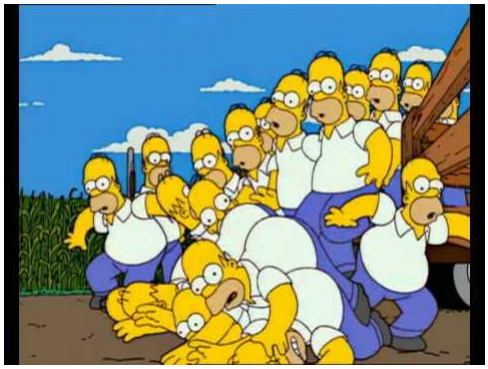
Arrayer

TDA548/Joachim von Hacht

Många av Samma



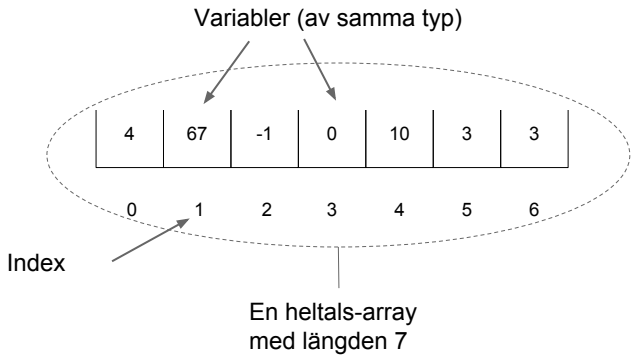
Literaler och variabler kan användas för enstaka värden.

- Men vad händer om vi behöver många värden …
- … 10 heltalsvariabler, eller 100, eller 100000 …. ?
- Orimligt att deklarera dessa en och en.
- Lösning: Om vi behöver <u>många variabler av samma typ</u> kan vi använda en array …

Finns tyvärr inget bra svenskt namn, “vektor” och “fält” förekommer.

- Vi använder det engelska namnet array.

Konceptuell bild



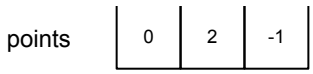
3

En array är en konsekutiv (inga tomrum) följd av variabler.

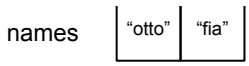
- Längden bestäms en gång för alla, den förändras aldrig!
- Varje variabel har ett index.
- Index börjar på 0 och slutar på längden-1.

Arrayer*)

```
// Declare and initialize array "points"
int[] points = {0, 2, -1};
```




```
String[] names = {"Otto", "Fia"};
```



*) Förenklat, mer senare

För att använda en array måste vi deklarera och initiera den.

- Görs som vanligt med: Typ, namn och ev. initiering
- Typ anges med typen för enskilda variabler + [ ] (man utgår från en "grundtyp")
  - Vi säger t.ex. int-array för typen int[]
- Namnet gäller för hela arrayen (de enskilda variablerna i array:en har inga namn).
- Initierar genom att ange värdena för de enskilda variablerna i en lista då vi deklarerar arrayen.
  - Enskilda variabler initieras med värden från listan i skriven ordning (från vänster till höger)
  - Array:en kommer att innehålla lika många variabler som värden vi angav

Utskrift av Array:er

```
int[] points = { 1, 2, 3, 4, 5 };

// Will print like [I@330bedb4
out.println(points);

// Better, use "built in" helper
String s = Arrays.toString(points);
out.println(s);  // Will print [ 1, 2, 3, 4, 5 ]
```


Att direkt använda out.println() ger en (normalt) oanvändbar/obegriplig utskrift typ: [I@7f31245a

- Arrays.toString(), omvandlar arrayen till en lättläst sträng som sedan kan skrivas ut.

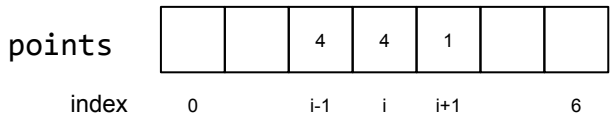
Indexering

```
int[] points = {0, 0, 0};

points[0] = 4;                  // { 4, 0, 0 }
points[2] = 1;                  // { 4, 0, 1 }
points[1] = points[2];          // { 4, 1, 1 }
points[0] == points[2];         // false
points[0] > points[1];          // true

points[6] = 3;                  // Exception
points[-1] = 3;                 // Exception
```


Indexering innebär att komma åt de enskilda variablerna, **elementen**, i en array

- Indexering skrivs: *array-namn* [*index*]
  - [ ] kallas indexeringsoperatorn
- Index måste var ett heltalsuttryck (uttryck t.ex. en variabel går bra).
- Vi måste själva se till att index inte hamnar utanför array:en, …
  - … om utanför så undantag (exception) vid körning, ett **exekveringsfel**

Array som Datastruktur




```
int[] points = {0, 0, 0, 0, 0, 0, 0};

int i = 3;
points[i] = 4;            // { 0,0,0,4,0,0,0 }
points[i+1] = 1;          // { 0,0,0,4,1,0,0 }
points[i-1] = points[i]; // { 0,0,4,4,1,0,0 }
```


En array har en struktur, det finns första/sista, före/efter, vänster/höger

- Vi säger att en array är en **datastruktur**.
- Ger oss möjlighet indirekta komma åt variabler: “grannen till”, “tre efter”, “ en före”, …
- Givet ett index i kommer vi åt
  - variabeln till vänster (före) med i-1
  - variabeln till höger (efter) med i+1
  - i-1 eller i+1 får inte hamna utanför array:en, om så: undantag (som tidigare).

Längden av en Array

```
int[] points = { 0, 5, 2, 0, 9 };

out.println(points.length);   // 5

// Last index!
out.println(points[points.length-1]); // 9
```


Man kan komma åt längden av en array på ett fördefinierat sätt (inbyggt i språket)

- Genom att skriva: *array-namn*.length

Traversering av Array:er

```
int[] arr = { 1, 2, 3, 4, 5, 6 };


for( int i = 0 ; i < arr.length ; i++){
   out.print(arr[i]);    //123456
}


for( int i = arr.length - 1 ; i >= 0 ; i--){
   out.print(arr[i]);   //654321
}
```

9

Traversering

- Ofta vill man **traversera** (genomlöpa) en array d.v.s. komma åt alla variabler i tur och ordning
  - Från vänster till höger eller tvärtom.
- Traversering görs vanligen med en for-sats (eftersom vi vet längden = antal varv)
- Som villkor använder man: i < array.length (vänster till höger)
  - D.v.s. i skall vara strikt mindre än längden eftersom sista index är ett mindre än längden
- eller i >= 0 (höger till vänster), större eller lika med eftersom första index är 0.

Inläsning till Array

```
String[] names = new String[3];

out.print("Input 3 names (enter after each) > ");
for (int i = 0; i < names.length; i++) {
   names[i] = sc.nextLine();
}
```


Måste använda en loop.

- Finns ingen genväg

Mer om Initiering

```
int[] points1;    // Declare arrays
int[] points2;

// Initialize [2, 9, 0, -1, 4]
points1 = new int[]{2, 9, 0, 1, -4};

String[] names;
names = new String[]{“Otto”, “Fia”,“Pelle”, “Siv”};

// No list, default values set, must supply length
points2 = new int[3];  // [0, 0, 0]
points2 = new int[points1.length]; // Or same length
```

11

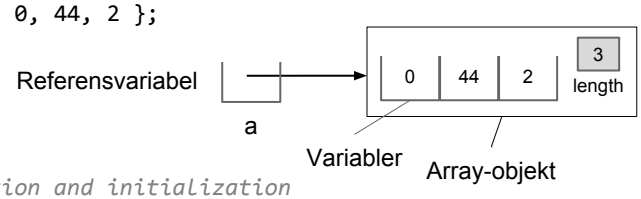
Man kan initiera arrayer på flera platser i programmet (inte bara vid deklarationen)

För att initiera en array på någon annan plats än vid deklarationen

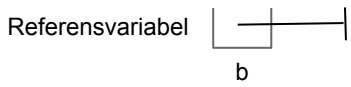
- Använd operatorn **new** + array-typen + [ ] + en lista med värden (mer senare)
- Om ingen lista med värden anges måste man ange ett värde för arrayens längd inom hakparenteserna
    - Variablerna får då förbestämda värden beroende på typ, int ger t.ex. 0.

Mer om Arrayer

```
// Declaration and initialization, array-object created
int[] a = { 0, 44, 2 };
```




```
// Declaration and initialization
// but NO array-object!
int[] b = null;
```



12

En variabeldeklaration ger en variabel!

- Men m.h.a. en array-deklaration kan vi plötsligt skapa flera variabler?
- Hur kommer det sig?

Förklaring:

- En deklaration av en array ger en (enda) referensvariabel!
- Värdet i variabeln är en referens till ett (namnlöst) **array-objekt** som i sin tur innehåller ett antal (namnlösa) variabler
- Vi kan bara nå objektet indirekt, via referensen och indexering.
  - Tappar vi bort referensen har vi tappat bort objektet, vi kan aldrig komma åt det igen.
- Vi kan välja om vi vill att variabeln skall peka på ett array-objekt eller ej
  - Om inte sätter vi värdet till **null**
- Array-objektet innehåller en konstant variabel: lenght, för att komma åt den använder vi punktnotation.
  - Vi ritar aldrig ut length förutom i denna bild.

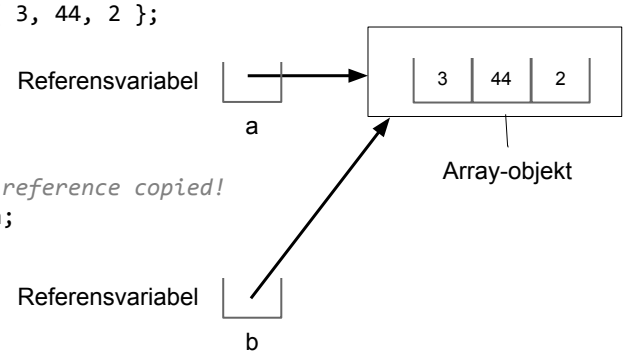
Att det skapas ett objekt i samband med deklaration och initieringen kallas att objektet instansieras (objektet är en instans av typen array)

- Samma effekt har operatorn new, den instansierar ett objekt som vi därefter kan initiera

Indexering betyder: “Gå till objektet som refereras, välj variabel utifrån index …”

Tilldelning och Likhet

```
// Declaration and initialization
int[] a = { 3, 44, 2 };
```




```
// Assign, reference copied!
int[] b = a;
```



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

13

Tilldelning:

- Eftersom array-variabler är referenser kommer en tilldelning att göra så att två referenser pekar på samma array-objekt

Likhet:

- Om två variabler pekar på samma objekt är de lika (d.v.s. som vanligt, de innehåller samma värde, nämligen en referens till objektet)
- Vill vi definiera någon annan typ av likhet för arrayer, t.ex. lika långa och samma värde för samma index, får vi själva implementera detta (t.ex. skapa en metod)

Testning av Arrayer

```
int[] arr = { … };


// Simple way to test expected values of array
// Convert to string and compare strings
out.println(Arrays.toString(arr)).equals("[1, 2, 3]"));
```


För att förenkla vid testning kan man omvandla arrayen till en sträng (så slipper vi loopar)

- Nackdel: Det gäller att skriva rätt i det förväntade resultatet!

Matriser

```
// Declare and initialize (instantiated automatically)
int[][] m = {       // An array of arrays
       { 1,2,3,},          // m[0]
       { 4,5,6,},          // m[1]
       { 7,8,9,},          // m[2]
   };

int[][] m;          // Just a single variable
m = new int[3][3];  // Instantiate object, default values 0
m = new int[2][2]{  // Instantiation and initialization
    {11, 12},
    {21, 22}
};
```

15

I Java kan man ha arrayer av godtycklig dimensioner (< 255)

Tvådimensionella array:er vanligt

- Tekniskt en array av array:er
- Vi kallar 2D arrayer för matriser
- En matrisvariabel deklareras med dubbla [ ] -parenteser efter typen
  - Först index anger vilken array (rad), …
  - … andra anger, element i aktuell array (kolumn)
  - Rad och kolumnindex börjar som vanligt på 0.
- Vi använder bara rektangulära matriser (inget krav, “ragged 2D arrays” är tillåtet)

Indexering på Matriser

```
int[][] m = { { 1,2,3,}, { 4,5,6,}, { 7,8,9,} };

// [row][col]
m[0][2] = 99;  // {{ 1,2,99,},{ 4,5,6,},{ 7,8,9,}}
m[2][1] = m[0][2]; // {{ 1,2,99,},{ 4,5,6,},{ 7,99,9,}}

// Exception
m[0][3] = 45;
```


För att komma åt enskilda element används som tidigare indexering (men nu med två index)

- Alltid rad, kolumn.

Traversera Matris

```
int[][] m = { { 1,2,3,}, { 4,5,6,}, { 7,8,9,} };


// Traverse
for(int row = 0 ; row < m.length ; row++){
   for( int col = 0 ; col < m[row].length; col++){
       out.print( m[row][col]);
   }
   out.println();
}
```


Traversering med nästlade for-loopar och length.

- Bild visar en typ av traversering (kvadratisk matris)

OutOfBoundsException

```
for (int i = 0; i < arr.length - 1; i++) {
   for (int j = 0; j < arr.length - i - 1; j++) {
       if (arr[j] > arr[j + 1]) {  // Out of bounds???
            ...
// Check: 0 <= j + 1 < arr.length for min and max indices
Assume: i = 0, j = 0
j + 1 = 1   (OK)

Assume: i = 0, j = arr.len - i - 2 (j's last value)
j + 1 = arr.len - 0 - 2 + 1 = arr.len - 1   (OK)

Assume: i = arr.len - 2, j = arr.len - i - 2
    j + 1 = arr.len - i - 2 + 1 =
    arr.len - (arr.len - 2) - 2 + 1 = 1    (OK)
```

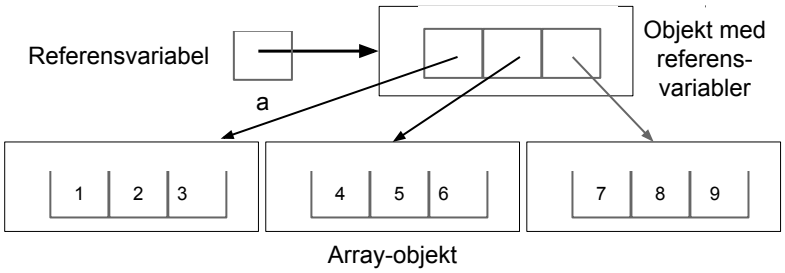
18

Om vi indexerar utanför arrayen/matrisen får vi ett undantag under körning, OutOfBoundsException

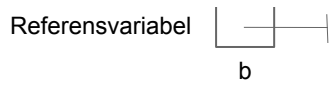
- Komplexa loopar bör vi alltid kontrollera så att för alla index i, så 0 <= i < length-1
- Eftersom indexeringen normalt är linjär, räcker det att kontrollera att första och sista värde för i inte hamnar utanför indexgränserna
    - Kan göras med några enkla beräkningar

Mer om Matris-variabler

```
int[][] a = { { 1,2,3,}, { 4,5,6,}, { 7,8,9,} };
```




```
// NO matrix object
int[][] b = null;
```

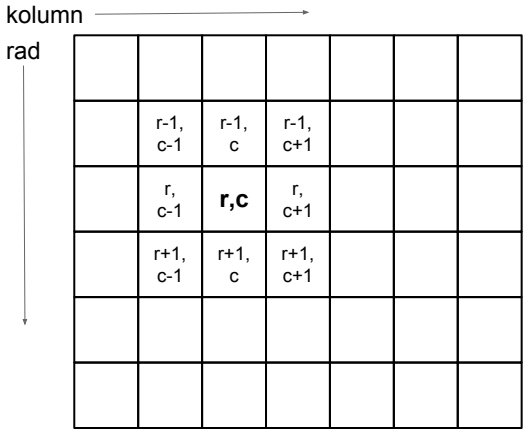


19

En deklaration av en matrisvariabel ger, på samma sätt som en array, en referensvariabel

- Objektet referensen pekar på innehåller i sin tur variabler av referenstyp

Matris som Datastruktur



20

En matris ger oss en datastruktur

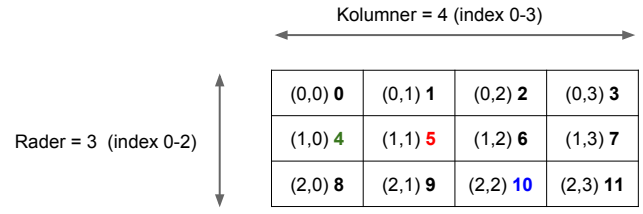
- Fler möjligheter: vänster/höger/över/under/snett över/snett under ...

Testning av Matriser

```
int[][] m = {
        {0, 1, 0,},
        {1, 1, 1,},
        {0, 1, 0,},
};

out.println(Arrays.toString(m[0]).equals("[0, 1, 0]"));
// m[1] false!
out.println(Arrays.toString(m[1]).equals("[1, 0, 1]"));
out.println(Arrays.toString(m[2]).equals("[0, 1, 0]"));
```


Ungefär som för arrayer fast vi får skriva ut alla rader.

Byte mellan Array och Matris



Array -> Matris:

radindex = index / kolumner (5 / 4 = 1)

kolumnindex = index % kolumner (10 % 4 = 2)

Matris -> Array:

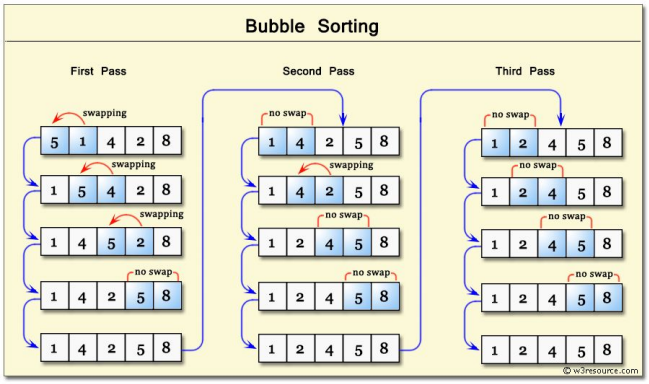
index = radindex * kolumner + kolumnindex (1 * 4 + 0 = 4)

22

Omvandling mellan array och en matris för någon typ av data kan behövas

- Datan är oförändrad det är bara strukturen som ändras.
- Ibland lättare att arbeta med array-format …
- … ibland lättare med matris.

Algoritmer



23

Typiska saker man vill göra med arrayer och matriser: Söka, sortera, flytta/hitta element utifrån visst kriterium, …

- Finns många färdiga lösningar, i form av standardalgoritmer, till dessa problem
- Algoritm
- Bilden: Sorteringsalgoritmen “Bubblesort”

Konvertering Array och List

```
// Use helper class Arrays to create unmodifiable
// list
List<Integer> iList1 = Arrays.asList(1, 2, 3, 4);

// Create modifiable list out of unmodifiable
List<Integer> iList2 = new ArrayList(iList1);

// Convert back to array. Must supply an array
// object as argument
Integer[] iArr = iList1.toArray(new Integer[]{});
```


Tekniskt lite rörigt, men möjligt att byta mellan List och Array

- Inget att kunna utantill
- Se Samlingar