

# CODERS.BAY

## EXERCISES

#3	spaghetti code	DONE
#4	calculator	DONE
#5	arrays	teilweise - Feedback von Lukas offen
	recap exercise	DONE
#6	sorting exercise	
	multidimensional arrays	<a href="https://gist.github.com/x21L/e9b7200cae8ecac02cf4d6729426b9b2">https://gist.github.com/x21L/e9b7200cae8ecac02cf4d6729426b9b2</a>
#7	Game of Life	
#8	Bebe-Sprache	teilweise - Feedback Lasi offen
#9	Căsar ciphre	ok - Feedback Lasi offen
	Soul Seller	versucht - Hilfe von Julia offen

<b>Basics of Java</b>	<b>2</b>
Java Compilation process	3
Setup	3
Data types	4
Variables (declaring and initialising)	8
Arithmetic/ mathematic operations	10
Relational operations	11
Logical operations	12
Boolean (calculating)	13
FOR - loop	16
DO WHILE	17
<b>FOR-EACH LOOP (enhanced for-loop)</b>	<b>18</b>

<b>Decision making</b>	<b>19</b>
IF	19
IF ELSE	19
NESTED IF	20
IF ELSE IF LADDER	20
SWITCH	21
JUMP	24
<b>Methods</b>	<b>27</b>
Structure of a method	27
Calling a method	27
Return types of a method	28
<b>Typumwandlung</b>	<b>30</b>
expliziter Cast	30
small to large data type	30
large to small data type	30
impliziter Cast	30
<b>Notes/ Tricks:</b>	<b>32</b>
<b>Arrays erklärt</b>	<b>32</b>
Arrays	32
Array anlegen bzw. deklarieren	32
Array mit Werten füllen	32
Auf Elemente eines Arrays zugreifen	33
<b>Foreach-Schleife - Anwendung bei Arrays</b>	<b>33</b>
2D arrays	34
Dynamic Arrays	34
<b>Sortierungen</b>	<b>34</b>
-----	36
Creating objects in java	36

JAVA

## Basics of Java

There are

- **script languages:** runs through an interpreter and runs by the browser (html, JavaScript, PHP, Shell Script)
- **compiled languages:** runs on your hardware, through a compiler and reads “machine language 0111010”. (C, C++, Java, Go)

**Java is independent from the platform it was created on.  
Compile once, run everywhere.**

## Java Compilation process

1. Java Code (Java)
2. JAVAC (Compiler) → human readable code in the editor, that is transformed to 0 and 1 by the compiler
3. Byte Code (.class) → needed to run the 0 and 1
4. Java Virtual Machine (JVM)
  - a. Windows, Linux, Unix (MacOS)

How it works in the terminal window:

1. enter “javac **HelloWorld**” → compiling the code
2. enter “java **HelloWorld**” → running the code

## Java Versions

- JDK . . . Java Development Kit ! for developing JAVA applications.
- JRE . . . Java Runtime Environment ! for running JAVA applications.
- JDK Versions > 8.

For programming you need an editor and the installation of the Java language in order to run it.

## Setup

### AdoptOpen JDK installieren

<https://adoptopenjdk.net/?variant=openjdk14&jvmVariant=hotspot>

Official documentation: <https://docs.oracle.com/javase/9/tools/javac.htm#JSWOR627>

Install **editor GEANY** <https://geany.org/download/releases/>

Other, more professional editor: **IntelliJ** (can be linked to Github)

 <https://www.java-tutorial.org/hello-world.html>

## Coding Standards

## Comments and styles

```
public class Example{
    // rectangle
    private double height = 3.4;
    private double width = 2.5;

    // PascalCasing & lowerCamelCase
    private String workshopName= "Roboterbau";

    // constant
    private static final int WEEKDAYS = 7;
}
```

- Classes and types **Pascal Casing**.
- **camelCasing** for variables
- **CONSTANTS** are upper case.
- Do not use underscores
- Use tabs not whitespaces.
- comments `/* */`
- `\t` --> Tab Stop
- `\n` --> new line
- not to be confused with `/**` → this is a java documentation

Google Java Standard: <https://google.github.io/styleguide/javaguide.html>

Programming jargon: <https://blog.codinghorror.com/new-programming-jargon/>

## TIPP

👉 Collect cool code, tag and sort it: **Github Gist** <https://gist.github.com/>

Use client in order to make it easier to use: **Lepton** <https://hackjutsu.com/Lepton/>

Other markdown editor: **Typora** <https://typora.io/>

- **Libraries** - for later use, when we know the basics
- **Maven** - the most famous JAVA library for professional use  
<https://mvnrepository.com/>
  - e.g. libraries from Amazon, Google (guava),...
- Famous Dijkstra Algorithmus <https://de.wikipedia.org/wiki/Dijkstra-Algorithmus>

## Data types

Name	Deutscher Name	Beispiel
boolean	Wahrheitswert	<code>boolean hungrig = true;</code> <code>boolean muede = false;</code>
int	Ganze Zahlen	<code>int zweit = 2;</code> <code>int wenig = -100;</code>
double	Kommazahlen	<code>double zahl = 1.2;</code>
char	Buchstabe/Zeichen	<code>char buchstabe = 'a';</code>
String	Zeichenketten	<code>String name = "Max";</code>

double      Kommazahl (8 bytes)      `double preis = 19.99;`  
float      Kommazahl (4 bytes)      `float f = 1.4f;`  
→ Avoid trouble, use double!

## Variablentypen

<u>String</u>	= Zeichenketten
<u>boolean</u>	= true / false
<u>char</u>	= 1x Zeichen / Zahl
<u>byte</u>	= GZ
<u>short</u>	= GZ
<u>int</u>	= GZ
<u>long</u>	= GZ
<u>float</u>	= KZ
<u>double</u>	= KZ

## Relationale Operatoren

Operatoren:

==

überprüft Werte auf Gleichheit

!=

überprüft Werte auf Ungleich

>

Wert größer als andere

<

Wert kleiner als andere

>=

Wert größer od. gleich

<=

Wert kleiner od. gleich



## Logische Operatoren

- $\&\&$  erhalten true wenn beide Werte wahr sind.
- $\|\|$  true wenn mindestens eine Bedingung erfüllt ist
- $!$  true wenn Bedingung nicht erfüllt ist.
- $\wedge$  Wenn genau eine Bedingung erfüllt ist true

## Ausgabe

```
System.out.println("Hello");
```

## Random Zahlen

```
double number = 10 + (0 ~ 1Math.random() * 20);
```

```
int number = 1 + (int)(Math.random() * 6);
```

## Auf/Ab runden

Math.ceil() → aufrunden

Math.floor() → abrunden

## Variables (declaring and initialising)

int	numA	=	9	;
string	helloWorld	=	"Hello World"	;
double	price	=	19.99	;
boolean	single	=	false	;
<b>datatype</b>	<b>unique name in camelCase</b>		<b>value, that matches the datatype</b>	
<b>STEP 1:</b> <u>DECLARING</u> the datatype			<b>STEP 2:</b> <u>INITIALISING</u> the value	

e.g.

```
int numA = 9;
```

```
int numA = 10;
```

 → Values can be overwritten, the data type has to be the same.

If the value is not initialised, then the value is NULL.

- also possible: `int numA = 3, numB = 4;`      initializing+declaring of more variables
- also possible: `int numA, numB;`      initializing first, declaring later  
                  `numA = 3;`  
                  `numB = 4;`



**MAIN**

① `double netPrice = 24,16;`  
`int pieces = 6;`

③ `double totalCostVal = totalV(netPrice, pieces);` → Zwischenwert  
 ④ `System.out.println(totalCostVal);` ← neue Variable m. halbschritt erfüllt!

② `private static double totalV(double netPrice, int pieces) {`  
`double vat = ((netPrice + pieces) / 100) * 20;`  
`return vat;` → erfüllt totalV!

① Variablen deklarieren u. initialisieren  
 ② In ausgelagerten Methode m. Variablen rechnen  
 Wert in Zwischenvariable "vat" geben u. ausgeben.  
 ③ Wert aus Methode in Variable geben!  
 ④ Mit Variable arbeiten.

oder neue Zwischenwerte

### Beispiel: kaufhaus

1. Die Methode kaufhaus wird mit (**int stockwerkwahl**) initialisiert.
2. Die Variable wird mit **int stockwerk = stockwerkwahl** deklariert und hat damit einen Wert.

```

public class Switch {

    public static void main(String[] args) {
        kaufhaus( stockwerkWahl: 3);
    }

    private static int kaufhaus(int stockwerkWahl) {
        int stockwerk = stockwerkWahl;
        if (stockwerk == 1) {
            System.out.println("Kinderabteilung");
        } else if (stockwerk == 2) {
            System.out.println("Herrenschuhe");
        } else if (stockwerk == 3) {
            System.out.println("Damenunterwäsche");
        } else {
            System.out.println("Hilfe!");
        }
        return stockwerk;
    }
}

```

▶▶ Useful links for learning:

[https://www.w3schools.com/java/java\\_data\\_types.asp](https://www.w3schools.com/java/java_data_types.asp)

<https://studyflix.de/informatik/primitive-datentypen-215>

## Arithmetic/ mathematic operations

“normal” calculations and some short forms

```

public class Calc {
    public static void main (String[] args) {

        int numA = 10;
        int numB = 3;

        double numC = 20;
        double numD = 6;

        int numE = 2;
        int numF = 8;

        System.out.println("Simple arithmetic calculations");
        System.out.println(numA + numB);
        System.out.println(numA / numB);
        System.out.println(numC / numD);
        System.out.println(100 + numA);
        System.out.println(numB + numC);
        System.out.println(numB / numD);

        System.out.println("Short forms");
        numE++;
        System.out.println(numE);
        System.out.println(numB *= 3);
        numF--;
        System.out.println(numF);
    }
}

```

```

Simple arithmetic calculations
13
3
3.3333333333333335
110
23.0
0.5
Short forms
3
9
7

```

## Relational operations

compare two values, the result is always true or false

== equals  
 != not equal  
 > bigger  
 < smaller  
 >= bigger equal  
 <= smaller equal

e.g.

```

int numA = 10;
int numB = 3;
boolean result;

```

```

System.out.println("Relational calculations");
result = numA==numB;
System.out.println(result);

```

## Logical operations

Calculations that return a boolean as result.

Operator	Name	Type	Description
!	not	unary	Returns true if the operand to the right evaluates to false.  Returns false if the operand to the right is true.
&	and	binary	Returns true if both operands are true.
	or	binary	Returns true if one of the operands is true
^	xor	binary	Returns true if one — and only one — of the operands evaluates to true.  Returns false if both operands evaluate to true or if both operands evaluate to false.
&&	conditional and	binary	<u>Returns true if both operands are true</u> , but if the operand on the left returns false, it returns false without evaluating the operand on the right.
	conditional or	binary	<u>Returns true if one of the operands is true</u> , but if the operand on the left returns true, it returns true without evaluating the operand on the right.

e.g. You can drink alcohol at 16 in Austria and at the age of 23 in America.  
(age>=16&&location=="Austria") || (age>=23&&location=="America")

e.g.

Die Bedingung ist: Du darfst an einem Programmierwettbewerb nur teilnehmen, wenn du mehr als 3 Jahre Programmiererfahrung hast, genau 20 Jahre alt bist und Englisch sprichst. Oder wenn du Glück hast und der Wettbewerb fällt auf einen Freitag, dann darfst du auch daran teilnehmen. Falls es aber regnen sollte während du die bisherige Bedingung erfüllt hast, darfst du nicht am Wettbewerb teilnehmen. Auch wenn es nicht regnet und du die bisherige Bedingung nicht erfüllt hast, darfst du nicht teilnehmen. Weiterhin muss die Negation von false gleich true sein, damit die bisherige Bedingung erfüllt wird. Du musst den Ausdruck einfach nur Stück für Stück aufbauen und die einzelnen Bausteine zusammenfügen.

```
((experience>3&&age==20&&language=="EN") || weekday=="Friday" ^ rain) && !false
```

Conjunction		Disjunction	
A	B	A	B
T	T	T	T
T	F	T	F
F	T	F	T
F	F	F	F

## Boolean (calculating)

int → echte Berechnungen mit Zahlen als Ergebnis  
 Ergebnis: float oder double

byte = nur Ganze Zahl, ein Wert um boolean zu berechnen

**boolean** → **hat nur 2 Werte (true oder false)**

▶▶ Calculating tipps: <https://www.wolframalpha.com/>





# Scanner

① `import java.util.Scanner;`

→ Scanner Bibliothek holen

METHODE

② `Scanner scanner = new Scanner(System.in);`  
`System.out.println("Etwas something.");`

→ Scanner "aktivieren"

③ `String input = scanner.nextLine();`  
`int intInput = Integer.parseInt (scanner.nextLine());`  
`globalVariable = Double.parseDouble(scanner.nextLine());`

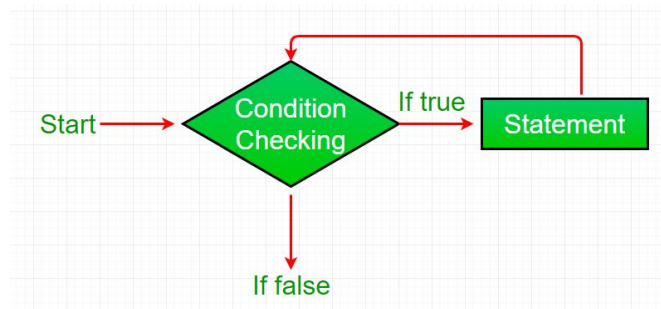
→ input in Variable  
geben und gleich  
lesen / umwandeln.

`int x = (int) scanner.nextDouble();`  
z.B.: input: 30,99 (wird nextDouble)  
↳ int: 30

→ Input von anderem Typ  
& gleichzeitig casten.

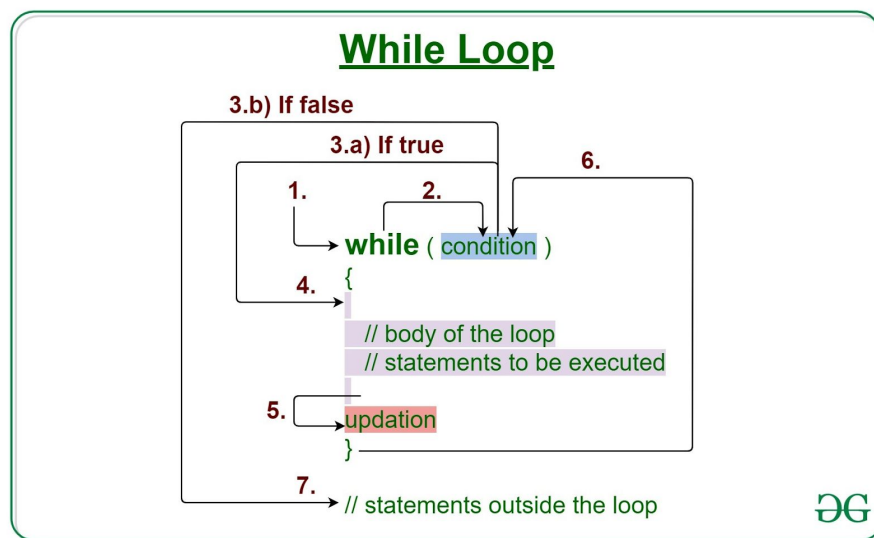
# WHILE - loop

**Kopfgesteuert** → checks at the head, if the code has to be executed



## How does a While loop execute?

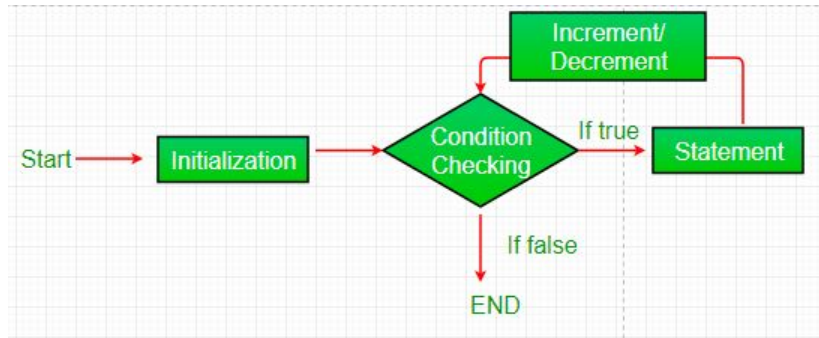
1. Control falls into the while loop.
2. The flow jumps to Condition
3. Condition is tested.
  1. If Condition yields true, the flow goes into the Body.
  2. If Condition yields false, the flow goes outside the loop
4. The statements inside the body of the loop get executed.
5. Updation takes place.
6. Control flows back to Step 2.
7. The do-while loop has ended and the flow has gone outside.



# FOR - loop

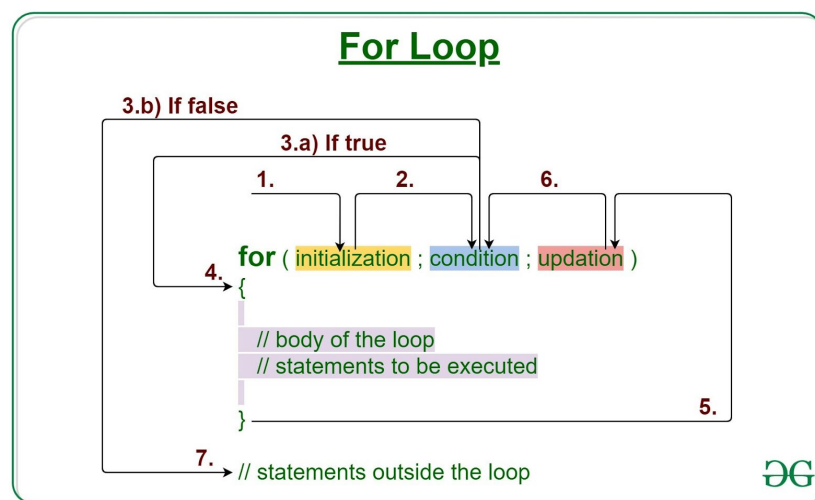
## Kopfgesteuert

Condition, stop and counter are IN the loop function.



## How does a For loop executes?

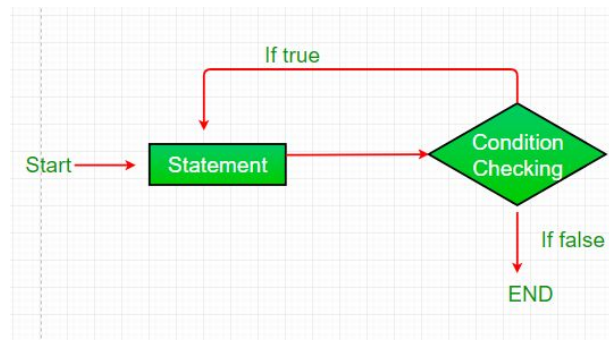
1. Control falls into the for loop. Initialization is done
2. The flow jumps to Condition
3. Condition is tested.
  1. If Condition yields true, the flow goes into the Body
  2. If Condition yields false, the flow goes outside the loop
4. The statements inside the body of the loop get executed.
5. The flow goes to the Updation
6. Updation takes place and the flow goes to Step 3 again
7. The for loop has ended and the flow has gone outside.



# DO WHILE

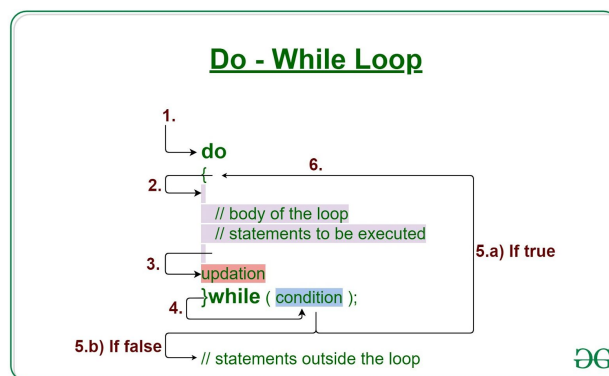
**Fußgesteuert** → checks the whole code

e.g. Check every time a user enters something. Do something and then check, if the condition is met. E.g. "Do you want to continue?"



## How does a do-While loop executes?

1. Control falls into the do-while loop.
2. The statements inside the body of the loop get executed.
3. Updation takes place.
4. The flow jumps to Condition
5. Condition is tested.
  1. If Condition yields true, goto Step 6.
  2. If Condition yields false, the flow goes outside the loop
6. Flow goes back to Step 2.



If I need at least one run of the loop I need FUSSGESTEUERT. If not, then KOPFGESTEUERT.

If I count upwards, then FOR is clearer and more practical.

## FOR-EACH LOOP (enhanced for-loop)

```
// Java program to illustrate enhanced for loop

public class enhancedforloop {

    public static void main(String args[])
    {
        String array[] = { "Ron", "Harry", "Hermoine" };

        // enhanced for loop
        for (String x : array) {
            System.out.println(x);
        }

        /* for loop for same function
        for (int i = 0; i < array.length; i++)
        {
            System.out.println(array[i]);
        }
        */
    }
}
```



# Decision making

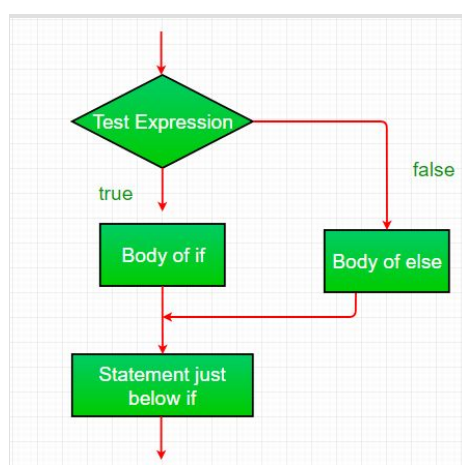
## IF

```
if(condition)
    statement1;
    statement2;
```

```
// Here if the condition is true, if block
// will consider only statement1 to be inside
// its block.
```

## IF ELSE

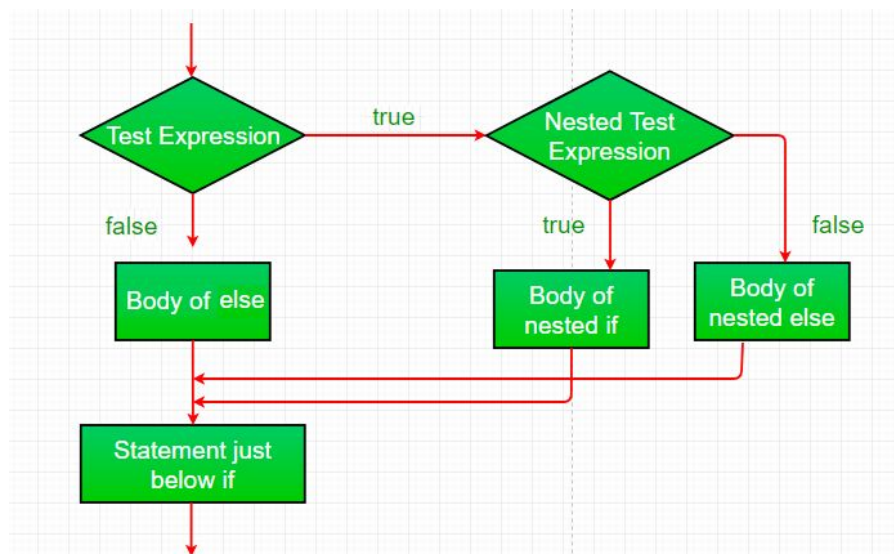
```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```



## NESTED IF

This is an if-statement inside an if statement

```
if (condition1)
{
    // Executes when condition1 is true
    if (condition2)
    {
        // Executes when condition2 is true
    }
}
```



## IF ELSE IF LADDER

The if statements are executed from top down. **As soon as one condition is true, the statement is executed and all other conditions bypassed.**

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
```

```
statement;
```

Example:

```
// Java program to illustrate if-else-if ladder
class ifelseifDemo
{
    public static void main(String args[])
    {
        int i = 20;

        if (i == 10)
            System.out.println("i is 10");
        else if (i == 15)
            System.out.println("i is 15");
        else if (i == 20)
            System.out.println("i is 20");
        else
            System.out.println("i is not present");
    }
}
```

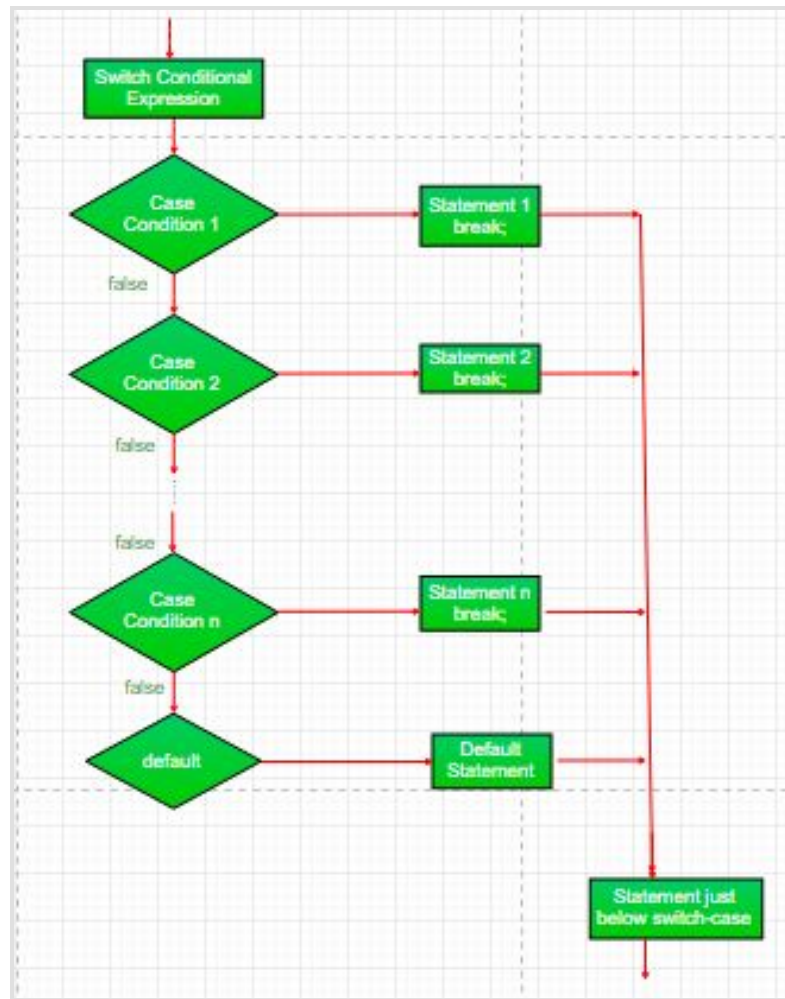
## SWITCH

Solution to show multiple branches of options.

- Duplicate case values are not allowed.
- The default statement is optional.
- The break statement is used inside the switch to terminate a statement sequence.
- The break statement is optional. If omitted, execution will continue on into the next case

```
switch (expression)
{
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    .
    .
    case valueN:
        statementN;
        break;
```

```
default:  
    statementDefault;  
}
```



```
// Java program to illustrate switch-case
class SwitchCaseDemo
{
    public static void main(String args[])
    {
        int i = 9;
        switch (i)
        {
            case 0:
                System.out.println("i is zero.");
                break;
            case 1:
                System.out.println("i is one.");
                break;
            case 2:
                System.out.println("i is two.");
                break;
            default:
                System.out.println("i is greater than 2.");
        }
    }
}
```



# JUMP

The three statements **break**, **continue** and **return** transfer the control to other parts of the program.

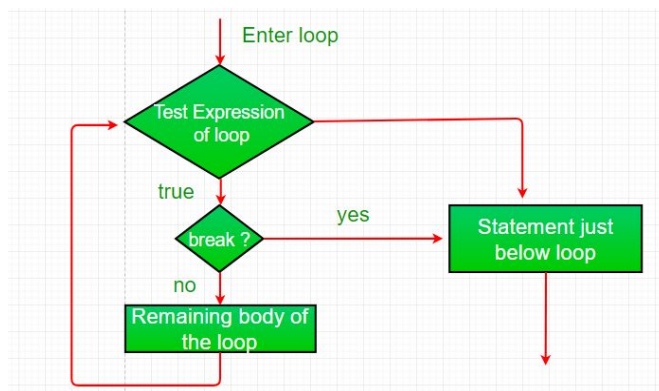
## BREAK

In Java, break is majorly used for:

- Terminate a sequence in a switch statement (discussed above).
- To exit a loop.
- Used as a “civilized” form of goto.

Note: Break, when used inside a set of nested loops, will only break out of the innermost loop.

### 1. Using break to exit a loop



```
// Java program to illustrate using
// break to exit a loop
class BreakLoopDemo
{
    public static void main(String args[])
    {
        // Initially loop is set to run from 0-9
        for (int i = 0; i < 10; i++)
        {
            // terminate loop when i is 5.
            if (i == 5)
                break;

            System.out.println("i: " + i);
        }
        System.out.println("Loop complete.");
    }
}
```

### 2. Using break to as a form of “go to”

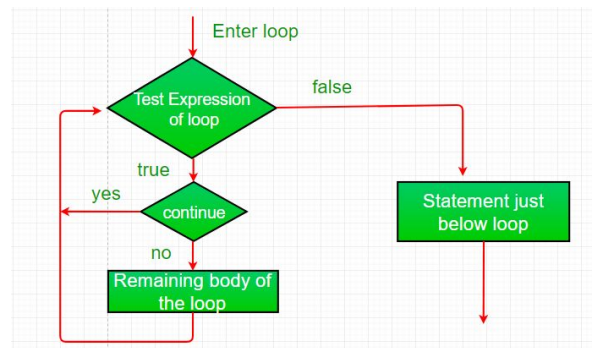
Java does not have a goto statement because it provides a way to branch in an arbitrary and unstructured manner. Java uses **label**. A Label is use to identifies a block of code. Now, break statement can be use to jump out of target block.

<https://www.geeksforgeeks.org/decision-making-javaif-else-switch-break-continue-jump/>

## CONTINUE

Force an early iteration of the loop

Add additional conditions in the loop that are tested and allow the program to continue.




```
// Java program to illustrate using
// continue in an if statement
class ContinueDemo
{
    public static void main(String args[])
    {
        for (int i = 0; i < 10; i++)
        {
            // If the number is even
            // skip and continue
            if (i%2 == 0)
                continue;

            // If number is odd, print it
            System.out.print(i + " ");
        }
    }
}
```

## RETURN

To explicitly return from a method back to the caller of a method.



```
// Java program to illustrate using return
class Return
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println("Before the return.");

        if (t)
            return;

        // Compiler will bypass every statement
        // after return
        System.out.println("This won't execute.");
    }
}
```

Output:

Before the return.

Methods - In Java there are only methods, methods are bound to objects

Function - Is the same as methods, but not bound to objects

# Methods

A method is a block of code which only runs when it is called. You can pass data, known as parameters, into a method. Methods are used to perform certain actions, and they are also known as functions.

**Why use methods?** To reuse code: define the code once, and use it many times.

## Structure of a method

```
public class MyClass {  
    static void myMethod() {  
        // code to be executed  
    }  
}
```

→ **myMethod()** → the name of the method

## Calling a method

A Method can be called multiple times and also within other methods.

```
public class MyClass {  
    static void myMethod() {  
        System.out.println("I just got executed!");  
    }  
}
```

```
public static void main(String[] args) {  
    myMethod();  
    myMethod();  
    myMethod();  
}
```

```
// I just got executed!  
// I just got executed!
```

```
// I just got executed!
```

## Return types of a method

If you want the method to return a value, you can use a primitive data type (such as `int`, `char`, etc.) instead of `void`, and use the `return` keyword inside the method.

### Normal return, nested in a method

```
public class MyClass {  
    static int myMethod(int x) {  
        return 5 + x;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(myMethod(3));  
    }  
}  
  
// Outputs 8 (5 + 3)
```

### Two or more methods are returned at once

```
public class MyClass {  
    static int myMethod(int x, int y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(myMethod(5, 3));  
    }  
}  
  
// Outputs 8 (5 + 3)
```

### The return value can be stored in a new variable

```
public class MyClass {  
    static int myMethod(int x, int y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        int z = myMethod(5, 3); → new variable “z” with the return value  
        System.out.println(z);  
    }  
}
```



```
}  
}
```

```
// Outputs 8 (5 + 3)
```

## Typumwandlung

<https://www.java-tutorial.org/typecasting.html>

### expliziter Cast

#### small to large data type

double toBeCasted = 12.235;

**Typ (Eingabe-Variablenname);**

```
public class MyClass {  
    public static void main(String[] args) {  
        int myInt = 9;  
        double myDouble = myInt; // Automatic casting: int to double  
  
        System.out.println(myInt);    // Outputs 9  
        System.out.println(myDouble); // Outputs 9.0  
    }  
}
```

#### large to small data type

int castToInt = (int) toBeCasted;

**Zieltyp Ziel-Variablenname = (Zieltyp) Eingabe-Variablenname;**

```
public class MyClass {  
    public static void main(String[] args) {  
        double myDouble = 9.78;  
        int myInt = (int) myDouble; // Manual casting: double to int
```

```

    System.out.println(myDouble);    // Outputs 9.78
    System.out.println(myInt);       // Outputs 9
}
}

```

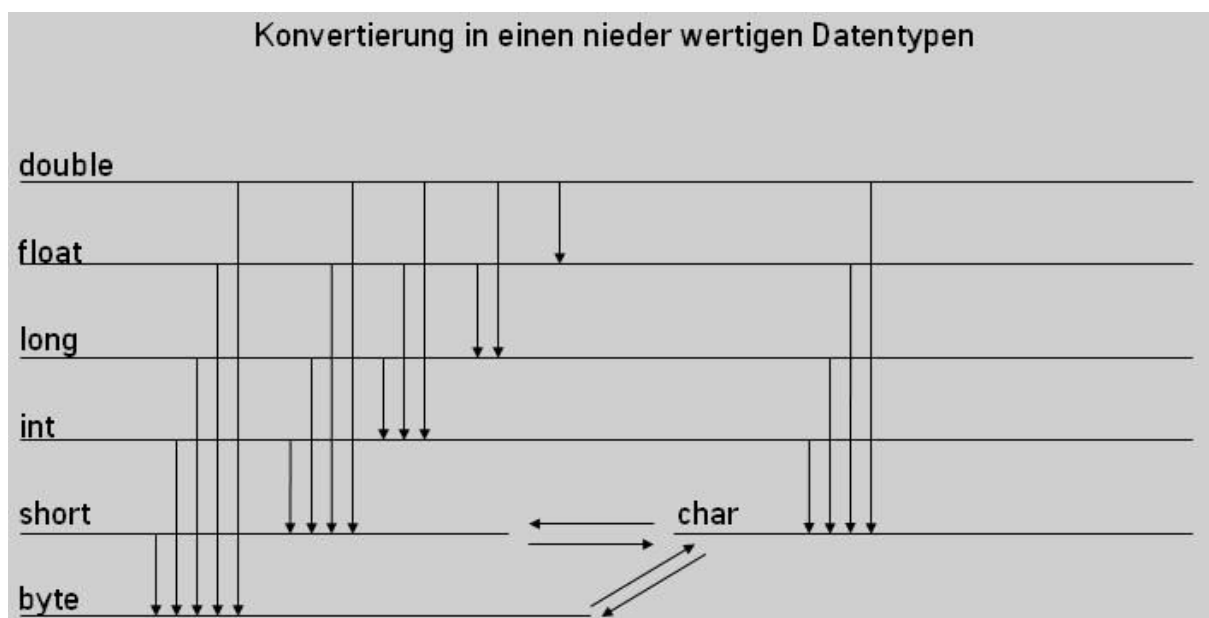
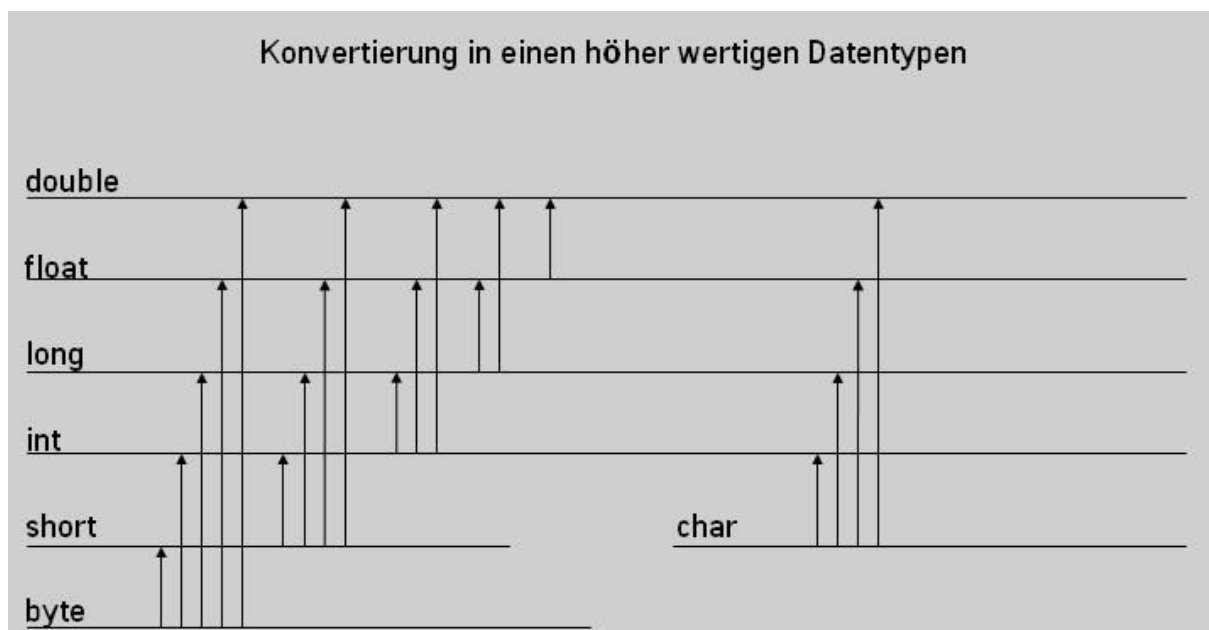
## impliziter Cast

```

int value1 = 12;
double value2;
value1 = value2; → 12.0

```

**Ziel-Variablenname = Eingabe-Variablenname;**





## Notes/ Tricks:

- Strings vergleichen mit s.equals
- SWITCH-Anweisungen: Verwendung für Bedienelemente
- String “ “
- Char ‘ ‘
- Klassen umbenennen in IntelliJ mit Refactor!

## Arrays erklärt

An **Array** is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. The length is fixed after creation.

A **dynamic array** is a variable-size list data structure that allows elements to be added or removed. Dynamic arrays overcome a limit of static arrays, which have a fixed capacity that needs to be specified at allocation.

## Arrays

### 1. Array anlegen bzw. deklarieren

```
Datentyp[] arrayName;  
arrayName = new Datentyp[Anzahl]
```

```
int[] testArray;  
testArray = new int[10]
```

oder in einem Schritt

```
int[] testArray = new int[10];
```

### 2. Array mit Werten füllen

```
arrayName[indexPosition] = Wert;
```

```
testArray[0] = 52;  
testArray[1] = 2;  
testArray[2] = 256;  
testArray[3] = 18;
```

Gleich beim Anlegen des Arrays Werte vergeben:

```
int[] testArray = {52, 2, 256, 18};
```

TIPP: Für Initialisierung einer bestimmten Datenreihe ist auch eine for-Schleife möglich.

Inhalte von Arrays einfach überschreiben:

```
testArray[0] = 99;
```

### 3. Auf Elemente eines Arrays zugreifen

Ausgabe der Position im Index →

```
System.out.println(testArray[1]);
```

Alle Variablen ausgeben

```
for(int=0; i<testArray.length; i++) {  
    System.out.println(testArray[i]);  
}
```

## Foreach-Schleife - Anwendung bei Arrays

Auf Werte eines Arrays zugreifen. Foreach ist kürzer als eine for-Schleife.

Beispiel:

**//Deklarieren von Array mit 5 Werten**

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

**//Initialisieren von Array mit den Zahlen 1-5 mit for-Schleife**

```
for (int i = 0; i < reihe.length; i++){  
    reihe[i] = i+1;  
}
```

```
for (Datentyp Variable : Arrayname){  
    xxx  
}
```

**//Array mit foreach-Schleife ausgeben lassen:**

```
for (int i : reihe) {  
    System.out.println(i);  
}
```

**Beispiel:**

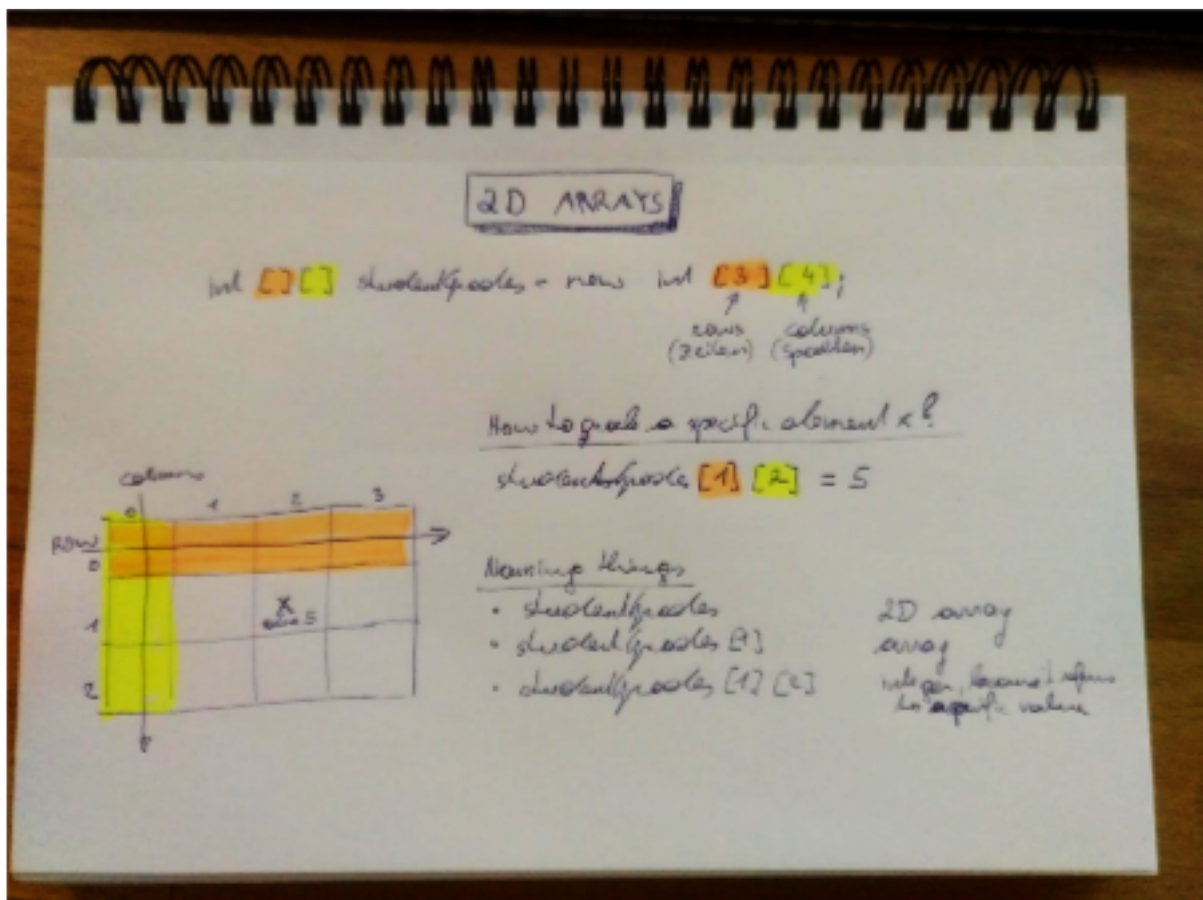
```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
for (String i : cars) {
```

```
System.out.println(i);
```

```
}
```

The example above can be read like this: for each **String** element (called *i* - as in index) in cars, print out the value of *i*.

## 2D arrays



xxx

## Dynamic Arrays

später



# Sortierungen

## Time complexity

- Beste Laufzeitkomplexität: 1 oder logarithmisch  $y = x * \log(x)$
- Radix sort: <https://studyflix.de/informatik/radix-sort-1408>

## Space complexity

- Beste Speicherplatzbelegung

Verschiedene Sortieralgorithmen

[The Sound of Sorting - "Audibilization" and Visualization of Sorting Algorithms](#)

<https://www.youtube.com/c/udiproduct/videos>

Bei array-sort wird schon der beste Algorithmus ausgewählt. Zur Übung schreiben wir das aber selbst.

Bubble Sort

<https://studyflix.de/informatik/bubblesort-1325>

<http://faculty.cs.niu.edu/~hutchins/csci241/sorting.htm>

[https://www.w3schools.com/js/js\\_array\\_sort.asp](https://www.w3schools.com/js/js_array_sort.asp)

Videos that explain the sorting algorithms:

<https://www.youtube.com/c/udiproduct/videos>

<https://www.youtube.com/watch?v=92WHN-pAFCs>

Java programming cheat sheet:

<https://introcs.cs.princeton.edu/java/11cheatsheet/>



-----

TO DO:

<https://www.w3schools.com/java/exercise.asp>

## Creating objects in java

<https://studyflix.de/informatik/objekte-225>

<https://www.programmierenlernenhq.de/klassen-und-objekte-in-java/>

<https://www.programmierenlernenhq.de/grundlagen-der-objektorientierten-programmierung-in-java/>

## Solving problems

Vom Problem zum Algorithmus – Problemstellungen verstehen und übersetzen  
Problemstellungen zu verstehen ist nicht immer einfach, das muss man lernen. Das hat nicht unbedingt damit zu tun, dass Angaben auf Übungszetteln oder bei Klausuren manchmal schwer zu verstehen sind, weil sie Fachbegriffe enthalten, die Ihnen noch nicht so vertraut sind. Im Gegenteil, in der Praxis sind Sie eher mit einer Situation wie der folgenden konfrontiert:

Ein(e) Nicht-InformatikerIn hat ein bestimmtes Problem, das sich durch ein einfaches Programm leicht lösen ließe. Sie sollen das Problem als ProgrammiererIn lösen, haben jedoch in diesem Bereich noch nicht gearbeitet und dadurch keine Erfahrung gesammelt. Der/Die AuftraggeberIn erklärt Ihnen das Problem und die Anforderungen an das Programm. Auch wenn Sie auf Anhieb verstehen, was dieses neue Programm leisten und wie das Ergebnis aussehen soll, müssen Sie die Problemstellung doch genau analysieren und in „Ihre Sprache“ (Algorithmen etc.) übersetzen, um nicht nur eine lauffähige, sondern auch eine wirklich gute Anwendung zu programmieren.

Dies gelingt am besten, wenn Sie sich viele Fragen stellen:

- Wie soll das Ergebnis aussehen?
- Welche Aufgaben hat das Programm?
- In welcher Reihenfolge sollen sie durchgeführt werden?
- Welche Einzelschritte sind notwendig?
- Welche Daten sollen eingegeben und welche ausgegeben werden?
- Welche Variablen und Konstanten, welche Datentypen werden benötigt?

- Wo, wann und wie werden die Daten eingegeben oder aus einer externen Datei eingelesen?
- Welchen Daten gehören zusammen?
- Wo verwende ich einfache Datentypen, wo Arrays, wo brauche ich Klassen?
- Wo sind Verzweigungen nötig?
- Wo sind Schleifen nötig? Welche sind wofür geeignet?