Educational Objectives

- You understand the advantage of defensive programming - “fail fast”
- You know how to use assertions.
- You undrestand the concept of *arrays* and you can create and use them.
- You develop an understanding of *reference semantics*.
- You know the basics of Strings and multi-dimensional arrays.

9. Defensive Programming

Programming with Assertions

Sources of Errors

- Errors that the compiler can find:
  syntactical and some semantical errors
- Errors that the compiler cannot find:
  runtime errors (always semantical)

Avoid Sources of Bugs

1. Exact knowledge of the wanted program behavior

   ≫ It’s not a bug, it’s a feature! ≪

2. Check at many places in the code if the program is still on track
3. Question the (seemingly) obvious, there could be a typo in the code

Against Runtime Errors: *Assertions*

```
assert expr : msg;
```

- halts the program if the boolean expression `expr` is false
- print the message `msg` in case the assertion doesn't hold (optional)
- gets activated by the flag `-ea` upon startup of the program
- In Code Expert the flag is activated in the playground and in future exercises.

Assertions for the $gcd(x, y)$

Check if the program is on track . . .

```
// Input x and y
Out.print("x =? ");
x = In.readInt();
Out.print("y =? ");
y = In.readInt();

// Check validity of inputs
assert x > 0 && y > 0 : "Invalid input: x and y must be positive!";

... // Compute gcd(x,y), store result in variable a
```

Input arguments for calculation

Precondition for the ongoing computation

Assertions for the $gcd(x, y)$

... and question the obvious! . . .

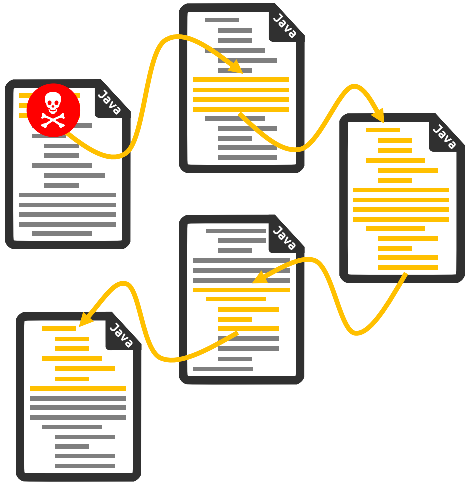
```
...
assert x > 0 && y > 0 : "Invalid input: x and y must be positive!";

... // Compute gcd(x,y), store result in variable a

assert a >= 1;
assert x % a == 0 && y % a == 0;
for (int i = a+1; i <= x && i <= y; ++i)
  assert !(x % i == 0 && y % i == 0);
```

Precondition for the ongoing computation

Properties of the gcd

Fail-Fast with Assertions

- Real software: many Java files, complex control flow
- Errors surface late(r) → impedes error localisation
- Assertions: Detect errors early



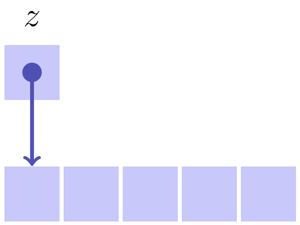
10. Java Arrays and Strings

Allocation, references, element access, multidimensional arrays, strings, string comparision

Arrays

Declare array variables: `int[] z;`

Create an array: `z = new int[5];`

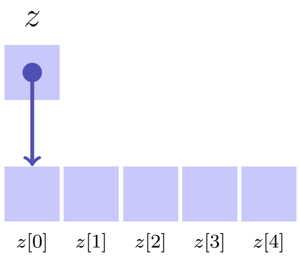


z is a *reference* to the array data,

- but only after the assignment to the created data
- otherwise it points to nowhere: `null`.

Arrays

```
int[] z;

z = new int[5];
```



Elements are indexed. The first index is 0 and the last index is array size -1

Element access: `name[index]`

Arrays are dynamic objects

Arrays are always created dynamically

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

Size of an array can be set at runtime
But an array doesn't grow automatically!

Arrays are not primitiv

Arrays carry *metadata*:

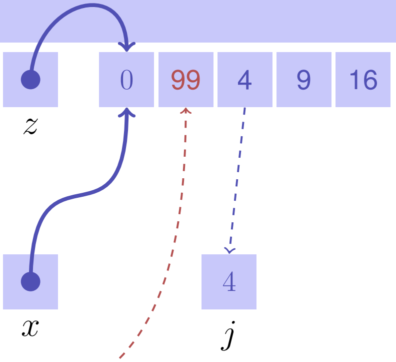
```
int sq = new int[7];
for (int i = 0; i< sq.length; ++i){
    sq[i] = i * i;
}
sq[8] = 64;⟵ java.lang.ArrayIndexOutOfBoundsException!
```

... even over method boundaries (next time):

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

Array assignments

```
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;
```



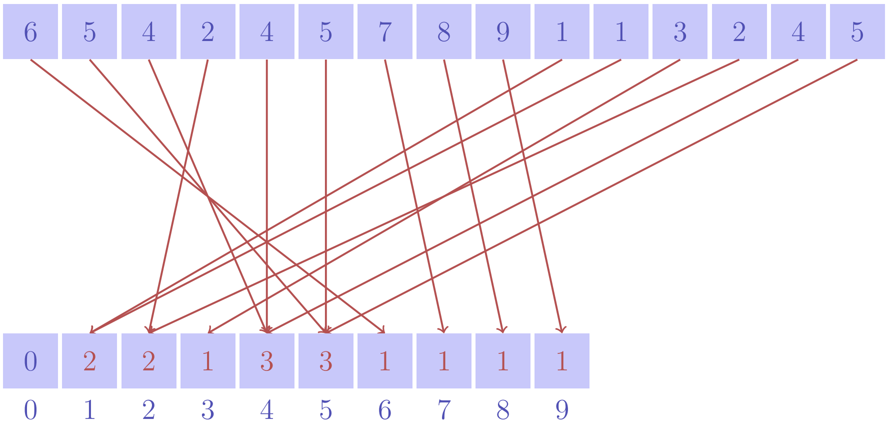
When assigning arrays, *references are being copied*, not data!

Example

Given an (unsorted) array x containing numbers from the range $[0,\ldots,9]$.

Task: Write an efficient program that output for each number, how many times it occurs in x.

Idea



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

0 2 2 1 3 3 1 1 1 1

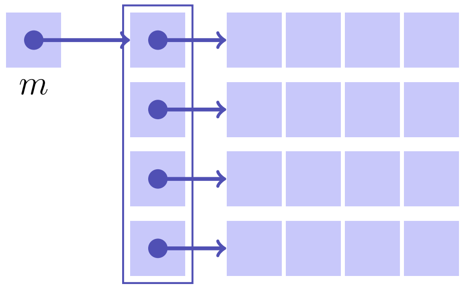
0 1 2 3 4 5 6 7 8 9

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]);
        }
    }
}
```

Multidimensional arrays

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



Multidimensional 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;
        }
    }
}
```

Multidimensional arrays

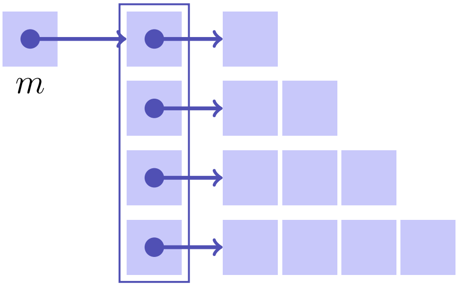
A two-dimensional array is an array of references to a one-dimensional array. Thus, the following is possible:

```
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();
}
```

Multidimensional arrays

Even this is possible:

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



Array comparisons

Attention (again): Arrays are references!

```
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!
```

For the experts:

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

Attention when using `Arrays.equals` on multidimensional arrays! (What is compared?)

Strings

String: an object that stores a character array.

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

Strings

Mind the evaluation order:

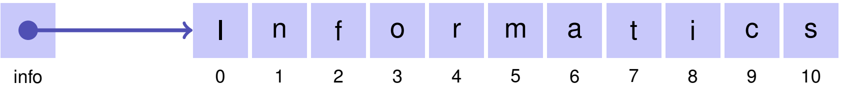
```
int x = 3;
int y = 5;
String s1 = x+y+"X";  // s1 = "8X"
String s2 = "X"+x+y+""; // s2 = "X35"
```

Characters

Elements of a string can be accessed by index (but not replaced)

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

Strings are references as well!



info

I	n	f	o	r	m	a	t	i	c	s
0	1	2	3	4	5	6	7	8	9	10

String comparisons

The comparison with '==' compars references, not content!

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

Input: Info Info

Output: Info != Info

String comparisons

The comparison with 'equals' compares the content![5]

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

Input: Info Info

Output: Info equals Info

[5] This doesn't apply to arrays