

9. Java Arrays und Strings

Allokation, Referenzen, Elementzugriff, Mehrdimensionale Arrays, Strings, Stringvergleiche

Arrays

Arrayvariable deklarieren: `int [] z;`

Arrays

Arrayvariable deklarieren: `int [] z;` \emptyset

Arrays

Arrayvariable deklarieren: `int [] z;` \emptyset

Array erzeugen: `z = new int [5];`

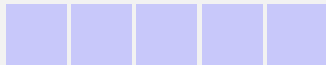
Arrays

Arrayvariable deklarieren: `int [] z;`

z

∅

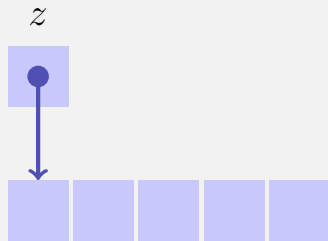
Array erzeugen: `z = new int [5];`



Arrays

Arrayvariable deklarieren: `int [] z;`

Array erzeugen: `z = new int [5];`

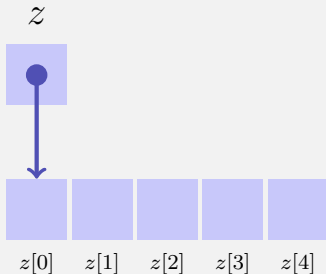


`z` ist eine *Referenz* auf die Arraydaten,

- aber erst nach der Zuweisung zu den erstellten Daten
- sonst zeigt es nirgendwo hin: `null`.

Arrays

```
int [] z;
```



```
z = new int [5];
```

Elemente werden indiziert. Index beginnt bei 0 und endet bei Arraygrösse – 1.

Elementzugriff: `name[index]`

Arrays in Java sind “zero based”



Arrays sind dynamische Objekte

Arrays sind grundsätzlich *dynamisch* erzeugt.

```
int[] b;  
b = new int[10]; // 10 elements with index 0...9  
...  
b = new int[20]; // can be reassigned
```

Grösse eines Arrays kann also zur Laufzeit festgelegt werden.
Ein Array wächst jedoch nicht automatisch!

Arrays sind nicht primitiv

Arrays tragen *Metadaten* mit sich herum:

```
int sq = new int[7];
for (int i = 0; i < sq.length; ++i){
    sq[i] = i * i;
}
sq[8] = 64; ← java.lang.ArrayIndexOutOfBoundsException!
```

... auch über Methodengrenzen hinweg (nächstes mal)

```
static void print(int[] a){
    for (int i = 0; i < a.length; ++i){
        Out.println("a[" + i + "]=" + a[i]);
    }
}
```

Arrayzuweisungen

```
int[] z = new int[5];
```

```
for (int i=0; i<z.length; ++i) {  
    z[i] = i*i;  
}
```

```
int[] x = z;
```

```
int j = x[2];
```

```
x[1] = 99;
```

Bei der Zuweisung von Arrays *wird die Referenz kopiert*, nicht die Daten!

Arrayzuweisungen

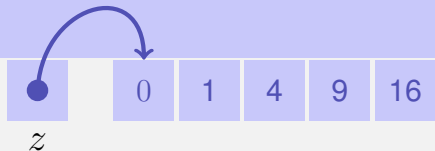
```
int[] z = new int[5];
```

```
for (int i=0; i<z.length; ++i) {  
    z[i] = i*i;  
}
```

```
int[] x = z;
```

```
int j = x[2];
```

```
x[1] = 99;
```



Bei der Zuweisung von Arrays *wird die Referenz kopiert*, nicht die Daten!

Arrayzuweisungen

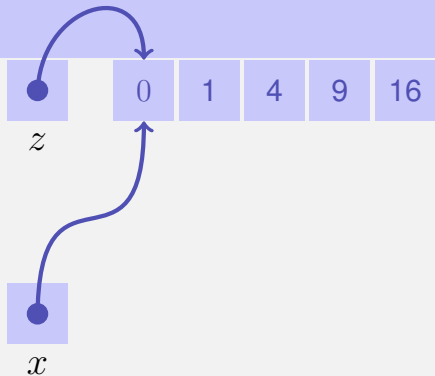
```
int[] z = new int[5];
```

```
for (int i=0; i<z.length; ++i) {  
    z[i] = i*i;  
}
```

```
int[] x = z;
```

```
int j = x[2];
```

```
x[1] = 99;
```



Bei der Zuweisung von Arrays *wird die Referenz kopiert*, nicht die Daten!

Arrayzuweisungen

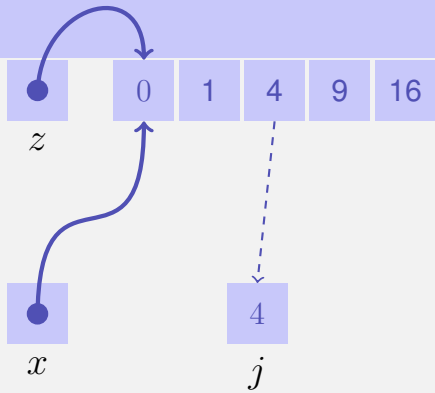
```
int[] z = new int[5];
```

```
for (int i=0; i<z.length; ++i) {  
    z[i] = i*i;  
}
```

```
int[] x = z;
```

```
int j = x[2];
```

```
x[1] = 99;
```



Bei der Zuweisung von Arrays *wird die Referenz kopiert*, nicht die Daten!

Arrayzuweisungen

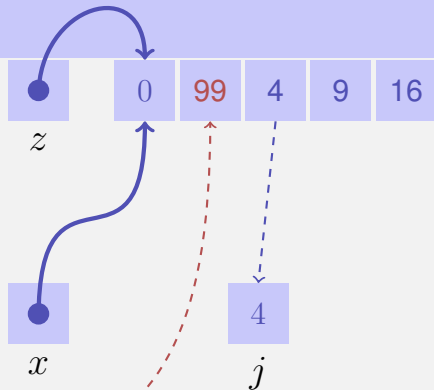
```
int[] z = new int[5];
```

```
for (int i=0; i<z.length; ++i) {  
    z[i] = i*i;  
}
```

```
int[] x = z;
```

```
int j = x[2];
```

```
x[1] = 99;
```



Bei der Zuweisung von Arrays *wird die Referenz kopiert*, nicht die Daten!

Beispiel

Annahme: Ein (unsortiertes) Array x enthalte nur die Zahlen aus dem Bereich $[0, \dots, 9]$.

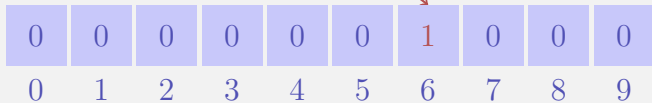
Aufgabe: Schreibe ein effizientes Programm, welches für jede solche Zahl ausgibt, wie oft sie in x vorkommt.

Idee

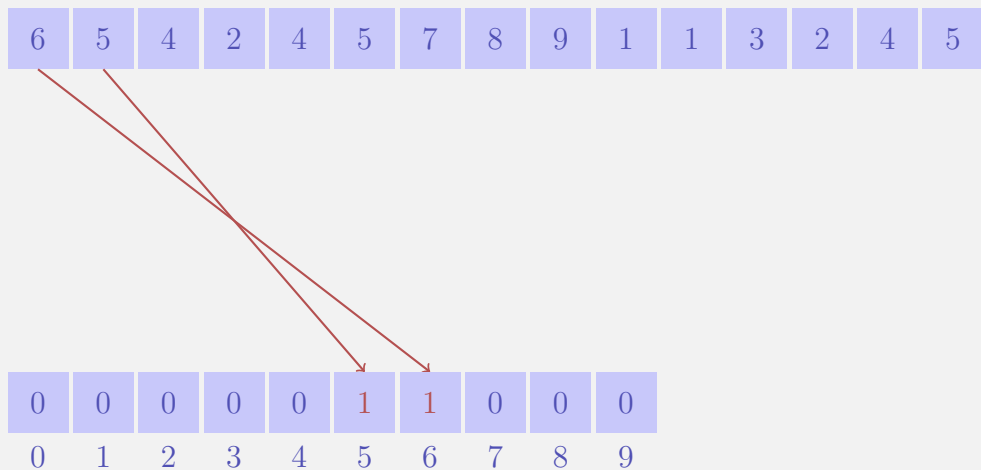
6	5	4	2	4	5	7	8	9	1	1	3	2	4	5
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

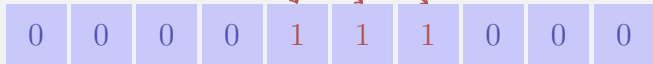
Idee



Idee

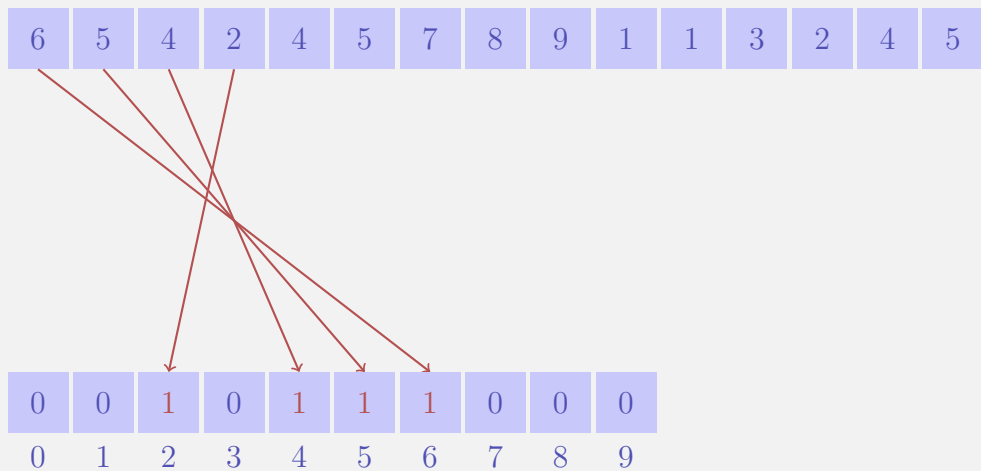


Idee

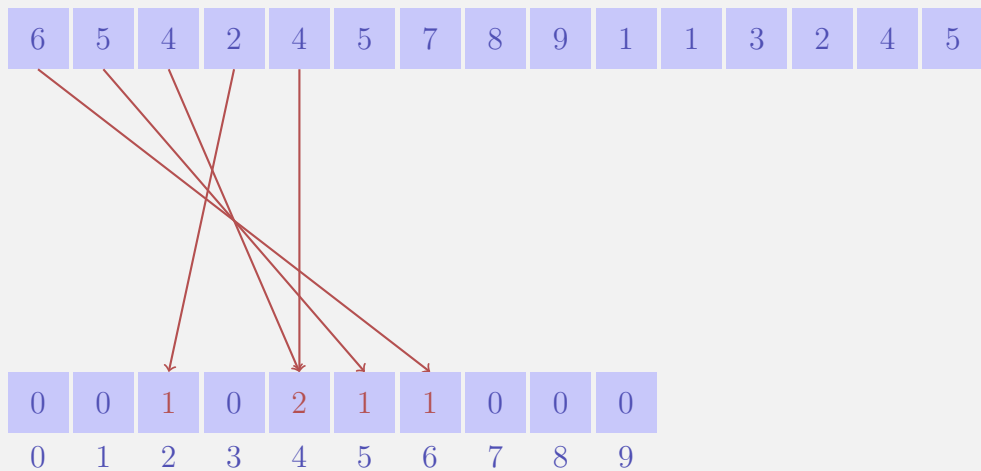


0 1 2 3 4 5 6 7 8 9

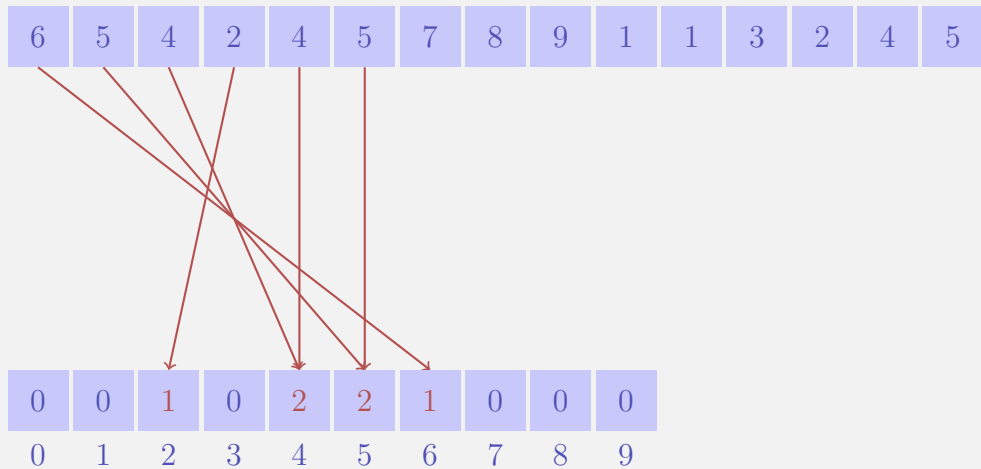
Idee



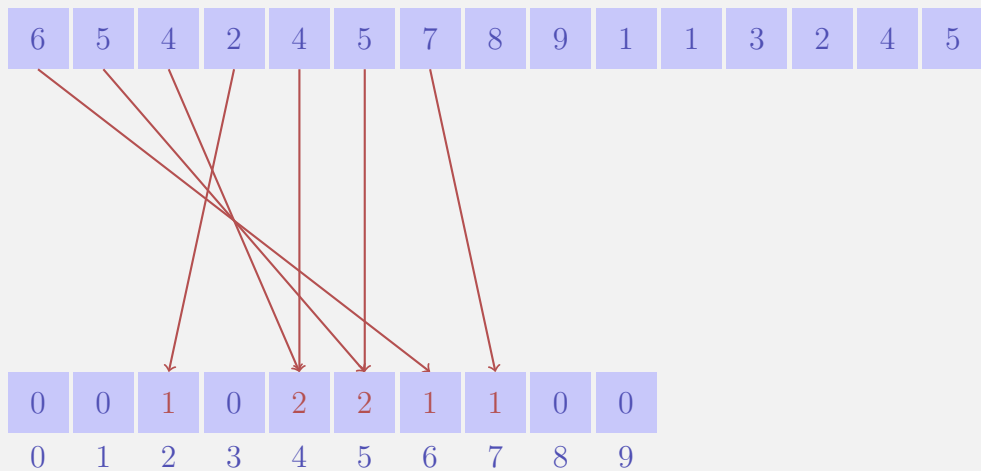
Idee



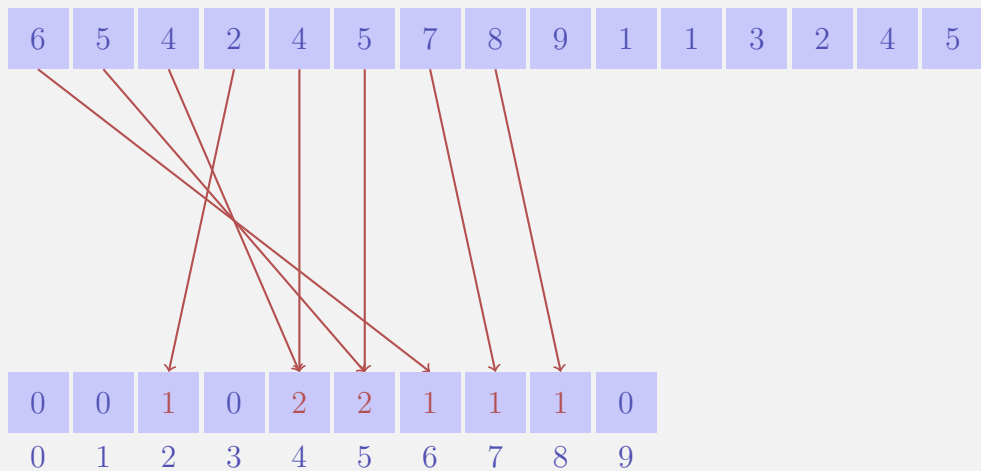
Idee



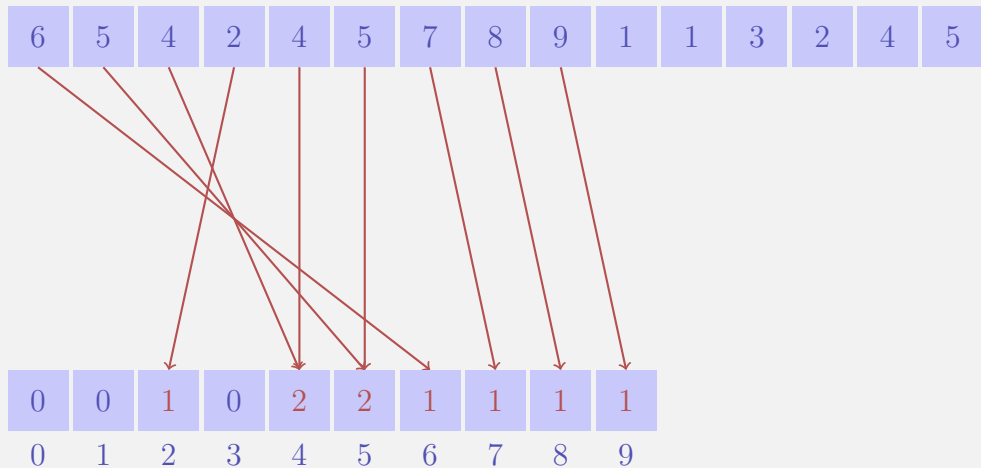
Idee



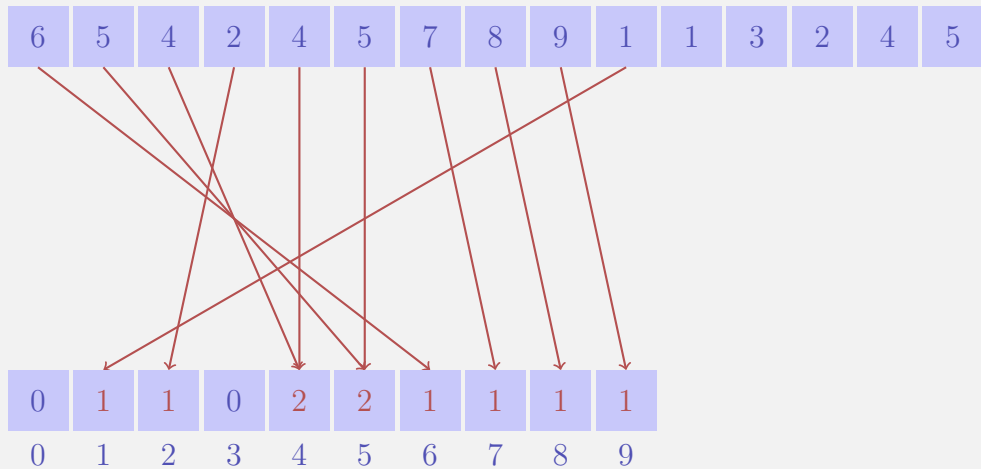
Idee



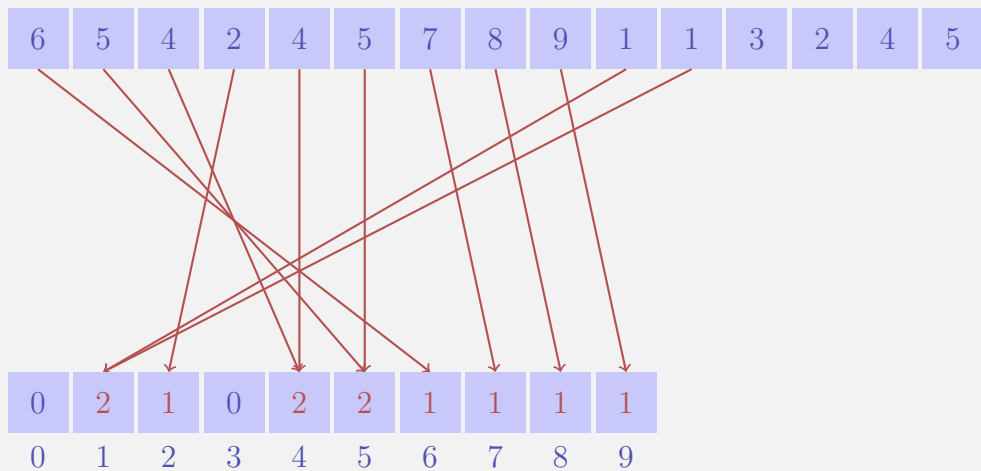
Idee



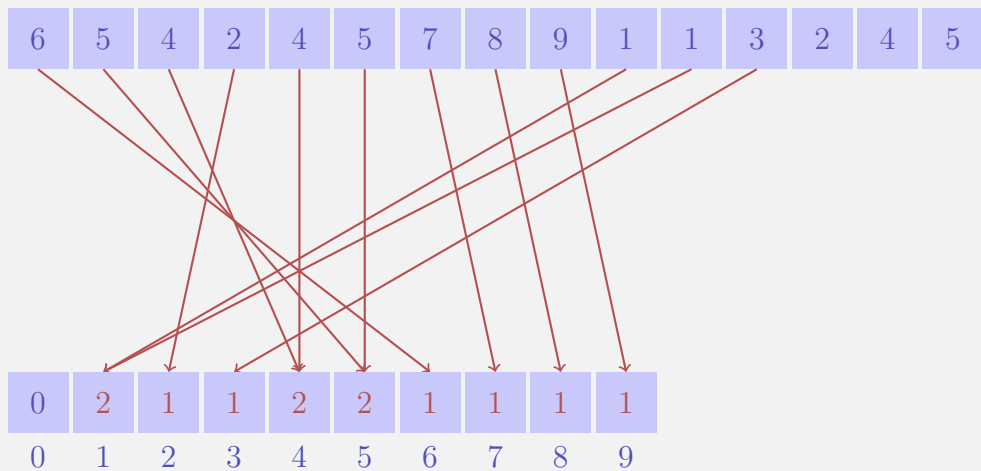
Idee



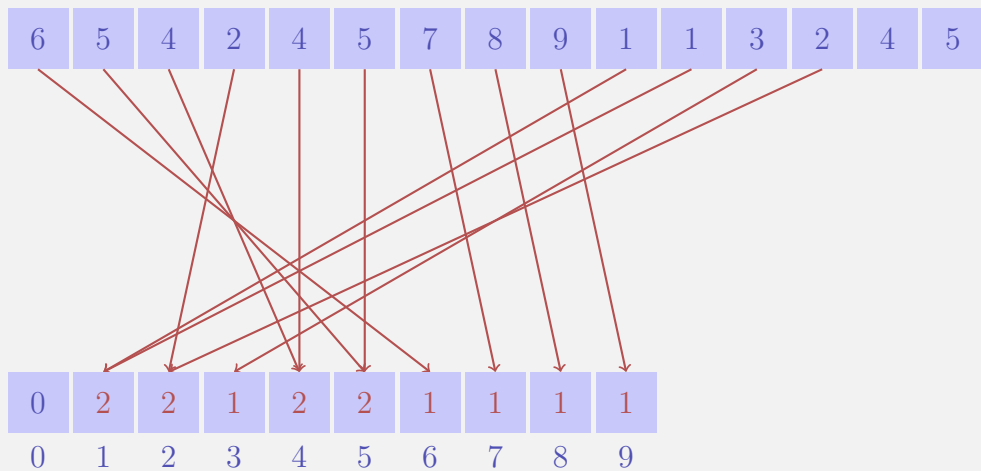
Idee



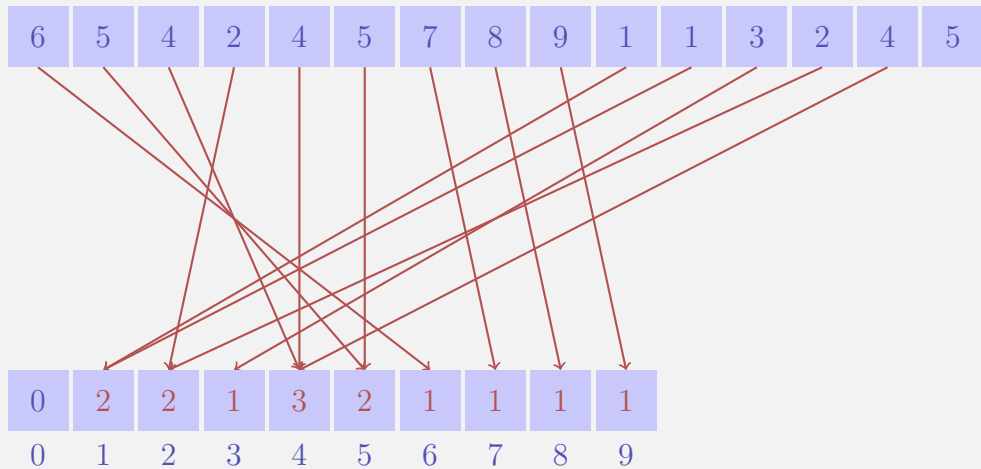
Idee



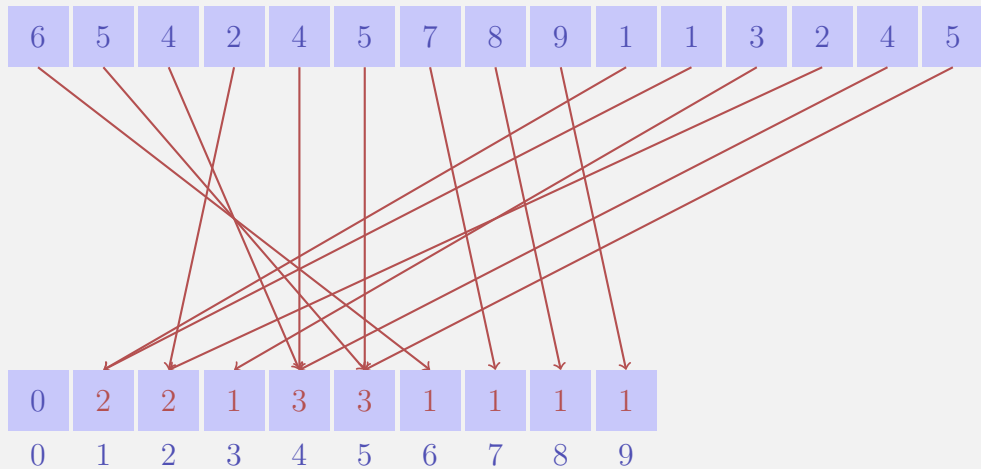
Idee



Idee



Idee



Code

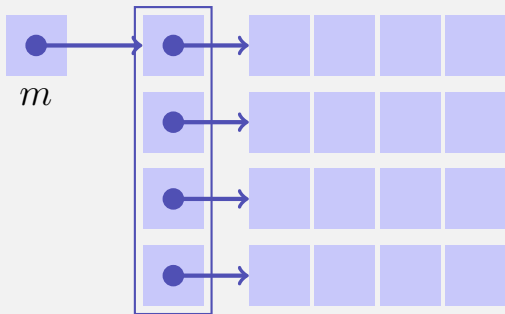
```
public class CountNumbers {
    public static void main(String [] args) {
        int [] numbers = {5, 4, 2, 4, 5, 7, 8, 9, 1, 1, 3, 2, 4, 5};
        int [] index = new int [10];

        for (int i = 0; i < numbers.length ; i++) {
            index [numbers[i]]++;
        }

        for (int i = 0; i < index.length ; i++) {
            Out.println("Count (" + i + ")=" + index[i]);
        }
    }
}
```

Mehrdimensionale Arrays

```
double[][] matrix = new double[4][4];
```



Mehrdimensionale Arrays

```
double[] [] matrix = new double[4][4];

// Identity matrix
for (int r=0; r < matrix.length; ++r){
    for (int c=0; c < matrix[r].length; ++c){
        if (r==c){
            matrix[r][c] = 1;
        } else {
            matrix[r][c] = 0;
        }
    }
}
```

Mehrdimensionale Arrays

Ein zweidimensionales Array ist ein Array von Referenzen auf eindimensionale Arrays. Also geht auch das:

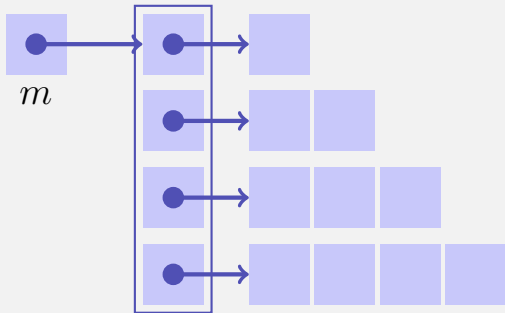
```
double[] [] matrix = ...;

for (int r = 0; r < matrix.length; ++r){
    double[] vector = matrix[r];
    for (int c = 0; c < vector.length; ++c){
        Out.print(vector[c] + " ");
    }
    Out.println();
}
```

Mehrdimensionale Arrays

Es geht sogar das:

```
double[] [] m = new double[5] [];  
for (int r = 0; r < m.length; ++r) {  
    m[r] = new double[r+1];  
}
```



Array-Vergleiche

Erneut aufgepasst: Arrays sind Referenzen!

```
double[] x = {1,2,3};  
double[] y = x;  
double[] z = {1,2,3};
```

```
if (y == x) {...} // y==x is true  
if (z == x) {...} // z==x is false!
```

Für Kenner:

```
if (z.equals(x)) {...} // z.equals(x) is also false !!  
if (Arrays.equals(x,z)) {...} // Arrays.equals(x,z) is true.
```

Vorsicht bei `Arrays.equals` bei mehrdimensionalen Arrays! (Was wird wohl geprüft?)

Strings

String: ein Objekt, welches Zeichenketten speichert.

```
String name = "Informatics";  
String university = "ETH";  
String lecture = name + " at " + university;  
int x = 3;  
int y = 5;  
String coordinates = "(" + x + "," + y + ")"; // "(3,5)"
```

Strings

Vorsicht, Evaluationsreihenfolge beachten:

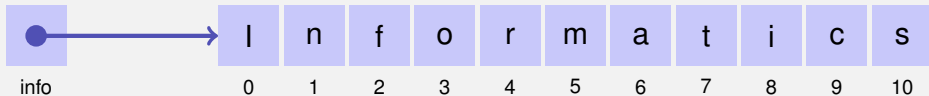
```
int x = 3;  
int y = 5;  
String s1 = x+y+"X"; // s1 = "8X"  
String s2 = "X"+x+y+""; // s2 = "X35"
```


Characters

Elemente eines Strings können auch per Index gelesen (nicht aber geschrieben) werden:

```
String info = "Informatics";  
char c = info.charAt(3); // c = 'o'
```

Strings sind auch Referenzen!



Stringvergleiche

Der Vergleich mit '==' vergleicht Referenzen, nicht den Inhalt!

```
String n1 = In.readWord();
String n2 = In.readWord();
if (n1 == n2){
    Out.println(n1 + "==" +n2);
} else {
    Out.println(n1 + "!=" + n2);
}
```

Stringvergleiche

Der Vergleich mit '==' vergleicht Referenzen, nicht den Inhalt!

```
String n1 = In.readWord();  
String n2 = In.readWord();  
if (n1 == n2){  
    Out.println(n1 + "==" +n2);  
} else {  
    Out.println(n1 + "!=" + n2);  
}
```

Eingabe: Info Info

Ausgabe: Info != Info

Stringvergleiche

Der Vergleich mit 'equals' vergleicht den Inhalt!⁷

```
String n1 = In.readWord();
String n2 = In.readWord();
if (n1.equals(n2)){
    Out.println(n1 + " equals " +n2);
} else {
    Out.println(n1 + " not equals " + n2);
}
```

⁷Bei Arrays geht das nicht!

Stringvergleiche

Der Vergleich mit 'equals' vergleicht den Inhalt!⁷

```
String n1 = In.readWord();  
String n2 = In.readWord();  
if (n1.equals(n2)){  
    Out.println(n1 + " equals " +n2);  
} else {  
    Out.println(n1 + " not equals " + n2)  
}
```

Eingabe: Info Info

Ausgabe: Info equals Info

⁷Bei Arrays geht das nicht!