EXERCISE 1-1

Statements

Conditionals

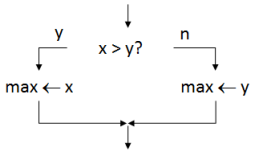
"Conditionals provide a simple change in the flow of execution - execute the statements in one of two blocks, depending on a specified condition." [2]

Java enables two different statements: the *if-statement* as two-way-branching and the *switch-statement* as multi-way-branching.

if-Statement

An if-statement checks a condition whether it is true or false. The program follows the statements dependent on the selected condition. The statement, which is executed in case of true, is called *if-branch* and the other is called *else-branch*.

The flow diagram



is coded in Java as follows (keywords are `if` and `else`):

```
1 if (x < y) max = x; else max = y;
```

The statement starts with the keyword `if`. The condition has to be put in round brackets, followed by the statements of the if-branch. The else-branch may be missing. In case of more than one statement in a branch, the statement block has to be encapsulated in curly brackets.

Dangling else If an if-branch consists of another if-branch, there may occur the following situation:

```
1 if (a > b)
2    if (a > 0) max = a;
3 else
4    max = b;
```

Now, the question is: To which if-statement belongs the else-branch? To `(a > b)` or to `(a > 0)`?

Java solves this problem - named ambiguity - by defining that an else-branch always belongs to the if-statement directly situated before the else. If this behaviour is unwanted, we will have to use curly brackets.

```
1 if (a > b){
2    if (a > 0) max = a;
3 } else
4    max = b;
```

For avoiding misunderstandings, we shall always use brackets.

Boolean Expressions

In Java we can compare two operators by using six different cases:

	Description	Example
`==`	equal	`x == 10`
`!=`	unequal	`x != y`
`>`	greater than	`7 > 3`
`<`	less than	`x < y`
`>=`	greater than or equal to	`x >= y`
`<=`	less than or equal to	`x <= y`

Boolean values (and comparisons) can be connected by using operators `&&`, `||` and `!`, which are listed as follows:

		And	Or	
x	y	x && y	x || y	!=
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Java stops execution of a comparing expression as soon as the value of this expression can be determined.

```
if (y != 0 && x/y > 100) ...
```

Firstly, the condition `y!=0` will be checked. If this first comparison delivers `false`, checking the second condition `x/y>100` will be obsolete and will be ignored by Java.

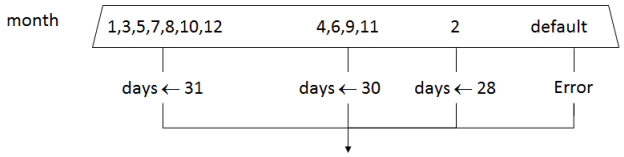
This is called conditional statement.

Having conditional expressions, there are some priority rules: `!` binds strongest, followed by `&&` and last `||`.

switch-Statement

The switch-statement allows handling multi-way-branches. It checks the value of an expression of type `int`, `short`, `byte`, `char` or `String`. Then, it takes the possible way dependent on this value.

The flow diagram



is coded in Java as follows (keywords are `switch`, `case`, `default` and `break`):

```
switch (month) {
   case 1: case 3: case 5: case 7: case 8: case 10: case 12:
      days = 31; break;
   case 4: case 6: case 9: case 11:
      days = 30; break;
   case 2:
      days = 28; break;
   default:
      System.out.println("Error");
}
```

The value, which has to be checked, is put in round brackets after the keyword `switch`. The list of case-markers, which proposes possible values, is enclosed by curly brackets. If a case-marker is true, the corresponding statement block will be executed. The keyword `default` hosts all cases that are not listed as case-markers. The keyword `break` guarantees that the switch-statement will be left at this position and that the program will continue with the next statement.

In comparison of switch-statement to if-statement, it has to be stated that the switch-statement is faster but allocates more memory.

Assertions with Conditionals

An assertion can be written as comment. It is ignored by the compiler but enables better understanding of the program.

In an if-statement there are two assertions: the first one is `true` and mostly trivial; the second one is `false` at the beginning of the else-branch.

```
if (x > y)  // x > y
   ...
else  // !(x > y) -> x <= y
   ...
```

Negation of Composed Expressions As already mentioned, at the beginning of an else-branch the negation of the if-condition is valid. Negation of a composed expression can be simplified by using rules of *DeMorgan* (August DeMorgan, 1806-1871):

$$!(x \,\&\&\, y) \equiv !x \,||\, !y$$

$$!(x \,||\, y) \equiv !x \,\&\&\, !y$$

assert-Statement Assertions are statements, which are always true. Java allows checking such statements by using an `assert`-statement.

```
assert x <= y && y <= z;
```

The assert-statement starts with the keyword `assert`, followed by a boolean expression that has to be true at runtime.

We can define a second expression, which can be displayed with an error message.

```
assert x >= 0: "value of x is negative"
```

In case that the expression before the colon is false, the program will terminate with an `AssertionError`.

Normally, assertions are ignored by the compiler because of using runtime. If we want considering assertions, we will have to enable them.

```
$ java -enableassertions MyProgram
```

Loops

"Loops provide for a more profound change in the flow of execution - execute the statements in a block as long as a given condition is true." [2]

Java enables three different ways of using loops: while-, do-while- and for-statement. [1]

while-Statement

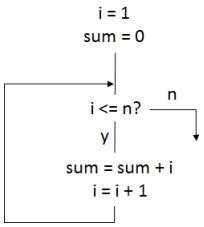
The expression in round brackets is called the condition. The statements in curly brackets are called body. The while-statement will be executed as follows:

1. Evaluate the condition, either it is true or false.
2. If the condition is false, then ignore the body and continue with the next statement.
3. If the condition is true, then execute the body and return to step 1.

We want to write a program that is able to sum up whole numbers from 1 to an upper limit `n`, such as

sum = 1 + 2 + 3 + 4 + ... n.

Then, the flow diagram



is coded in Java as follows (keyword is `while`):

```
...
i = 1; sum = 0;
while (i <= n) {
    sum = sum + i;
    i++;
}
```

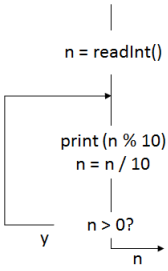
Assertions with Loops

It is advisable to use assertions also in connection with loops. The first assertion shall be placed at the beginning of the loop body (mostly identical to the loop condition). The second assertion shall be put after the loop and can be produced by negation of the loop condition.

do-while-Statement

We want to write a program that is able to display the digits of a whole number in reverse order. For example, the number `123` should be displayed such as `321`.

The flow diagram



is coded in Java as follows (keywords are `do`, `while`):

```
...
int n = scanner.nextInt();
do {
   System.out.print(n % 10);
   n = n / 10;
} while (n > 0);
```

for-Statement

If we know how many iterations will be requested by the loop, we can use the for-statement. It consists of four parts:

- *Initialization*: will be executed before entering the loop and defines the control variable.
- *Termination Condition*: will be checked each time the loop is entered.
- *Increment*: will be executed at the end of an iteration and increases the control variable.
- *Body*: statement block within the loop.

The example "sum up whole numbers from 1 to an upper limit `n`" can be realized with the help of the for-statement as well:

```
int sum = 0;
int i;
for (i=1; i<=n; i++)
   sum = sum + i;
```

Termination of Loops

It is possible to terminate loops prior running through all iterations dependent on a particular condition by using the `break`-statement.

Normally, we leave the current loop by using the `break`-statement. However, we can terminate enclosing loops as well. Therefore, we have to use labels. For example,

```
L: // label
   for (;;) { // infinite loop
      ...
      for (;;) {
         ...
         if (...)
            break; // terminates inner loop
         else
            break L; // terminates outer loop
```

Measurements

The following algorithm executes a data stream with unknown length.
A sensor delivers measurements as a stream of whole numbers. We have to write a program namend `FilterSamples` that is able to work with these measurements as follows:

- Negative values will be deleted.
- If there is a group of more than two or more directly sequenced zeros this group will be summarized to one single zero.
- The remaining measurements will be displayed.
- The end of a measurement sequence is indicated by the triple `-1, -1, -1` which is normally not part of a stream.

The program reads measurements from the keyboard (first line), executes the stream and displays the result as output (second line). It terminates only in case of `-1 -1 -1`.

What You Should See

Use Java via Command Prompt

```
C:\...\workspace\neon\Statements\bin> java at.ithtl.sew.statements.FilterSamples
1 2 3 0 0 -1 -1 3 4 -1 -1 -1
1 2 3 0 3 4
```

```
C:\...\workspace\neon\Statements\bin> java at.ithtl.sew.statements.FilterSamples
-2 -1 -1 -1
```

```
C:\...\workspace\neon\Statements\bin> java at.ithtl.sew.statements.FilterSamples
0 0 -2 0 0 -1 -1 -1
0
```

Do It

1. *Conditionals.*

   Write a Java program that accepts three values `x`, `y` and `z` as input and checks

   - if `x`, `y` and `z` are not all the same,
   - if `x`, `y` and `z` are all different,
   - if at least t wo values are equal.

2. *Boolean Expressions.*

   Simplify the following boolean expressions with the help of the rules of DeMorgan:

   (a) $!(x < y \;\&\&\; y < z)$

   (b) $(x != y) \,||\, !(y == z \;\&\&\; y == z)$

   (c) $!(x \geq -3 \;\&\&\; x \leq 0) \;\&\&\; 5 < x$

3. *Boolean Variables.*

   What are the values of the variables `a`, `b`, and `c`, if the following sequence of instructions is given and the input values for `x` are subsequently -1, 0, 5, and 10?

```
...
int x = scanner.nextInt();
boolean a = x > 0 && x <= 10;
boolean b = x < 5 || x > 9;
boolean c = !(b || a);
...
```

4. *Leap Year.*

   Write a Java class that allows determining leap years.

   Write a test that executes the leap year functionality for each two years between 1999 and 2000.

5. *Conditionals.*

   What (if anything) is wrong with each of the following statements?

   (a) `if (a > b) then c = 0;`

   (b) `if a > b { c = 0; }`

   (c) `if (a > b) c = 0;`

   (d) `if (a > b) c = 0 else b = 0;`

6. *Loop.*

   What does the following program print?

```java
int f = 0;
int g = 1;
for (int i = 0; i <= 15; i++) {
   System.out.println(f);
   f = f + g;
   g = f - g;
}
```

7. *Sum of Digits.*

   Write a program that is able to compute and display the sum of digits of the number `x`. For instance, the sum of digits of the number 4512 is 12.

Common Student Questions [2]

Is ambiguity in nested if-statements a problem?

Yes. In Java, when we write

```
if <expr1> if <expr2> <stmntA> else <stmntB>
```

it is equivalent to

```
if <expr1> { if <expr2> <stmntA> else <stmntB> }
```

even if we might have been thinking

```
if <expr1> { if <expr2> <stmntA> } else <stmntB>
```

Using explicit braces is a good way to avoid this dangling else pitfall.

What is the difference between a for loop and its while formulation?

The code in the for loop header is considered to be in the same block as the for loop body. In a typical for loop, the incrementing variable is not available for use in later statements; in the corresponding while loop, it is. This distinction is often a reason to use a while instead of a for loop.

Vocabulary [1]

In the following, we define several terms, which are useful for further understanding.

iteration Executing a sequence of statements repeatedly.

loop A statement that executes a sequence of statements repeatedly.

loop_body The statement inisde the loop.

infinite_loop A look whose condition is always true.

program_development A process for writing programs. So far we have seen “incremental development” and “encapsulation and generalization”.

encapsulate To wrap a sequence of statements in a method.

generalize To replace something unnecessarily specific (like a constant value) with something appropriately general (like a variable or parameter).

loop_variable A variable that is initialized, tested, and updated in order to control a loop.

increment Increase the value of a variable.

decrement Decrease the value of a variable.

References

[1] Allen B. Downey and Chris Mayfield. *Think Java.* O'Reilly Media, 2016.

[2] Robert Sedgewick and Kevin Wayne. *Algorithms Fourth Edition.* Addison-Wesley, 2011.