Object-Oriented Programming in Java

Lecture 2 - Imperative Concepts

Emily Lucia Antosch

HAW Hamburg

13.08.2025

Contents

1. Introduction

1.1 Where Are We Now?

- In the introduction, I gave you an overview of the topics for the upcoming lecture.
- You have also written your first program in Java!
- Today we'll cover **Imperative Concepts**.

1.1 Where Are We Now?

1. **Imperative Concepts**
2. Classes and Objects
3. Class Library
4. Inheritance
5. Interfaces
6. Graphical User Interfaces
7. Exception Handling
8. Input and Output
9. Multithreading (Parallel Computing)

1.2 The Goal of This Chapter

- We will discuss imperative concepts in programming with Java.
- You will understand the simple data types in Java.
- You will control program flow with control structures and loops.
- You will apply the correct coding style.

2. Simple Data Types

? Question

How can a program remember its state?

? Question

How can a program remember its state?

- Variables that store the state in the computer's memory.
- The content of the memory on the computer is interpreted based on the **data type**.

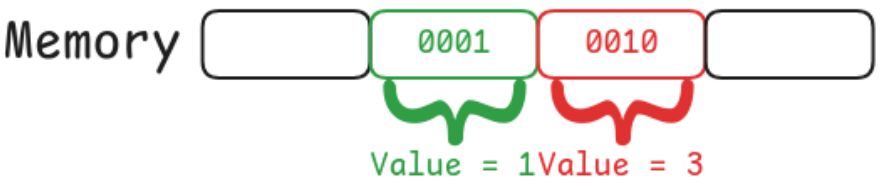


Figure 1: Memory in the computer with values from the program

? Question

Which data types do you already know from C?

? Question

Which data types do you already know from C?

- **int**, **char**, **float**, **double**
- **struct**, **enum**, **union**
- **void**, **bool**
- **Arrays with []** and **Pointers with ***

2.2 Data Types in Java

The following data types are available in Java:

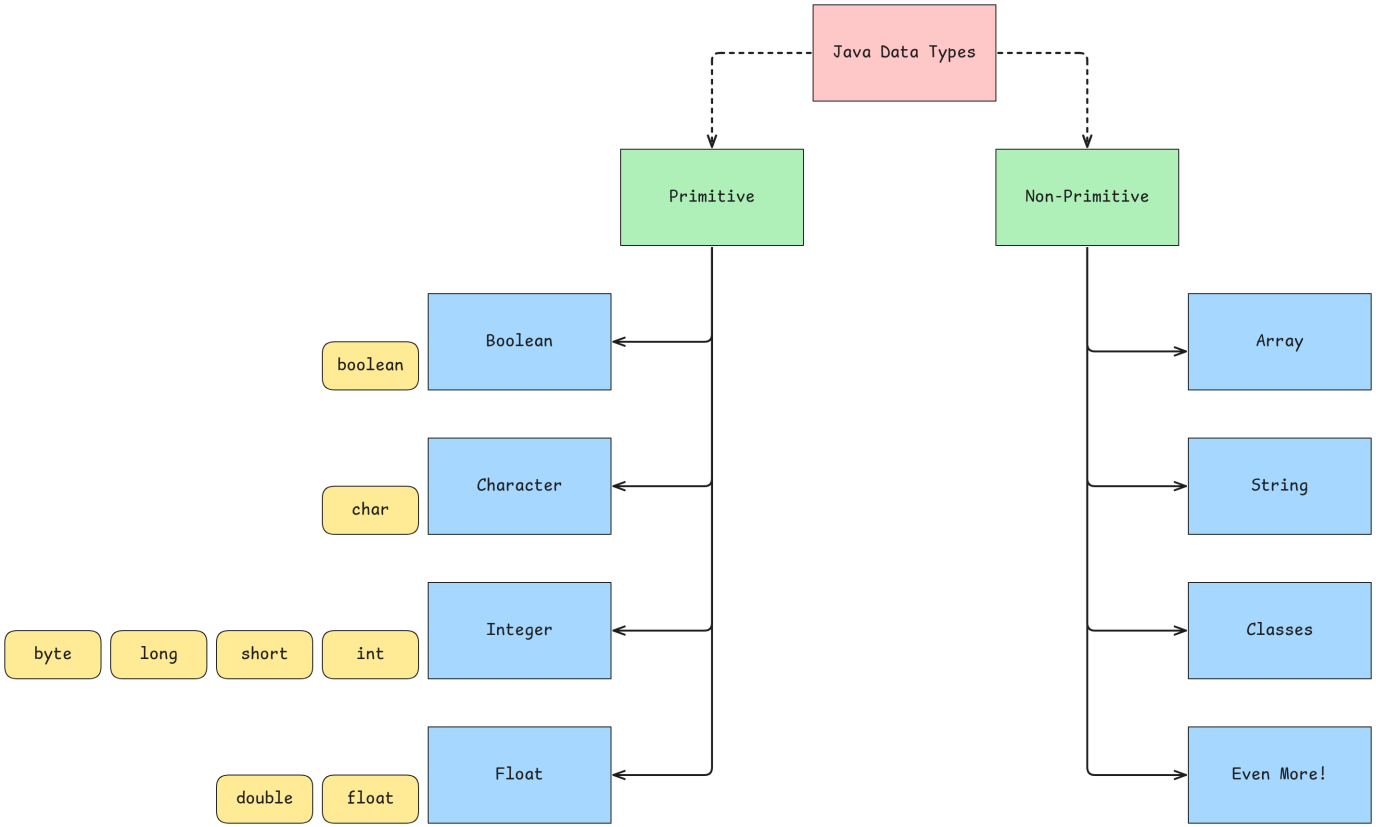


Figure 2: Data types in Java

2.2 Data Types in Java

- Memory sizes and the corresponding value ranges:

2.2 Data Types in Java

Type	Data Type	Size	Value
Integer	`byte`	1 Byte	-2^7 to $2^7 - 1$
	`short`	2 Byte	-2^{15} to $2^{15} - 1$
	`int`	4 Byte	-2^{31} to $2^{31} - 1$
	`long`	8 Byte	-2^{63} to $2^{63} - 1$
Character	`char`	2 Byte	0 to $2^{16} - 1$
Floating Point	`float`	4 Byte	
	`double`	8 Byte	
Truth	`boolean`	1 Bit	`true` or `false`

Table 1: Value ranges of data types

2.3 Declaration of Variables

Memorize

Variables must be declared before they can be used.

- A data type is written before the variable name.
- A declaration could look like this:

```java
int a;
float b;
char c;
```

2.4 Initialization of Variables

! Memorize

After declaration, a value can be assigned. This is called initialization.

- A value is assigned to the variable using the assignment operator =:

```java
a = 5;
b = 3.5;
c = 'A';
```

2.5 Definition of Variables

! Memorize

Declaration and initialization can also be done in one step.
This is then called definition.

Java

```java
int a = 5;
float b = 3.5;
char c = 'A';
```

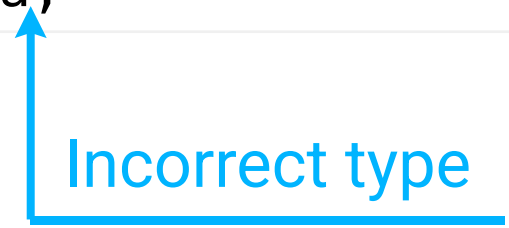
2.6 Scope of Variables

- Variables have a scope that is defined by the curly braces.
- Variables can be declared at any point in the code.
- The compiler prevents the use of variables that have not been initialized.

2.7 Type Correctness

- Types must be correct to avoid errors.
  - Unlike in C, values must be assigned to the correct data type.
  - The following would not work:

```java
int a = 5;
float b = a;
```



2.7 Type Correctness

? Question

What differences do you see between C and Java when it comes to data types?

- No composite data types in Java.
- No `unsigned` in Java.
- Memory sizes are fixed and guaranteed.
- Characters are encoded with 2 bytes.
  - 65,536 characters can be represented instead of 256.

2.8 Literals

! Memorize

A **literal** is a constant, immutable number or string that appears directly in the code.

- So when you write a specific value directly in code, you use a literal.
- This is then not represented by a variable.

? Question

Why do you think the following code doesn't work?

```java
float point = 3.1416;
```

2.8 Literals

? Question

Why do you think the following code doesn't work?

```java
float point = 3.1416;
```

- The number is a fixed floating-point number that is interpreted by Java as **double**.
- Due to type correctness, the value is not stored in a **float** variable. The Java compiler gives an error.

2.8 Literals

? Question

How would you correct the code?

2.8 Literals

? Question

How would you correct the code?

- You can write the value as a **float** literal:

```java
float point = 3.1416f;
```

- Alternatively, you can store the value in a **double** variable:

```java
double point = 3.1416d;
```

2.9 Constants

- We just had the example of the circle number π.
- In Java, there is the keyword to define constants.
- These can then no longer be changed.

```java
final double PI = 3.1416;
```

- After a constant has been declared, it can no longer be changed. The following code would therefore generate an error:

```java
PI = 3;
```

2.10 Creating Console Output

Task 1

We now want to create a console output:

- Open IntelliJ IDEA and open or create a new executable class.
- Try the following code:

```java
int age = 24;
System.out.println(24);
System.out.println(age);
```

2.10 Creating Console Output

Task 2

- Using the “+” operator, you can combine text and variables:

```java
int age = 24;
System.out.println("My age is " + 24);
System.out.println("My age is " + age);
```

Tip

- Type `sout` in IntelliJ IDEA and press the Tab key. This saves time when writing `System.out.println()`!

2.11 Coding Style

? Question

What is a **Coding Style**? What does the term tell you?

2.11 Coding Style

? Question

What is a **Coding Style**? What does the term tell you?

- The coding style is a collection of rules that determine how code should be written.
- Uniform code is easier to read and maintain.

! Memorize

Compliance with the coding style will be evaluated in the exam!

2.12 Coding Style: Naming Conventions

- All names, and this applies to all identifiers, should be written in English!
- The following naming conventions should be followed:
  - Classes: **CamelCase**
  - Methods and variables: **camelCase**
  - Constants: **UPPER_CASE**
  - Packages: **lowercase**

> 🔥 **Tip**
>
> From my experience: Make your variables as meaningful as possible! Then the name can also be longer.

3. Comments and Identifiers

3.1 Character Set

- As already mentioned, Java uses the Unicode character set.
- This means more characters are possible (65,536 to be exact).
- So you can write your comments in German, English, or Chinese without major restrictions.
- However, I would ask you to write your comments in **German** or **English**.

Memorize

Since your keyboard doesn't have 65,536 characters, you can also copy and paste the characters. Alternatively for 😀 :

```java
System.out.println("\u{1F600}");
```

3.2 Comments

? Question

What do you think about the following statement? Why are comments important?

Quote

> Make the code readable? Who else is supposed to read this?
>
> — Many Developers

3.2 Comments

- Comments are important for documenting code and improving maintainability.
- Both users of the code and the developers will need to understand the code. Comments are essential for this.

! Memorize

Not the quantity, but the quality of comments is crucial!
Always comment directly while you are programming!

3.2 Comments

? Question

What is the difference between a **block comment** and a **line comment**?

3.2 Comments

? Question

What is the difference between a **block comment** and a **line comment**?

- **Line comments** start with `//` and end at the end of the line.
- **Block comments** start with `/*` and end with `*/`.

3.2 Comments

- Example of a line comment:

```java
// This is a line comment
int distance; // Euclidean distance between a and b
```

- Example of a block comment:

```java
/* The calculation of the Euclidean distance
follows these steps:
  1. Calculate the difference of coordinates
  2. Square the difference
  ...  */
```

3.3 Identifiers

- All things that you name in Java are called **identifiers**. Many things you write need a name!

! Memorize

- Follow these rules for identifiers:
  - Letters, numbers, underscores, and dollar signs are allowed.
  - The first character may not be a number.
  - Case sensitivity is observed.
  - No spaces or keywords.
  - Not the literals `true`, `false`, or `null`.

3.3 Identifiers

- All reserved keywords in Java:

abstract	double	int	super
assert	else	interface	switch
boolean	enum	long	synchronized
break	extends	native	this
byte	final	new	throw
case	finally	package	throws
catch	float	private	transient
char	for	protected	try
class	goto	public	void
const	if	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp	

3.3 Identifiers

? Question

Which of the identifiers are allowed in your opinion and why?

Java

```java
int length;
int länge;
int maxLength;
int max_length;
int _max_length;
int max-length;
int !maxLength;
```

3.3 Identifiers

```
8
9     int 3dlength;
10    String öpnvKosten;
11    String €kosten;
12    String kostenin€
13    String €;
14    int long;
15    int c.o.s.t;
16    String @cost;
```

4. Operators

4.1 Operators

- There are the usual arithmetic operators.
- In general, operators are also evaluated from left to right.

Operator	Name	Example	Priority
`+`	Prefix	`a = 7`	1
`-`	Prefix	`a = -7`	1
`++`	Increment	`++count, count++`	1
`--`	Decrement	`--count, count--`	1
`*`	Multiplication	`area = length * width`	2
`/`	Division	`mean = sum / count`	2
`%`	Modulo	`11 % 4 (ergibt 3)`	2
`+`	Addition	`a = b + c`	3
`-`	Substraction	`a = b - c`	3

Table 1: Value ranges of data types

4.2 Increment and Decrement

- There are also the same operators for incrementing and decrementing as in C.

Operator	Type	Value of Expression	Change of a
`++a`	Prefix	`a + 1`	`a = a + 1`
`a++`	Postfix	`a`	`a = a + 1`
`--a`	Prefix	`a - 7`	`a = a - 1`
`a--`	Postfix	`a`	`a = a - 1`

Table 2: Value ranges of data types

4.2 Increment and Decrement

? Question

Think about it: What will appear on the console here?

Java

```java
int a = 1;
System.out.println("++a : " + ++a);
System.out.println("a   : " + a);
System.out.println("a++ : " + a++);
System.out.println("--a : " + --a);
System.out.println("a-- : " + a--);
```

4.3 Comparison Operators

- There are also the same comparison operators as in C!

Operator	Name	Priority
<	less than	5
<=	less than or equal to	5
>	larger than	5
>=	larger than or equal to	5
==	equal to	6
!=	not equal to	6

Table 3: Value ranges of data types

4.3 Comparison Operators

? Question

Think about it: What happens here?

Java

```java
int a = 7, b = 4;
boolean parentheses = (a > b) == (a <= b);
boolean priorities = a > b == a <= b;
System.out.println(parentheses);
System.out.println(priorities);
```

4.4 Logical Operators

- The result of logical operators is always a truth value, which is represented as `boolean` in Java.

Operator	Name	Priority
!	NOT	1
^	XOR	8
&&	AND	10
||	OR	11

Table 4: Value ranges of data types

4.4 Logical Operators

! Memorize

- With logical operators, the right operand is not executed if the result is already determined. In the following example, `a` is not evaluated.
- Example: `(true || a)`
- This is called **Short Circuit**.
- This becomes interesting when the right operand is, for example, a function/method.

4.4 Logical Operators

? Question

Think about it again: What happens in the following code?

Java

```java
int a = 3, b = 4;
System.out.println((++a == b) || (a++ > b));
System.out.println("a = " + a);
```

4.5 Assignment Operators

- As in C, there are also assignment operators in Java. These can also be combined with other operators.
- The placeholder `<op>` stands for `*`, `/`, `+` and `-`, among others.

Operator	Name	Priority
`=`	Assignment	13
`<op>=`	Combined Assignment: `a <op>= b <=> a = a <op> b`	13

Table 5: Value ranges of data types

4.5 Assignment Operators

? Question

One last time: What happens in this code?

Java

```java
int a = 1;
a += 2;
System.out.println(a);
System.out.println(a *= --a);
System.out.println(a *= -a++);
System.out.println(a /= 10);
```

5. Type Conversion

5.1 Type Conversion

- As a reminder: Type correctness prevents variables from getting a value that does not correspond to their data type.
- This prevents errors and makes the code safer.

> ⚠ **Warning**
>
> However, a variable of type `int` does not fit into a variable of type `byte`. How can you still store the value from `int` in a `byte` variable?

5.1 Type Conversion

Idea

You can simply write that you want this explicitly!

```java
int a = 80;
byte b = (byte) a;
System.out.println(b);
```

5.1 Type Conversion

? Question

What happens in the following code?

Java

```java
double a = 128.38;
int b = (int) a;
byte c = (byte) a;
System.out.println("double: " + a);
System.out.println("int   : " + b);
System.out.println("byte  : " + c);
```

? Question

What happens when you store the value `128` in a `byte` variable?

? Question

What happens when you store the value `128` in a `byte` variable?

- Since the data type can only store values from `-128` to `127`, the value will overflow.
- The result will be a negative number. In this case it will be `-128`.

5.2 Implicit Type Conversion

! Memorize

- Principle of implicit type conversion:
  - No data loss when assigning from a smaller to a larger type.
  - The cast operator is not necessary.
  - Automatic conversion takes place.

5.2 Implicit Type Conversion

Example

- `short` (-32,768 to 32,767) fits into `int` (-2,147,483,648 to 2,147,483,647).

```java
short a = 71;
int b = (int) a;
int c = a;
```

5.2 Implicit Type Conversion

? Question

Think about it: Which of the following lines will compile?

```java
short a = 1024;
long b = a;
float c = b;
```

5.2 Implicit Type Conversion

Java

```java
char d = 'A';
short e = d;
int f = d;
```

5.2 Implicit Type Conversion

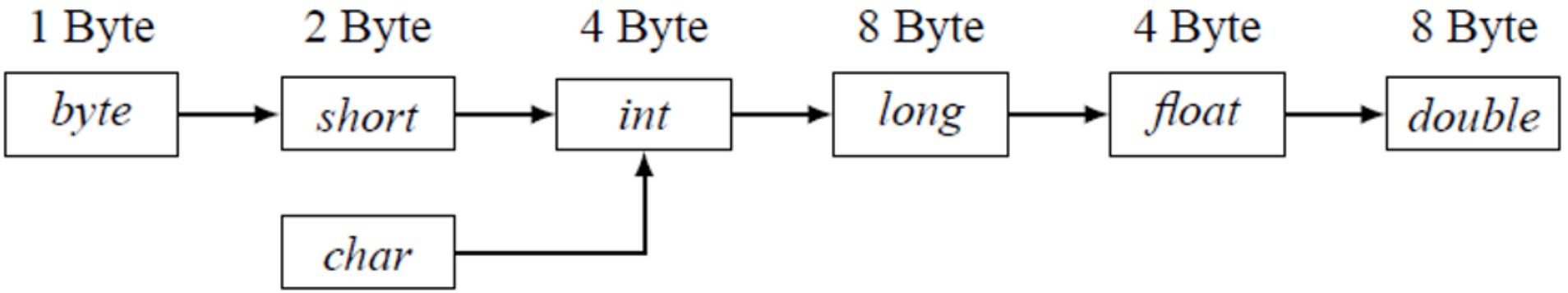


Figure 1: Implicit type conversion in Java

5.2 Implicit Type Conversion

! Memorize

- Integer types `char` and `short` each have 2 bytes, but `char` is an **unsigned** data type.
  - Value range char: 0 to 65,535
  - Value range short: −32,768 to 32,767
- Not all long values can be represented in float (potential data loss!).

6. Control Structures

6.1 if Statement

! Memorize

If statements are the simplest form of control structures. They allow statements to be executed only when a condition is met.

```java
if (condition) {
  statements
}
```

6.1 if Statement

- The condition must always be a `boolean`, unlike in C.
- Statements are only executed when the condition is true (`true`).
- With only one statement, the curly braces can be omitted.

```java
int a = 4, b = 8;
  int maximum = a;

  if (b > maximum) {
    maximum = b;
  }
```

6.2 if-else Statement

Using an `else` statement, you can specify a block that is executed when the condition is not met.

```java
if (condition) {
    statements 1
} else {
    statements 2
}
```

6.2 if-else Statement

Statement 2 in the above example is executed when the condition is `false`.

Java

```java
int a = 4, b = 8;
  int maximum;

  if (a > b) {
    maximum = a;
  } else {
    maximum = b;
  }
```

6.3 The ? Operator

For simple assignment using `if-else` statements, an expression in this form can be used:

```java
(condition) ? expression 1 : expression 2;
```

- Condition `true`: expression 1 is used
- Condition `false`: expression 2 is used

```java
int a = 4, b = 8;
int maximum = (a > b) ? a : b;
```

6.4 if-else Statement

Task 3

- Given is an integer `weekDay` between 1 and 7.
- It corresponds to: 1 = Monday, 2 = Tuesday, 3 = Wednesday, etc.

Generate the following console outputs depending on the value:

- Monday to Friday: “Working”
- Saturday: “Shopping”
- Sunday: “Resting”

6.4 if-else Statement

Example

Java

```java
byte weekDay = 3;

  if (weekDay <= 5) {
    System.out.println("Working");
  } else if (weekDay == 6) {
    System.out.println("Shopping");
  } else if (weekDay == 7) {
    System.out.println("Resting");
  }
```

6.5 switch Statement

With the `switch` statement, `if-else` statements can be simplified.

Java

```java
switch (expression) {
    case value 1:
    statements
    break;
    case value 2:
    ...
    default:
    statements
    }
```

6.5 switch Statement

- Expression is e.g. an integer variable (except type `long`) or a `String` (from Java 7).
- Statements, `break` and `default` are optional.
- Multiple `case` labels directly in succession are allowed.
- Jump to …
  - `case` label, if it has the value of the expression
  - `default`, if no matching `case` label
  - End of `switch` block, if no matching `case` label and no `default`
- From `case` label or `default` continue until `break` or end of `switch` block

6.5 switch Statement

Task 4

Implement a solution for task 3 as a `switch` statement

6.5 switch Statement

Java

```java
switch (weekDay) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        System.out.println("Working");
        break;
    case 6:
        System.out.println("Shopping");
        break;
```

6.5 switch Statement

```
12      case 7:
13          System.out.println("Resting");
14          break;
15      default:
16          System.out.println("I don't know that day...");
17      }
```

6.6 while Loop

With the `while` loop, a statement is executed as long as the condition is `true`.

```java
while (condition) {
    statements
}
```

- If the condition is already false at the beginning, the statement is never executed.
- Also called a **head-controlled** or **rejecting** loop.

6.6 while Loop

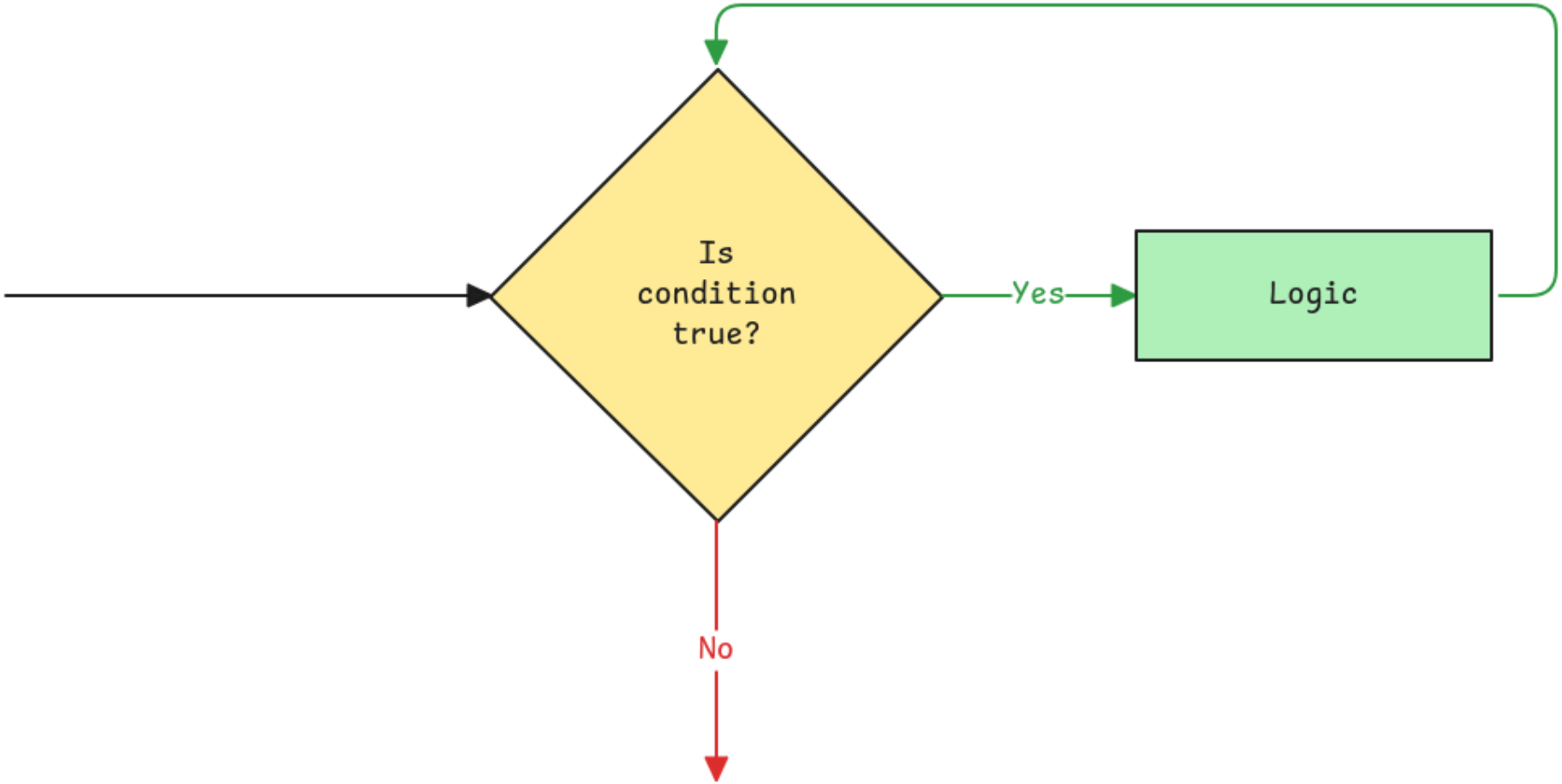


Figure 2: while-Schleife in Java

6.7 do-while Loop

With the `do-while` loop, a statement is executed at least once. If the condition is `true`, the statement is executed again.

Java

```
do {
        statements
    } while (condition);
```

- Also called a **foot-controlled** or **non-rejecting** loop.

6.7 do-while Loop

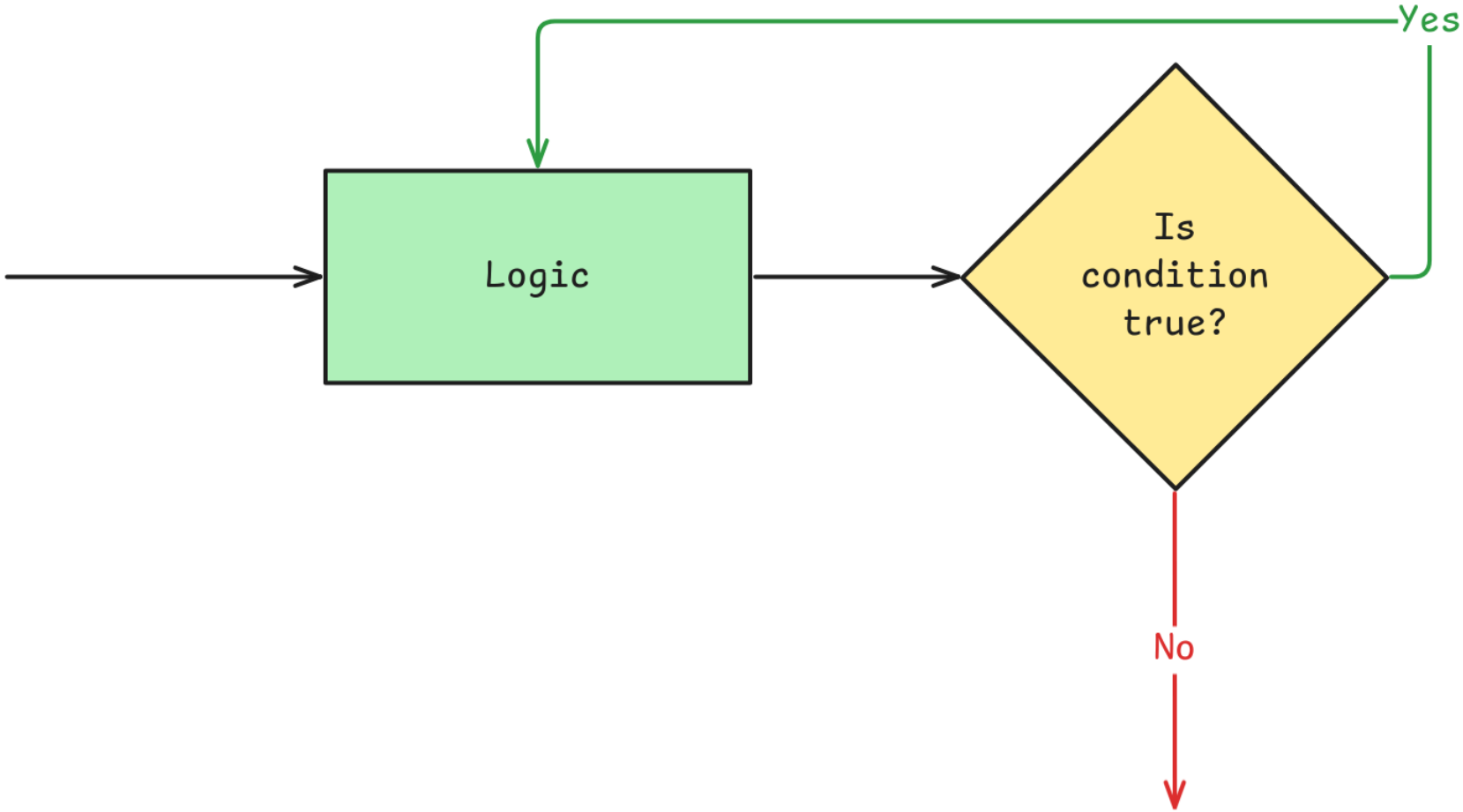


Figure 3: do-while-Schleife in Java

6.8 for Loop

With the `for` loop, you can repeat a statement a certain number of times.

```java
for (init; condition; update) {
    statements
}
```

- If condition is `false`, the statement is never executed.
- Init is executed only once, but always.
- Update is executed after each iteration.

6.8 for Loop

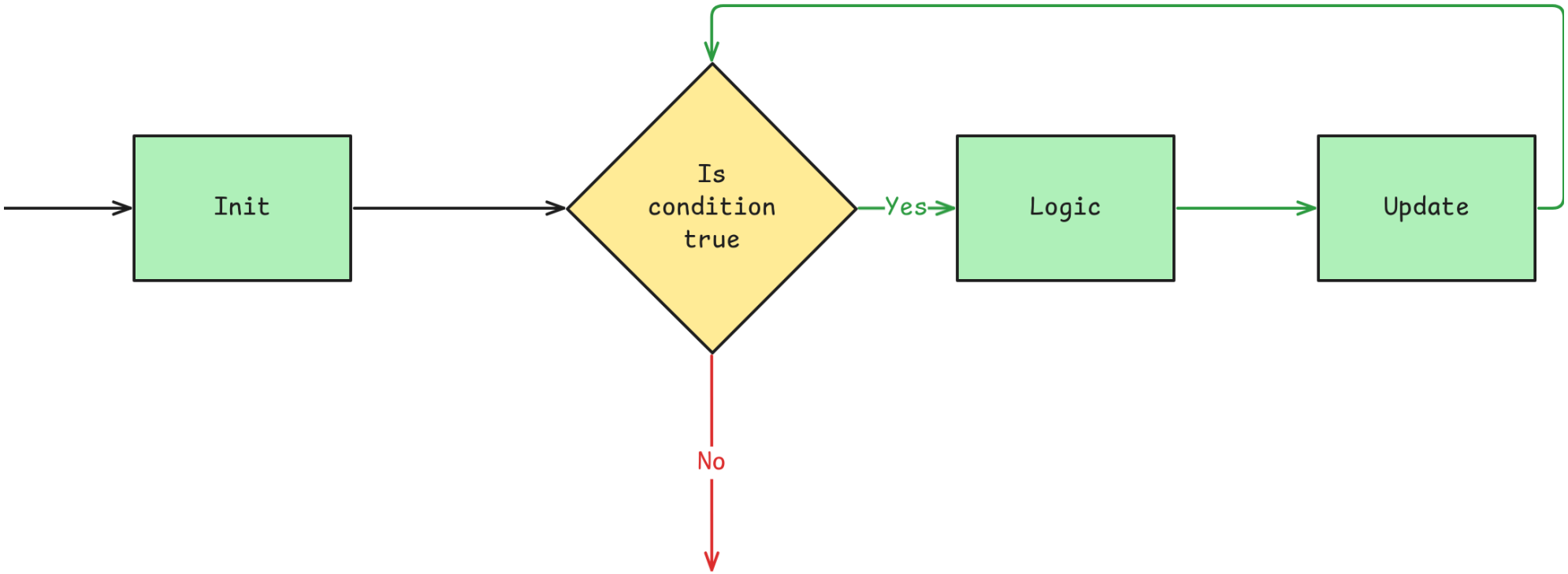


Figure 4: for-Schleife in Java

6.9 Jump Statements

Using jump statements, you can control the program flow. `break` terminates the loop and `continue` jumps to the next loop iteration.

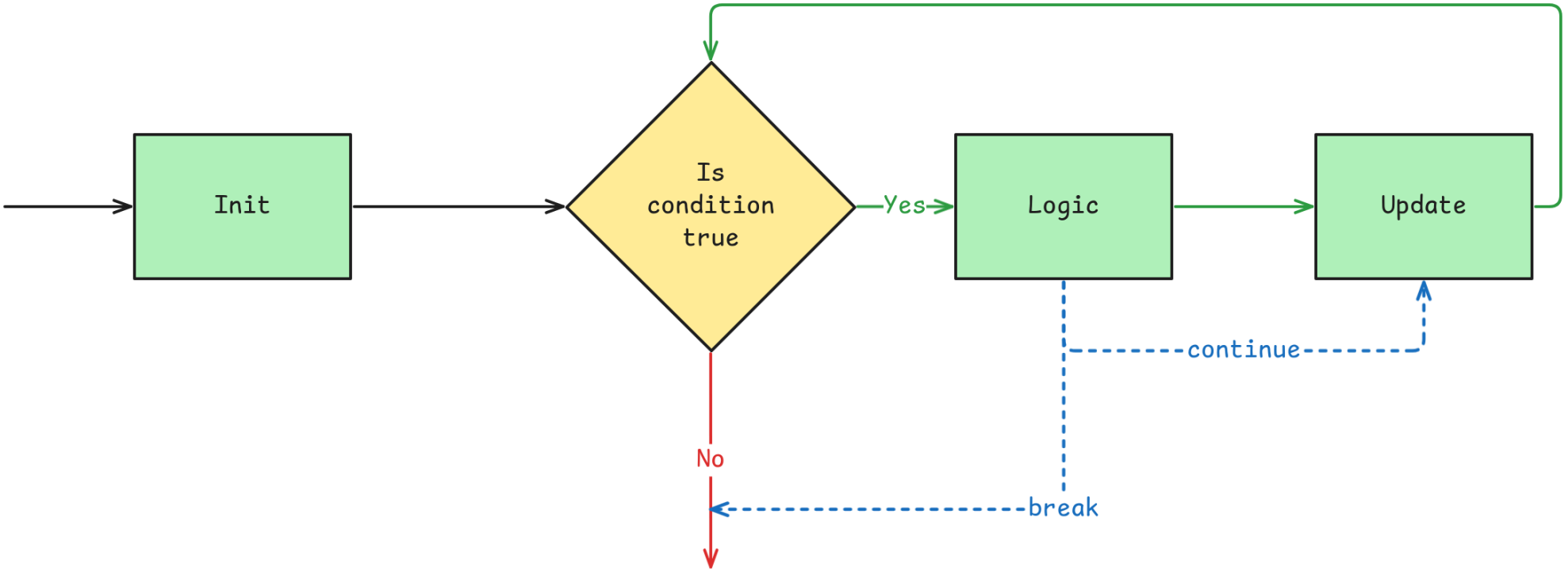


Figure 5: Visualisierung von break und continue in Java

6.9 Jump Statements

? Question

What happens in the following code?

Java

```java
System.out.println("Break (when i == 2):");
for (int i = 0; i <= 4; i++) {
  if (i == 2) {
    break;
  }
  System.out.println("  i = " + i);
}
```

6.9 Jump Statements

? Question

What happens in the following code?

Java

```java
System.out.println("\nContinue (when i == 2):");
for (int i = 0; i <= 4; i++) {
  if (i == 2) {
    continue;
  }
  System.out.println("  i = " + i);
}
```

6.10 Coding Style

- As already mentioned, coding style is important. Therefore, there is also a coding style for control structures.
- Opening curly braces are written on the same line as the control structure (this applies to all opening braces).
- After a closing curly brace, a line break is made. With `else`, the closing brace is on the same line.

6.10 Coding Style

Example

Java

```java
int a = 4, b = 8;
  int maximum;

  if (a > b) {
    maximum = a;
  } else {
    maximum = b;
  }
```

7. License Notice

7.1 Attribution

- This work is shared under the CC BY-NC-SA 4.0 License and the respective Public License
- https://creativecommons.org/licenses/by-nc-sa/4.0/
- This work is based off of the work Prof. Dr. Marc Hensel.
- Some of the images and texts, as well as the layout were changed.
- The base material was supplied in private, therefore the link to the source cannot be shared with the audience.