

java from scratch Knowledge Base

- ✓ Welcome
- ✓ Mastering Agile and Scrum: Video Training Series
- ✓ Program
- ✓ Introduction
- ✓ The architecture of an operating system
- ✓ The structure of files and directories
- Navigating through directories
- Environment variables
- Extracting archives
- Installing the software
- Monitoring the usage of system resources
- Ending – control questions
- Software Installations
- IntelliJ EduTools – installation
- Introduction
- A brief history of Java
- First program
- Types of data
- Operators
- Conditional statements**
- Loops

| java from scratch Knowledge Base

Conditional statements

Until now, the code we wrote was executed sequentially from top to bottom; unconditionally. However, in programming the logic of our application very often depends on some conditions. For example, if the amount to be paid exceeds 1000, we would like to charge a discount. Otherwise, the amount will remain unchanged. To accomplish this, we can use *conditional statements*.

The if statement

The **if** conditional statement checks the logical condition contained between brackets (and). If the condition **is met** (is true), the statement executes the code contained between brackets { and }.

Structure of the **if** statement

```
if (<logical-condition>) {  
    // operations  
}
```

Example

```
if (totalPrice > 1000) {  
    totalPrice = totalPrice - discount;  
}
```

Note

Please note the indentation of the code in the line in the **if** body. Similarly as in the code in the **main** function, also in conditional statements indentation is applied.

A logical condition is a statement which after calculation returns

- Arrays
- Object-oriented programming
- Conclusion
- Assignments
- Basics of GIT – video training
- HTTP basics – video training
- Design patterns and good practices video course
- Prework Primer: Essential Concepts in Programming
- Cybersecurity Essentials: Must-Watch Training Materials
- Java Developer – introduction
- Java Fundamentals – coursebook
- Java fundamentals slides
- Java fundamentals tasks
- Test 1st attempt | after the block: Java fundamentals
- Test 2nd attempt | after the block: Java fundamentals
- GIT version control system coursebook
- Java – Fundamentals: Coding slides
- Java fundamentals tasks
- Software Testing slides
- Software Testing Coursebook
- Software Testing tasks
- Test 1st attempt | after the block: Software testing
- Test 2nd attempt | after the block: Software testing
- Java – Advanced Features coursebook

value **true** or **false**. It is placed in round brackets. Examples of correct logical conditions are as follows:

- `age > 5`
- `price == 7.45`
- `dayNr != 7`
- `!active` (where `active` is of the `boolean` type)
- `true`
- `false`

Examples of incorrect logical conditions:

- `age + 1`
- `price = 7.45`
- `dayNr % 2`
- `1`
- `"Alice has a cat"`

A logical condition can be complex and consist of several expressions, but the result must still be a logical value.

Example of a complex logic condition

```
if ((age > 18 && dayNr == 6) || amount > 1000) {  
}
```

The if – else statement

The code after the `else` statement (included in brackets `{` and `}`) is invoked when the logical condition of the `if` statement is **not met** (is false).

Structure of the if – else statement

```
if (<logical-condition>) {  
    // operations  
} else {  
    // operations  
}
```

Example of the if – else statement

```
if (age >= 18) {  
    System.out.println("You are adult");  
} else {  
    System.out.println("You are underage");
```

- Java – Advanced Features slides
- Java – Advanced Features tasks
- Test 1st attempt | after the block: Java Advanced Features
- Test 2nd attempt | after the block: Java Advanced Features
- Java – Advanced Features: Coding slides
- Java – Advanced Features: Coding tasks
- Test 1st attempt | after the block: Java Advanced Features coding
- Test 2nd attempt | after the block: Java Advanced Features coding
- Data bases SQL coursebook
- Databases SQL slides
- Databases – SQL tasks
- Coursebook: JDBC i Hiberate
- Excercises: JDBC & Hibernate
- Test 1st attempt | after the block: JDBC
- Test 2nd attempt | after the block: JDBC
- Design patterns and good practices
- Design patterns and good practices slides
- Design Patterns & Good Practices tasks
- Practical project coursebook
- Practical project slides
- HTML, CSS, JAVASCIRPT Coursebook
- HTML, CSS, JAVASCRIPT slides
- HTML, CSS, JavaScript tasks

```
    }  
}
```

The if – else if – else statement

The `else if` structure is used to combine conditional statements and works just like the connection of `if` and `else`.

Structure of the `if – else if – else` statement

```
if (<logical-condition- 1>) {  
    // operations  
} else if (<logical-condition- 2>) {  
    // operations  
  
    ...  
  
} else {  
    // operations  
}
```

Example of the `if – else if – else` statement

```
if (dayOfTheWeekNr == 7) {  
    System.out.println("Today is Sunday – holiday!");  
} else if (dayOfTheWeekNr == 6) {  
    System.out.println("Today is Saturday – free!");  
} else {  
    System.out.println("Today is a business day – work!");  
}
```

Braces `{` and `}` are not required, but it is ADVISABLE to ALWAYS use them, even when the `if` is followed by one statement only. A code with one statement after `if` will be working properly.

```
if (totalPrice > 1000)  
    totalPrice = totalPrice - discount;
```

But if we would like to add, for example, a discount from the seller in the next line, unfortunately the code will not work as we would expect—the discount from the seller will always be counted unconditionally. In addition, the indentation in the second line of the `if` statement can be confusing.

- Test 1st attempt | after the block: HTML,CSS,JS
- Test 2nd attempt | after the block: HTML,CSS,JS
- Frontend Technologies coursebook
- Frontend technologies slides
- Frontend Technologies tasks
- Test 1st attempt | after the block: FRONTEND TECHNOLOGIES (ANGULAR)
- Test 2nd attempt | after Frontend technologies
- Spring coursebook
- Spring slides
- Spring tasks
- Test 1st attempt | after the block: spring
- Test 2nd attempt | after the block: spring
- Mockito
- PowerMock
- Testing exceptions
- Parametrized tests
- Final project coursebook
- Final project slides
- Class assignments

```
if (totalPrice > 1000)
    totalPrice = totalPrice - discount;
    totalPrice = totalPrice - dealersDiscount;
```

To make the code work as intended, we must enclose it in brackets.

```
if (totalPrice > 1000) {
    totalPrice = totalPrice - discount;
    totalPrice = totalPrice - dealersDiscount;
}
```

Ternary operator

A *ternary operator* is also called a *conditional operator*. It is similar to the `if-else` statement, but it returns the value. In addition, it can be seen as an abbreviated record of this statement.

Structure of a ternary operator

```
<logical-expression> ? <value-1> : <value-2>
```

If the value of the *logical expression* is `true`, the *value-1* expression is calculated and the operator returns its result. If the *logical expression* is `false`, then the *value-2* expression is calculated and the operator returns its result.

Example of a ternary operator

```
String text = age >= 18 ? "adult" : "underage";
System.out.println("You are" + text);
```

The above logic can also be expressed with the use of the `if-else` statement.

Example of the `if - else` statement corresponding to the ternary operator

```
String text;
if (age >= 18) {
    text = "adult";
} else {
    text = "underage";
}
System.out.println("You are " + text);
```

```
<logical-expression> ? <value-1> : <value-2>
```

Usually *value-1* and *value-2* are of the same type, e.g. `int` or `String` as in the previous example. However, in the example below:

```
System.out.println(true ? 1 : 3.1415);
```

although the *logical expression* is `true` (is unconditionally true), the screen will display value `1.0` and not `1` as initially expected. The ternary operator works in such a way that the returned type is the common (narrowest) type, in this case it is `double`. If we would like to assign the result of this operation, we could write, for example:

```
double result = true ? 1 : 3.1415;
```

The switch statement

Another type of conditional statement is the `switch` statement. It is useful in situations where there are many options to choose from; then the `if - else` statement may be ineffective.

Structure of the `switch` statement

```
switch (<variable>) {  
    case <value-1>:  
        // operations  
        break;  
    case <value-2>:  
        // operations  
        break;  
  
    ...  
  
    default:  
        // operations  
}
```

Example of rolling the dice

```
int valueOnTheDice = 5;  
switch (valueOnTheDice) {
```

```
case 1:  
    System.out.println("Outcome is 1");  
    break;  
case 2:  
    System.out.println("Outcome is 2");  
    break;  
case 3:  
    System.out.println("Outcome is 3");  
    break;  
case 4:  
    System.out.println("Outcome is 4");  
    break;  
case 5:  
    System.out.println("Outcome is 5");  
    break;  
case 6:  
    System.out.println("Outcome is 6");  
    break;  
default:  
    System.out.println("Error or the dice has  
landed on the edge!");  
}
```

For the given value of the `valueOnTheDice = 5` variable, the following result will be displayed on the screen:

Outcome is 5

If we change the value of the `valueOnTheDice` variable for instance to 100, the following text will be displayed:

Error or the dice has landed on the edge!

The `case` variable and values can be as follows:

- constant expressions of `byte`, `short`, `char` and `int` types
- relevant wrapper classes: `Byte`, `Short`, `Character` and `Integer`
- enumeration constants (`Enum`)
- character `Strings`

The `default` section is optional. It is executed when none of the values in the `case` can be adjusted.

The `break` statements are also optional, but they are very significant in this structure. I will present it using an example with dice. Let us remove the `break` statement from the `case` statement:

```
int valueOnTheDice = 5;
switch (valueOnTheDice) {
    case 1:
        System.out.println("Outcome is 1");
    case 2:
        System.out.println("Outcome is 2");
    case 3:
        System.out.println("Outcome is 3");
    case 4:
        System.out.println("Outcome is 4");
    case 5:
        System.out.println("Outcome is 5");
    case 6:
        System.out.println("Outcome is 6");
    default:
        System.out.println("Error or the dice has
                           landed on the edge!");
}
```

For the provided value of the `valueOnTheDice = 5` variable, the following text will be displayed on the screen now:

```
Outcome is 5
Outcome is 6
Error or the dice has landed on the edge!
```

Why has this happened? The `switch` statement works so that as soon as it matches the `case` value, it executes all statements from all the `case` sections (including `default`), until it encounters a possible `break` statement. It may look a bit unintuitive, but remember about this and use the `break` statement properly.

When can such an operation be useful? For example, when for the value of pips on the dice we want to display the information whether the result was even. Then we can apply two `break` statements and group the `case` values together.

```
int valueOnTheDice = 5;
switch (valueOnTheDice) {
    case 1:
    case 3:
    case 5:
        System.out.println("Odd number of pips");
        break;
    case 2:
    case 4:
    case 6:
        System.out.println("Even number of pips");
}
```

```
        System.out.println("Even number of pips");
        break;
    default:
        System.out.println("Error or the dice has
    landed on the edge!");
}
```

For values 1, 3 and 5, the following text will be displayed:

Odd number of pips

For values 2, 4 and 6, the displayed text will be as follows:

Even number of pips

Summary

What have you learned in this chapter?

What are the conditional statements in Java?

These are the **if** and **switch** statements. The **if** statement has a few forms: **if**, **if – else** and **if, else if – else**.

What is a logical condition?

A logical condition is an expression that (after a possible calculation) returns the **true** or **false** value.

What types can be used in the `case` value in the `switch` statement?

These can be as follows:

- byte, short, char and int types
- relevant wrapper classes: Byte, Short, Character and Integer
- enumeration constants (Enum)
- character strings (String)

What is a `ternary operator`?

This is an abbreviated form of the **if – else** statement, yet it returns the value. It is expressed as:

<logical –expression>? <value-1> : <value-2>

When is it advisable to use the `switch` statement instead of the `if` statement?

When we have many options to choose from or there is one option for multiple values.

Exercises

- The following code simulates the roll of a dice:

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

 - will display whether the result is even or odd
 - if the result is 6, write additionally the word WON
- Using a class that offers date and time operations, download:
 - current time
 - number of the day of the week
 - number of the day of the month
 - number of the month
 - based on these data, choosing the optimal conditional statement, display:
 - information about the time of day (setting arbitrarily): morning, noon, afternoon, evening, night
 - information about the day: working, free, holiday
 - information whether it is already after the payment (assuming that the payment comes by the 10th of each month)
 - information about the season: spring, summer, autumn, winter

Complete Lesson