accumulator add( int i ){
       n += i;
       return this;   // Make chaining possible
   }
   int getN(){ return n; }
}

int i = a.add(1).add(2).add(3).getN();
```


En variant för att möjliggöra kedjade anrop är att returnera this