Programming for Beginners

A Course in Java

Goal of this course is starting to use Java and actively coding a small programme. This course is not an overall course in Java. Even the basics are not covered completely. The concept of object-oriented programming is not covered.

1 Basics

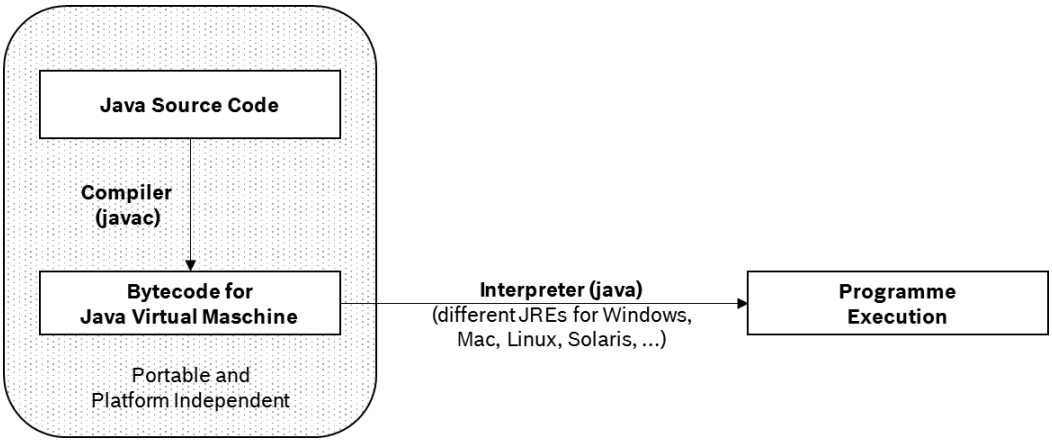
1.1 Introduction to Java

The language in which instructions for computers are written is a computer-programming language. The text written in a computer-programming language is called code; source code is the originally by humans created code which is not „translated for machines“ (compiled). A whole computer programme can be compared to a book: a whole set of texts belonging together.

Java is an object-oriented computer-programming language which could be run on many different devices.

Programming languages could be read by human and not by machines. Therefore, the code written by human has to be translated into machine-readable code. Java translates the source code to machine-readable bytecode by using a compiler. Compilation is done by *javac*, the primary Java compiler included in the Java Development Kit (JDK).

Before compiling, the source code is saved as a `.java` file. The compiler produces a `.class` file. Afterwards, the application could be distributed to different machines where the code could be run by interpreters[1]. The compiled code stays the same for every machine. But the different interpreters take the characteristics of different machines into account. Therefore, you do not have to care about different machines when programming in Java.



The process of deploying a Java programme could be compared to the process of preparing coffee: The source code represents the raw coffee beans. The compiler grinds the coffee into coffee powder

[1] Java is always run in a virtual machine (Java Virtual Machine, JVM). The JVM is different for all machines. Its realization is called Java Runtime Environment (JRE)

(bytecode). After grinding the coffee (compiling the code), everyone has the same product: coffee powder (Java Bytecode). Finally, the coffee powder could be used for many different coffee machines (different interpreters on different computers). Interpreters could work differently but produce the same product: coffee (the executed Java programme).

1.2 Structure

```
public class ClassName {

    public static void main(String[] args) {

        // content of the main method...

    }

}
```

Each Java file has to start with the class name[2]. It is prefixed by the keyword `class`. This indicates that it is a class (most of the files in Java are classes). At the beginning, you could find the modifier `public` which means that the class is accessible by all other classes within your programme (unimportant for the beginning). – If a file has a class, it must have exactly one public class. The name of the file should match exactly the class name.

Conventionally, the first letter of a class name is written in upper case; if several words are used as a class name, you should do camel casing (“ThisIsAClassName”).

The next row (`public static…`) is the main method. In the main method, the processing starts. Exactly one main method is mandatory for all Java programmes.

For the beginning, just accept the class and the main method as a necessary container needed for a Java programme; write your programme code instead of the green text.

Blank lines (containing only whitespace) are ignored by Java.

1.3 Comments

Comments in the code are ignored by the compiler. They could be used for describing your code in any letters. Comments are either enclosed by `/* comment */` or introduced by two slashes (`//`). The former could span over different lines. The latter marks everything as a comment in the line following the two slashes.

1.4 Syntax

Java is case sensitive. Capital letters are different from small letters; `hello` is unequal to `Hello`.

[2] Here, we just assume the very basics. Advanced Java files could be written differently.

Java identifiers (names of variables and methods) have to start with a letter (A to Z or a to z), with a dollar symbol ($) or an underscore (_). After the first character, identifiers can have any combination of characters. An identifier must not be a keyword[3]!

All commands must be closed by a semicolon (`;`).

After calling a method, you always have to write parentheses. In those parentheses, you can provide arguments which are being used by the methods.

Classes and methods (usually also statements and loops) are enclosed by curly brackets. For better readability, the content of those brackets is indented.

If you want to pass a text to a method or want to assign it to a variable, you have to write it in quotation marks. If you want to use the value of any number (e.g. for a variable), you have to write it without quotation marks. To handle numbers as a text, you have to put it in quotation marks.

2 Simple Outputs

The outputs we use are limited to console outputs. In our Integrated Development Environment (IDE), Eclipse, these outputs are displayed in the console.

For console outputs, the commands `System.out.print()` or `System.out.println()` have to be invoked. The former prints out the content of the parentheses. The latter prints it out and is automatically followed by a new line.

```
public class SimpleOutputs {

    public static void main(String[] args) {

        System.out.println("first command...");
        System.out.print("...second command...");
        System.out.println(" ...third command");

        // Output:
        // First command...
        // ...second command... ...third command

    }

}
```

For printing out text, the text has to be put in quotation marks. For printing out numbers, there is no need to surround them with quotation marks.

```
                                    //  Outputs
System.out.println("Text");         //  Text
System.out.println(65);             //  65
System.out.println("65");           //  65
```

[3] https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

3 Arithmetic Operations

In Java the four calculation methods `+`, `-`, `*` and `/` could be used. Important is to not surround the numbers used for calculations with quotation marks.

```
                                        //  Outputs
System.out.println(5 + 5);              // 10
System.out.println(5 - 5);              // 0
System.out.println(5 * 5);              // 25
System.out.println(5 / 5);              // 1
```

Java (and many other programming languages) has a fifth arithmetic sign: % (modulo or modulus). It calculates the remainder when dividing by the number given after the modulo/modulus.

```
                                        // Outputs
System.out.println(3 % 2);              // 1
System.out.println(23 % 5);             // 3
System.out.println(100 % 10);           // 0
System.out.println(8 % 6);              // 2
```

The `+` could also be used to concatenate text. Text and numbers could be concatenated as well.

```
System.out.println("This sentence " + "is concatenated " + "multiple times.");
// Output:  This sentence is concatenated multiple times.
System.out.println("Now follows a number: " + 44);
// Output:  Now follows a number: 44
System.out.println(74 + " is written in front of the text.");
// Output:  74 is written in front of the text.
System.out.println(1 + "1" + 1);
// Output:  111
System.out.println(1 + 1 + 1);
// Output:  3
System.out.println("Parentheses help: " + 1 + 1);
// Output:  Parentheses help: 11
System.out.println("Parentheses help: " + (1 + 1));
// Output:  Parentheses help: 2
```

4 Variables

In Java, different data types exist, every value has a data type. The four data types relevant for us are `String`, `int`, `double` and `boolean`. There are some more primitive data types in Java. We do not cover them in this course.

The data type `String` is used for texts which are always encapsulated in quotation marks. The data type `String` has to start with a capital letter. `int` is used for integers/whole numbers whereas `double` is used for floating-point numbers. `boolean` is a data type which is integral for programming and has only two different values: `true` or `false`.

```
String stringVariable = "Hello Bosch!";
int integerVariable = 4;
double doubleVariable = 3.1;
boolean booleanVariable = true;
```

Before usage, variables must be declared and afterwards initialised. For declaring a variable, the data type is written and afterwards its name. Initialising variables means the initial assignment of a value. These two steps could also be combined.

```java
int x;      // declaring x
x = 3;      // initialising x
int y = 8;  // declaring and initialising y
```

Variables serve as a placeholder. If a variable is written anywhere, it will be replaced by its current value during runtime. You could also change the value of a variable.

```java
String name = "Hans";   // Declaring and initialising name
int age = 8;            // Declaring and initialising age

System.out.println(name + " is " + age + " years old.");
                        // Printing out text with variables

name = "Peter";         // Assigning new value to name
age = 7;                // Assigning new value to age

System.out.println(name + " is " + age + " years old.");
                        // Printing out text with variables

// Output
// Hans is 8 years old.
// Peter is 7 years old.
```

If you want to increase a variable (usually an integer), you could write `variable = variable + 5`. Another option is writing `variable += 5`. There is a short version for increasing a variable by one: `variable++`.

```java
int plus = 5;               // Outputs

plus = plus + 5;
System.out.println(plus);   // 10

plus += 5;
System.out.println(plus);   // 15

plus++;
System.out.println(plus);   // 16
```

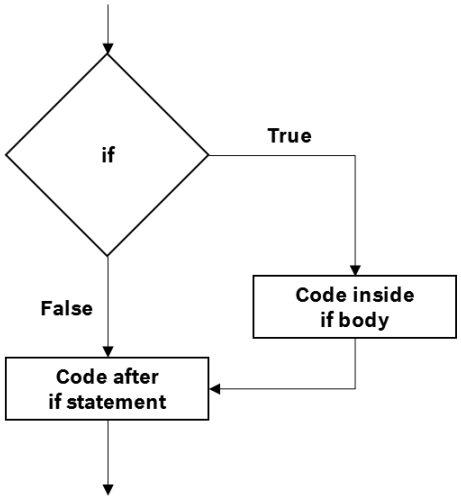
The same options exist for decreasing a variable.

```java
int minus = 8;              // Outputs

minus = minus - 1;
System.out.println(minus);  // 7

minus -= 1;
System.out.println(minus);  // 6

minus--;
System.out.println(minus);  // 5
```

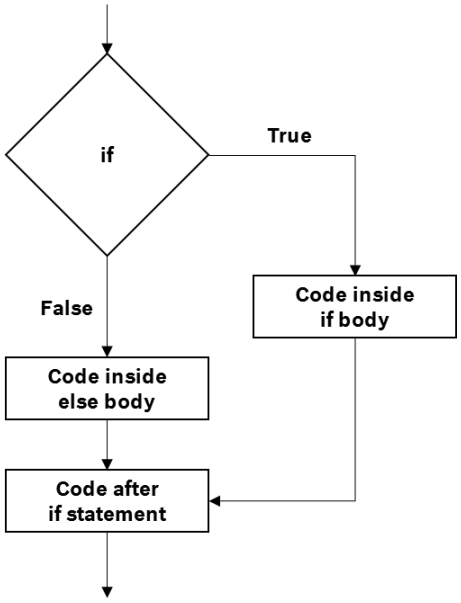
Conventionally, the first letter of variable names is written in lower case. If several words are used as a variable name, each inner word's first letter should be written in upper case.

5 If/Else Statements

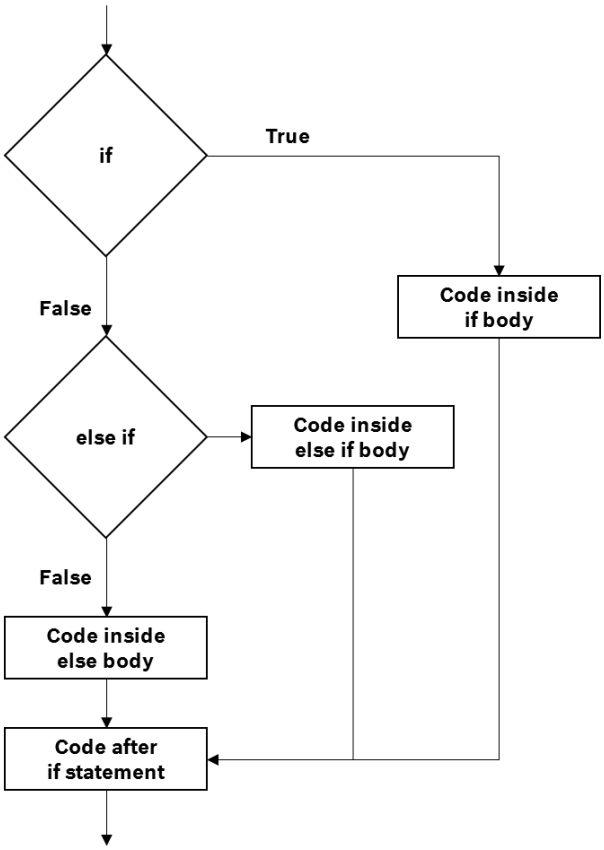
If/else statements are used for decision making. They consist out of a condition (in parentheses) followed by a code block (in curly brackets). The condition is a boolean: it can be only `true` or `false`. The code in the curly brackets following the condition gets only executed if the condition is true.



An if statement could also be followed by an `else`. The code in the else statement is executed in all cases in which the condition of the if statement is not true.



For checking a second condition (multiple conditions) before executing the else statement, an `else if` could be added.



As conditions, of course, you could just use the words true or false. But often you want to check if a variable has a certain value. For comparing two values (e.g. integer variable and number) you could use different relational operators.

int x = 10, y = 20;		
Operator	Description	Example
== (equal to)	Checks if values of two operands are equal or not. If yes, condition becomes true	(x==y) is false
!= (not equal to)	Checks if values of two operands are equal or not. If values are not equal, condition becomes true	(x!=y) is true
> (greater than)	Checks if value of left operand is greater than value of right operand. If yes, condition becomes true	(x>y) is false
< (less than)	Checks if value of left operand is less than value of right operand. If yes, condition becomes true	(x<y) is true
>= (greater than or equal to)	Checks if value of left operand is greater than or equal to value of right operand. If yes, condition becomes true	(x>=y) is false
<= (less than or equal to)	Checks if value of left operand is less than or equal to the value of right operand. If yes, condition becomes true	(x<=y) is true

Furthermore, several if statements could be nested. You could write one if statement inside another one.

```java
int age = 58;

if(age < 21) {
    System.out.println("You are too young.");
}
else if(age < 60) {
    System.out.print("Your age is sufficient.");
    if(age > 55) {
        System.out.println(" But you are nearly too old.");
    }
}
else {
    System.out.println("You are too old.");
}

// Output
// Your age is sufficient. But you are nearly too old.
```

If you have more than one condition, you have the option of nesting/writing several if statements or you could combine two conditions with `||` (or) or `&&` (and).

```java
int no1 = 5;
int no2 = 22;

// several ifs
if(no1 < 10) {
    System.out.println("At least one number is less than 10.");
}
else if(no2 < 10) {
    System.out.println("At least one number is less than 10.");
}
// Output
// At least one number is less than 10.

// combination with ||
if(no1 < 10 || no2 < 10) {
    System.out.println("At least one number is less than 10.");
}
// Output
// At least one number is less than 10.
```

```java
int no1 = 5;
int no2 = 22;

// nested if
if(no1 < 10) {
    if(no2 > 10) {
        System.out.println("no1 is less than 10, no2 is greater.");
    }
}
// Output
// no1 is less than 10, no2 is greater.

// combination with ||
if(no1 < 10 && no2 > 10) {
    System.out.println("no1 is less than 10, no2 is greater.");
}
// Output
// no1 is less than 10, no2 is greater.
```

When you want to check boolean variables, you can compare it with the relevant value or just regard the respective variable as a whole statement. For negating the condition use an exclamation mark.

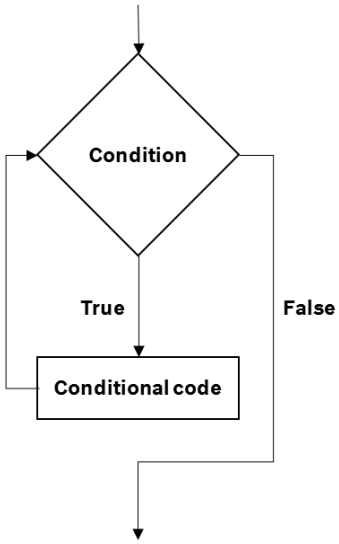
```
boolean sweet = true;
boolean sour = false;

if(sweet) {
    System.out.println("You like sweet stuff.");
}
else {
    System.out.println("You don't like sweet stuff.");
}
if(!sour) {
    System.out.println("You don't like sour stuff.");
}
else {
    System.out.println("You like sour stuff.");
}

// Output
// You like sweet stuff.
// You don't like sour stuff.
```

Conditions could also compare arithmetic operations or rather the generated values by those operations.

```
double z = 256.6;

if(z/10 <= 25) {
    System.out.println("The provided value is to small.");
}
else if(z/5 > 50 && z <= 500-2) {
    System.out.println("The provided value lies in the interval (250,498].");
}
else {
    System.out.println("The provided value is to big");
}

// Output
// The provided value lies in the interval (250,498].
```

6 Loops

Loops are used for executing a block of code multiple times. Loops have, also, a condition which is checked and a code block which is executed as long as the condition is true.



Similar to if/else statements, the condition is a boolean. You could imagine loops as a looped if statement; the condition gets checked and the related block of code gets executed until the condition is false.

The most universal solution is the while loop: The condition gets checked. If the condition is true, the related code gets executed and content checked in the condition must be changed. Eventually, the condition gets checked again. If the result of this check is still true, the loop will be executed again. Otherwise, the programme continues after the loop. – When the content checked in the condition does not change in the body of the loop, you will have an infinite loop running forever!

```
int i = 0;

while(i < 8) {
    System.out.println("I'm still sleeping...");
    i++;
}

System.out.println("I'm awake now!");

// Output
// I'm still sleeping...
// I'm still sleeping...
// I'm still sleeping...
// I'm still sleeping...
// I'm still sleeping...
// I'm still sleeping...
// I'm still sleeping...
// I'm still sleeping...
// I'm awake now!
```

The iterator used in the condition could be used in the body.

```java
int i = 5;

while(i >= 1) {
    System.out.println("I will arrive in " + i + " minutes.");
    i--;
}

// Output
// I will arrive in 5 minutes.
// I will arrive in 4 minutes.
// I will arrive in 3 minutes.
// I will arrive in 2 minutes.
// I will arrive in 1 minutes.
```

Be careful: If you declare a variable within a loop, it will only be available in the respective iteration of the loop. It will be declared newly each iteration and only the one declared in the last iteration will be available after the loop.

You could declare a variable before entering the loop and use it in the loop (similar to the iterator).

```java
/*
 * 5! is pronounced as five faculty.
 * It means 5*4*3*2*1
 */

int facultyOf = 5;
int result = 1;

while(facultyOf > 0) {
    result *= facultyOf;
    facultyOf--;
}

System.out.println("Faculty of 5: " + result);
```

7 Task

Write a Java programme which iterates and prints the integers from 1 to 100. For multiples of three replace the number with your first name. For multiples of five replace the number with your last name. When the number is divisible by both three and five, print your first and last name.

You should start structuring the task and developing the result iterative. At the beginning, you could try to print the integers from 1 to 100. In the next step, the numbers divisible by three could be replaced...

After solving this task completely, you could refine the solution. Pretend the used data is not fixed and fetched from a database or based on user input; try replacing some fixed values by variables:

- The range of numbers being iterated
- The texts used for replacing numbers
- The numbers being replaced
- ...

```
1
2
Felix
4
Seifert
Felix
7
8
Felix
Seifert
11
Felix
13
14
Felix Seifert
16
17
Felix
19
...
```

Example Output

8 Where to go from here?

With this course, you just got a simple idea how you have to think about problems as a programmer, how humans try to communicate with binary working machines. This way of thinking analytically could be used for many different programming languages. Also, lots of other problems could be structured like programmers do this and solved analytically.

If you want to continue learning Java, you should complete the basics and learn the other primitive data types and the other ways of writing loops. As you already understand the content of this course, picking up this knowledge might be easier for you. It should be also a good repetition of the process of development. Furthermore, you should understand how variables could be stored in arrays and how to handle these.

The next big and important step might be methods. Learning how to use methods would help you structuring your code. But it is also a new step in the direction of abstraction. You should understand especially the two ways of programmatically solving problems: iteration and recursion. – This might be a good point of thinking about the efficiency of different ways of programming.

After understanding methods, there is a quite difficult step: understanding the process of object-oriented programming (OOP). This is a very extensive topic and sometimes quite complicated to abstract your problem in the OOP way. However, it could be very often transferred to the real world.

For continues learning, you could search for different tutorials. There are many tutorials available for free on the internet. You have to find one which you like because there are different ways of imparting knowledge. – The most important thing is to continue learning and practising. When waiting too long without any usage of this new knowledge, you might forget lots of it.

If you have any questions, you could message me. However, asking other programmers might often work as they like to help. Asking for help, you should always provide your source code and the produced error.

Additionally, you could just ask other people learning Java or search for your problem on the internet; helping others also improves your own capabilities, not only of programming but also of imparting technical knowledge to others.