

Typische Java-Syntaxfehler

Auflistung aus Klausuraufgaben

Autor

Luis Staudt

24. November 2025

Kategorie 1: Typfehler

1. Zuweisung falscher Datentypen

Beschreibung:

Bei der Zuweisung von Werten an Variablen muss der Datentyp des Wertes mit dem Datentyp der Variable kompatibel sein. Java ist eine streng typisierte Sprache und erlaubt keine automatische Konvertierung zwischen inkompatiblen Typen.

Beispiele:

```
1 // Beispiel 1: int zu boolean
2 boolean flag = 42; // FEHLER: int kann nicht zu boolean werden
3
4 // Beispiel 2: String zu int
5 int number = "123"; // FEHLER: String ist kein int
6
7 // Beispiel 3: double zu int ohne Cast
8 int x = 3.14; // FEHLER: double kann nicht ohne Cast zu int werden
9
10 // Beispiel 4: char zu boolean
11 boolean isTrue = 'A'; // FEHLER: char ist kein boolean
12
13 // Beispiel 5: Array zu einzelnen Wert
14 int value = new int[5]; // FEHLER: Array kann nicht zu int werden
```

Worauf achten:

- Links und rechts vom = müssen zusammenpassen
- Zahlen (wie 42) sind keine Wahrheitswerte (true/false)
- Text in Anführungszeichen ist immer ein String
- Kommazahlen (3.14) passen nicht in ganze Zahlen (int)

2. Vergleich inkompatibler Typen

Beschreibung:

Beim Vergleich mit ==, !=, <, >, <=, >= müssen die Operanden kompatible Typen haben.
Bei Objekten (wie String) sollte .equals() statt == verwendet werden.

Beispiele:

```

1 // Beispiel 1: String mit int vergleichen
2 String text = "Hello";
3 int number = 5;
4 if (text == number) { } // FEHLER: String und int sind inkompatibel
5
6 // Beispiel 2: boolean mit int vergleichen
7 boolean flag = true;
8 if (flag == 1) { } // FEHLER: boolean und int sind nicht vergleichbar
9
10 // Beispiel 3: double mit String vergleichen
11 double pi = 3.14;
12 if (pi > "3") { } // FEHLER: double kann nicht mit String verglichen
13      werden
14
15 // Beispiel 4: Array mit einzelnen Wert
16 int[] numbers = {1, 2, 3};
17 if (numbers == 1) { } // FEHLER: Array kann nicht mit int verglichen
18      werden
19
20 // Beispiel 5: char mit String
21 char c = 'A';
22 String s = "A";
23 if (c == s) { } // FEHLER: char und String sind verschiedene Typen

```

Worauf achten:

[leftmargin=*]Man kann nur gleiche Datentypen vergleichen Zahlen mit Zahlen, Text mit Text true/false nur mit true/false vergleichen Bei Text: benutze .equals() statt ==

3. Verwendung von Operatoren mit falschen Typen

Beschreibung:

Arithmetische Operatoren (+, -, *, /, %) funktionieren nur mit numerischen Typen. Logische Operatoren (&&, ||, !) nur mit boolean. Der +-Operator bei Strings ist eine Ausnahme für Konkatenation.

Beispiele:

```
1 // Beispiel 1: boolean addieren
2 boolean a = true;
3 int result = a + 5; // FEHLER: boolean kann nicht addiert werden
4
5 // Beispiel 2: String subtrahieren
6 String text = "Hello";
7 String result = text - "ll"; // FEHLER: - funktioniert nicht mit Strings
8
9 // Beispiel 3: int mit && verknuepfen
10 int x = 5;
11 int y = 10;
12 if (x && y) { } // FEHLER: && braucht boolean-Operanden
13
14 // Beispiel 4: String multiplizieren
15 String s = "abc";
16 String result = s * 3; // FEHLER: * funktioniert nicht mit Strings
17
18 // Beispiel 5: boolean modulo
19 boolean b1 = true;
20 boolean b2 = false;
21 boolean result = b1 % b2; // FEHLER: % funktioniert nicht mit boolean
```

Worauf achten:

[leftmargin=*] Rechnen (+, -, *, /) geht nur mit Zahlen && und || gehen nur mit true/false Text kann man nur mit + zusammenfügen true/false kann man nicht rechnen

4. Methodenaufrufe mit falschen Datentypen

Beschreibung:

Beim Aufruf einer Methode müssen die übergebenen Argumente mit den deklarierten Parametertypen übereinstimmen. Die Reihenfolge und Anzahl müssen ebenfalls passen.

Beispiele:

```
1 // Beispiel 1: int-Methode mit double aufrufen
2 public static boolean isEven(int number) {
3     return number % 2 == 0;
4 }
5 // Aufruf:
6 double x = 4.5;
7 boolean result = isEven(x); // FEHLER: double passt nicht zu int
8
9 // Beispiel 2: String-Methode mit int aufrufen
10 public static void printMessage(String msg) {
11     System.out.println(msg);
12 }
13 // Aufruf:
14 printMessage(42); // FEHLER: int ist kein String
15
16 // Beispiel 3: Mehrere Parameter in falscher Reihenfolge
17 public static void display(String name, int age) {
18     System.out.println(name + " ist " + age);
19 }
20 // Aufruf:
21 display(25, "Anna"); // FEHLER: Reihenfolge ist falsch
22
23 // Beispiel 4: boolean statt int
24 public static int multiply(int a, int b) {
25     return a * b;
26 }
27 // Aufruf:
28 int result = multiply(5, true); // FEHLER: boolean statt int
29
30 // Beispiel 5: Array statt einzelner Wert
31 public static void printNumber(int num) {
32     System.out.println(num);
33 }
34 // Aufruf:
35 int[] numbers = {1, 2, 3};
36 printNumber(numbers); // FEHLER: Array statt int
```

Worauf achten:

[leftmargin=*] Schau dir an, was die Methode erwartet Die Werte müssen in der richtigen Reihenfolge sein Wenn Methode `int` will, gib kein `String` Kommazahlen sind nicht das gleiche wie ganze Zahlen

Kategorie 2: Variablen-Fehler

5. Verwendung nicht deklarierter Variablen

Beschreibung:

Jede Variable muss deklariert werden, bevor sie verwendet wird. Java muss wissen, welcher Datentyp die Variable hat.

Beispiele:

```
1 // Beispiel 1: Variable nie deklariert
2 public static void main(String[] args) {
3     System.out.println(name); // FEHLER: name wurde nie deklariert
4 }
5
6 // Beispiel 2: Tippfehler im Variablenamen
7 int counter = 0;
8 counter++;
9 System.out.println(countr); // FEHLER: Tippfehler (countr statt counter)
10
11 // Beispiel 3: Variable ausserhalb des Scopes
12 if (true) {
13     int localVar = 5;
14 }
15 System.out.println(localVar); // FEHLER: localVar existiert hier nicht
16     mehr
17
18 // Beispiel 4: Falsche Gross-/Kleinschreibung
19 int myValue = 10;
20 System.out.println(MyValue); // FEHLER: Java ist case-sensitive
21
22 // Beispiel 5: Variable in falscher Reihenfolge verwendet
23 System.out.println(result); // FEHLER: result wird erst spaeter deklariert
24 int result = 5;
```

Worauf achten:

- Erst `int x;` schreiben, dann `x` benutzen
- Achte auf Schreibfehler im Namen
- Groß- und Kleinbuchstaben machen einen Unterschied
- Variable muss vorher im Code stehen

6. Verwendung nicht initialisierter Variablen

Beschreibung:

Lokale Variablen müssen initialisiert werden, bevor sie gelesen werden. Anders als Instanzvariablen erhalten lokale Variablen keinen Standardwert.

Beispiele:

```

1 // Beispiel 1: Einfache nicht initialisierte Variable
2 public static void main(String[] args) {
3     int x;
4     int y = x + 5; // FEHLER: x wurde nicht initialisiert
5 }
6
7 // Beispiel 2: Bedingte Initialisierung
8 int value;
9 if (Math.random() > 0.5) {
10     value = 10;
11 }
12 System.out.println(value); // FEHLER: value koennte uninitialisiert sein
13
14 // Beispiel 3: Variable in Schleife
15 int sum;
16 for (int i = 0; i < 10; i++) {
17     sum += i; // FEHLER: sum wurde nicht initialisiert
18 }
19
20 // Beispiel 4: String ohne Initialisierung
21 String message;
22 message = message.toUpperCase(); // FEHLER: message hat keinen Wert
23
24 // Beispiel 5: Array-Element
25 int[] numbers = new int[5];
26 int first = numbers[0]; // OK: Arrays werden mit 0 initialisiert
27 int standalone;
28 int test = standalone; // FEHLER: lokale Variable nicht initialisiert

```

Worauf achten:

- Schreibe `int x = 0;` statt nur `int x;`
- Die Variable braucht einen Startwert
- Java gibt dir keinen automatischen Wert
- Bei `if/else`: beide Wege müssen einen Wert geben

7. Zugriff auf nicht-statische Variablen aus statischem Kontext

Beschreibung:

Statische Methoden (wie main) können nicht direkt auf nicht-statische (Instanz-) Variablen zugreifen, da diese zu einem Objekt gehören, das möglicherweise nicht existiert.

Beispiele:

```
1 // Beispiel 1: Direkte Verwendung in main
2 public class Test {
3     int instanceVar = 10;
4
5     public static void main(String[] args) {
6         System.out.println(instanceVar); // FEHLER: nicht-static in static
7     }
8 }
9
10 // Beispiel 2: In statischer Methode
11 public class Calculator {
12     double result = 0;
13
14     public static void reset() {
15         result = 0; // FEHLER: result ist nicht static
16     }
17 }
18
19 // Beispiel 3: String-Variablen
20 public class Program {
21     String name = "Test";
22
23     public static void printInfo() {
24         System.out.println(name); // FEHLER: name ist nicht static
25     }
26 }
27
28 // Beispiel 4: Array-Variablen
29 public class Data {
30     int[] numbers = {1, 2, 3};
31
32     public static void process() {
33         int first = numbers[0]; // FEHLER: numbers ist nicht static
34     }
35 }
36
37 // Beispiel 5: boolean-Variablen
38 public class App {
```

```
39 boolean isActive = true;  
40  
41 public static void check() {  
42     if (isActive) {} // FEHLER: isActive ist nicht static  
43 }  
44 }
```

Worauf achten:

- **static** bedeutet: gehört zur Klasse selbst
- Ohne **static**: gehört zu einem einzelnen Objekt
- In **main** kannst du nur **static** Sachen direkt benutzen
- Lösung: Schreibe **static** vor die Variable

8. Zugriff auf lokale Variablen außerhalb ihres Gültigkeitsbereichs

Beschreibung:

Variablen, die in einem Block (z.B. if, while, for) deklariert werden, existieren nur innerhalb dieses Blocks. Nach der schließenden Klammer sind sie nicht mehr verfügbar.

Beispiele:

```
1 // Beispiel 1: if-Block
2 public static void main(String[] args) {
3     if (true) {
4         int x = 5;
5     }
6     System.out.println(x); // FEHLER: x existiert hier nicht
7 }
8
9 // Beispiel 2: for-Schleife
10 public static void main(String[] args) {
11     for (int i = 0; i < 10; i++) {
12         int sum = i * 2;
13     }
14     System.out.println(sum); // FEHLER: sum existiert hier nicht
15 }
16
17 // Beispiel 3: while-Schleife
18 public static void main(String[] args) {
19     while (true) {
20         String message = "Loop";
21         break;
22     }
23     System.out.println(message); // FEHLER: message existiert hier nicht
24 }
25
26 // Beispiel 4: Verschachtelte Blöcke
27 public static void main(String[] args) {
28     {
29         int temp = 100;
30     }
31     int value = temp; // FEHLER: temp existiert hier nicht
32 }
33
34 // Beispiel 5: Switch-Case
35 public static void main(String[] args) {
36     int choice = 1;
37     switch (choice) {
38         case 1:
```

```
39         String result = "One";
40         break;
41     }
42     System.out.println(result); // FEHLER: result existiert hier nicht
43 }
```

Worauf achten:

- Variablen leben nur zwischen { und }
- Nach der } ist die Variable weg
- Wenn du sie später brauchst: Schreibe sie vor die {
- Schleifenvariablen wie i existieren nur in der Schleife

Kategorie 3: Methodenfehler

9. Aufruf nicht-statischer Methoden aus statischem Kontext

Beschreibung:

Genau wie bei Variablen können statische Methoden nicht direkt auf nicht-statische Methoden zugreifen, da diese zu einem Objekt gehören müssen.

Beispiele:

```
1 // Beispiel 1: In main-Methode
2 public class Test {
3     public void printMessage() {
4         System.out.println("Hello");
5     }
6
7     public static void main(String[] args) {
8         printMessage(); // FEHLER: printMessage ist nicht static
9     }
10 }
11
12 // Beispiel 2: Von statischer Hilfsmethode
13 public class Calculator {
14     public int add(int a, int b) {
15         return a + b;
16     }
17
18     public static int calculate() {
19         return add(5, 3); // FEHLER: add ist nicht static
20     }
21 }
22
23 // Beispiel 3: Getter-Methode
24 public class Person {
25     private String name = "Max";
26
27     public String getName() {
28         return name;
29     }
30 }
```

```
31 public static void displayName() {
32     System.out.println(getName()); // FEHLER: getName ist nicht static
33 }
34 }
35
36 // Beispiel 4: Setter-Methode
37 public class Counter {
38     private int count = 0;
39
40     public void increment() {
41         count++;
42     }
43
44     public static void resetAndIncrement() {
45         increment(); // FEHLER: increment ist nicht static
46     }
47 }
48
49 // Beispiel 5: Void-Methode aufrufen
50 public class Helper {
51     public void doSomething() {
52         System.out.println("Doing...");
```

Worauf achten:

- Gleiche Regel wie bei Variablen
- Ohne **static** brauchst du ein Objekt
- In **main**: schreibe **static** vor die Methode
- Oder erstelle erst ein Objekt: `new Test().printMessage();`

10. Falsche Anzahl von Parametern bei Methodenaufrufen

Beschreibung:

Die Anzahl der übergebenen Argumente muss exakt mit der Anzahl der deklarierten Parameter übereinstimmen (außer bei Varargs).

Beispiele:

```
1 // Beispiel 1: Zu wenige Parameter
2 public static int add(int a, int b) {
3     return a + b;
4 }
5 // Aufruf:
6 int result = add(5); // FEHLER: 2 Parameter erwartet, nur 1 uebergeben
7
8 // Beispiel 2: Zu viele Parameter
9 public static void printName(String name) {
10     System.out.println(name);
11 }
12 // Aufruf:
13 printName("Max", "Mustermann"); // FEHLER: 1 Parameter erwartet, 2
14 uebergeben
15
16 // Beispiel 3: Keine Parameter erwartet
17 public static void greet() {
18     System.out.println("Hello!");
19 }
20 // Aufruf:
21 greet("World"); // FEHLER: 0 Parameter erwartet, 1 uebergeben
22
23 // Beispiel 4: Drei Parameter, zwei uebergeben
24 public static double calculate(int a, int b, int c) {
25     return (a + b) * c;
26 }
27 // Aufruf:
28 double result = calculate(5, 10); // FEHLER: 3 Parameter erwartet
29
30 // Beispiel 5: Ein Parameter fehlt
31 public static boolean isInRange(int value, int min, int max) {
32     return value >= min && value <= max;
33 }
34 // Aufruf:
35 boolean check = isInRange(50, 0); // FEHLER: 3 Parameter erwartet, 2
36 uebergeben
```

Worauf achten:

- Zähle die Werte in den Klammern bei der Methodendefinition
- Gib beim Aufruf genauso viele Werte
- Jeder Wert in der Definition braucht einen beim Aufruf
- Fehlt einer? Java beschwert sich!

11. Falsche Parametertypen bei Methodenaufrufen

Beschreibung:

Die Typen der übergebenen Argumente müssen mit den Parametertypen der Methode übereinstimmen oder kompatibel sein.

Beispiele:

```
1 // Beispiel 1: String statt int
2 public static void printNumber(int num) {
3     System.out.println("Number: " + num);
4 }
5 // Aufruf:
6 printNumber("42"); // FEHLER: String statt int
7
8 // Beispiel 2: int statt boolean
9 public static void setFlag(boolean flag) {
10    System.out.println("Flag: " + flag);
11 }
12 // Aufruf:
13 setFlag(1); // FEHLER: int statt boolean
14
15 // Beispiel 3: double statt String
16 public static void displayMessage(String message) {
17    System.out.println(message);
18 }
19 // Aufruf:
20 displayMessage(3.14); // FEHLER: double statt String
21
22 // Beispiel 4: Array statt einzelner Wert
23 public static int square(int x) {
24     return x * x;
25 }
26 // Aufruf:
27 int[] numbers = {2, 3, 4};
28 int result = square(numbers); // FEHLER: Array statt int
29
30 // Beispiel 5: Mehrere Parameter mit falschen Typen
31 public static void register(String name, int age, boolean active) {
32     System.out.println(name + ", " + age + ", " + active);
33 }
34 // Aufruf:
35 register(25, "Max", 1); // FEHLER: alle Typen falsch
```

Worauf achten:

- Schau nach: will die Methode eine Zahl oder Text?
- Zahlen in Anführungszeichen ("42") sind Text
- true/false sind keine Zahlen (nicht 1 oder 0)
- Die Typen müssen genau passen

12. Fehlende return-Anweisungen in Methoden mit Rückgabewert

Beschreibung:

Wenn eine Methode einen Rückgabetyp hat (nicht void), muss sie in jedem möglichen Ausführungspfad einen Wert zurückgeben.

Beispiele:

```
1 // Beispiel 1: Komplett fehlendes return
2 public static int getValue() {
3     int x = 5;
4     // FEHLER: kein return
5 }
6
7 // Beispiel 2: return nur in if, nicht in else
8 public static int getNumber(boolean condition) {
9     if (condition) {
10         return 10;
11     }
12     // FEHLER: was wenn condition false ist?
13 }
14
15 // Beispiel 3: return nach unerreichbarem Code
16 public static double calculate() {
17     if (true) {
18         return 3.14;
19     }
20     // Dieser Code ist unerreichbar, aber return fehlt formal
21 }
22
23 // Beispiel 4: return in Schleife, aber nicht ausserhalb
24 public static String findName() {
25     for (int i = 0; i < 10; i++) {
26         if (i == 5) {
27             return "Found";
28         }
29     }
30     // FEHLER: was wenn die Schleife durchlaeuft ohne return?
31 }
32
33 // Beispiel 5: switch ohne default
34 public static int getDays(int month) {
35     switch (month) {
36         case 1: return 31;
37         case 2: return 28;
38         // ...
39     }
40 }
```

```
39     }
40     // FEHLER: was bei ungultigen month?
41 }
```

Worauf achten:

- Methode gibt was zurück? Dann brauchst du **return**
- Jeder mögliche Weg muss zu einem **return** führen
- Auch wenn nur vielleicht kein **return**: Fehler!
- Am sichersten: **return** am Ende der Methode

13. Nicht alle Codepfade führen zu einem return

Beschreibung:

Der Compiler analysiert alle möglichen Wege durch den Code. Wenn auch nur ein Pfad ohne return existiert, gibt es einen Fehler.

Beispiele:

```
1 // Beispiel 1: if ohne else
2 public static int getResult(boolean flag) {
3     if (flag) {
4         return 1;
5     }
6     // FEHLER: wenn flag false ist, kein return
7 }
8
9 // Beispiel 2: mehrere if ohne finales return
10 public static String getGrade(int points) {
11     if (points >= 90) {
12         return "A";
13     }
14     if (points >= 80) {
15         return "B";
16     }
17     // FEHLER: was bei points < 80?
18 }
19
20 // Beispiel 3: while-Schleife
21 public static int findValue() {
22     int i = 0;
23     while (i < 10) {
24         if (i == 5) {
25             return i;
26         }
27         i++;
28     }
29     // FEHLER: was wenn 5 nie erreicht wird?
30 }
31
32 // Beispiel 4: try-catch ohne return ausserhalb
33 public static int parseValue(String s) {
34     try {
35         return Integer.parseInt(s);
36     } catch (Exception e) {
37         System.out.println("Error");
38     }
}
```

```
39 // FEHLER: im catch-Block fehlt return
40 }
41
42 // Beispiel 5: Verschachtelte Bedingungen
43 public static double compute(int a, int b) {
44     if (a > 0) {
45         if (b > 0) {
46             return a + b;
47         }
48     }
49     // FEHLER: mehrere Pfade ohne return
50 }
```

Worauf achten:

- Java prüft alle if/else Möglichkeiten
- Mindestens ein Weg ohne **return**? Fehler!
- Lösung: **return** ganz am Ende einfügen
- Oder: zu jedem **if** ein **else** mit **return**

14. Rückgabe eines falschen Datentyps

Beschreibung:

Der Typ des zurückgegebenen Wertes muss exakt mit dem deklarierten Rückgabetyp der Methode übereinstimmen (oder kompatibel sein).

Beispiele:

```

1 // Beispiel 1: int statt String
2 public static String getNumber() {
3     return 42; // FEHLER: int wird zurueckgegeben, String erwartet
4 }
5
6 // Beispiel 2: String statt int
7 public static int getValue() {
8     return "123"; // FEHLER: String statt int
9 }
10
11 // Beispiel 3: double statt boolean
12 public static boolean check() {
13     return 1.0; // FEHLER: double statt boolean
14 }
15
16 // Beispiel 4: Array statt einzelner Wert
17 public static int getFirst() {
18     int[] numbers = {1, 2, 3};
19     return numbers; // FEHLER: Array statt int
20 }
21
22 // Beispiel 5: boolean statt int
23 public static int calculate() {
24     return true; // FEHLER: boolean statt int
25 }
```

Worauf achten:

- Schau dir den Typ vor dem Methodennamen an
- Das Gleiche muss nach **return** kommen
- **int** → gib eine Zahl zurück
- **String** → gib Text zurück
- **boolean** → gib true oder false zurück

15. Rückgabe eines Wertes in einer Methode ohne Rückgabe (`void`)

Beschreibung:

Methoden mit dem Rückgabetyp `void` dürfen keinen Wert zurückgeben. Sie können nur `return;` ohne Wert verwenden (optional).

Beispiele:

```
1 // Beispiel 1: String zurueckgeben
2 public static void printMessage() {
3     return "Hello"; // FEHLER: void-Methode kann keinen Wert zurueckgeben
4 }
5
6 // Beispiel 2: int zurueckgeben
7 public static void calculate() {
8     int result = 5 * 5;
9     return result; // FEHLER: void kann keinen int zurueckgeben
10 }
11
12 // Beispiel 3: boolean zurueckgeben
13 public static void checkValue(int x) {
14     if (x > 0) {
15         return true; // FEHLER: void kann keinen boolean zurueckgeben
16     }
17 }
18
19 // Beispiel 4: double zurueckgeben
20 public static void computeAverage(int a, int b) {
21     double avg = (a + b) / 2.0;
22     return avg; // FEHLER: void kann keinen double zurueckgeben
23 }
24
25 // Beispiel 5: Object zurueckgeben
26 public static void createObject() {
27     Object obj = new Object();
28     return obj; // FEHLER: void kann kein Object zurueckgeben
29 }
```

Worauf achten:

- `void` bedeutet: gibt nichts zurück
- Kein `return` mit Wert erlaubt
- Willst du was zurückgeben? Ändere `void` zu `int`, `String`, etc.
- `return;` ohne Wert ist ok (zum Beenden)

16. Verschachtelte Methodendefinitionen

Beschreibung:

In Java können Methoden nicht innerhalb anderer Methoden definiert werden. Jede Methode muss direkt in einer Klasse definiert sein.

Beispiele:

```

1 // Beispiel 1: Methode in Methode
2 public static void outerMethod() {
3     public static void innerMethod() { // FEHLER: nicht verschachtelt
4         System.out.println("Inner");
5     }
6 }
7
8 // Beispiel 2: Hilfsmethode innerhalb
9 public static int calculate(int x) {
10    public static int helper(int y) { // FEHLER: nicht erlaubt
11        return y * 2;
12    }
13    return helper(x);
14 }
15
16 // Beispiel 3: main mit innerer Methode
17 public static void main(String[] args) {
18     public static void greet() { // FEHLER: nicht innerhalb main
19         System.out.println("Hello");
20     }
21     greet();
22 }
23
24 // Beispiel 4: Mehrfach verschachtelt
25 public static void methodA() {
26     public static void methodB() { // FEHLER
27         public static void methodC() { // FEHLER
28             System.out.println("C");
29         }
30     }
31 }
32
33 // Beispiel 5: return-Typ-Methode verschachtelt
34 public static int getValue() {
35     public static int compute() { // FEHLER
36         return 42;
37     }

```

```
38     return compute();  
39 }
```

Worauf achten:

- Methoden gehören in die Klasse, nicht in andere Methoden
- Alle Methoden auf der gleichen Ebene schreiben
- Einrückung hilft: Methoden auf gleicher Einrückung wie `main`
- Jede Methode zwischen Klassen-{ und Klassen-}

Kategorie 4: Kontrollstruktur-Fehler

17. Verwendung von nicht-boolean Ausdrücken in if-Bedingungen

Beschreibung:

Die Bedingung in einer if-Anweisung muss immer einen boolean-Wert (true oder false) ergeben. Zahlen oder andere Typen sind nicht erlaubt.

Beispiele:

```
1 // Beispiel 1: int statt boolean
2 int x = 5;
3 if (x) { // FEHLER: x ist int, nicht boolean
4     System.out.println("true");
5 }
6
7 // Beispiel 2: String-Bedingung
8 String name = "Max";
9 if (name) { // FEHLER: String ist nicht boolean
10    System.out.println("Name exists");
11 }
12
13 // Beispiel 3: Zuweisung statt Vergleich
14 int value = 10;
15 if (value = 5) { // FEHLER: = ist Zuweisung, nicht Vergleich (sollte ==
16    System.out.println("Five");
17 }
18
19 // Beispiel 4: Null-Wert
20 Object obj = null;
21 if (obj) { // FEHLER: Object ist nicht boolean
22     System.out.println("Exists");
23 }
24
25 // Beispiel 5: Arithmetischer Ausdruck
26 int a = 10, b = 20;
27 if (a + b) { // FEHLER: a + b ist int (30), nicht boolean
28     System.out.println("Sum");
29 }
```

Worauf achten:

- Nach `if` muss etwas mit `true/false` kommen
- Schreibe `if (x > 0)` statt `if (x)`
- Ein `=` setzt einen Wert, `==` vergleicht
- Zahlen sind nicht automatisch `true` oder `false`

18. Verwendung von nicht-boolean Ausdrücken in while-Schleifen

Beschreibung:

Genau wie bei if-Anweisungen muss die Bedingung einer while-Schleife einen boolean-Wert ergeben.

Beispiele:

```
1 // Beispiel 1: int statt boolean
2 int count = 5;
3 while (count) { // FEHLER: count ist int, nicht boolean
4     System.out.println(count);
5     count--;
6 }
7
8 // Beispiel 2: Zahl als Bedingung
9 while (10) { // FEHLER: 10 ist kein boolean
10    System.out.println("Loop");
11 }
12
13 // Beispiel 3: String-Bedingung
14 String text = "running";
15 while (text) { // FEHLER: String ist nicht boolean
16     System.out.println(text);
17 }
18
19 // Beispiel 4: Zuweisung statt Vergleich
20 int x = 0;
21 while (x = 5) { // FEHLER: = statt == (und dann ist es auch noch int)
22     x++;
23 }
24
25 // Beispiel 5: Array als Bedingung
26 int[] numbers = {1, 2, 3};
27 while (numbers) { // FEHLER: Array ist nicht boolean
28     System.out.println("Array");
29 }
```

Worauf achten:

- Gleiche Regel wie bei `if`
- Schreibe `while (count > 0)` statt `while (count)`
- `while (true)` für Endlosschleife ist richtig
- Achte auf `=` (setzen) vs `==` (vergleichen)

Kategorie 5: Zugriffsmodifikator-Fehler

19. Zugriff auf private Variablen von außen

Beschreibung:

Private Variablen und Methoden können nur innerhalb derselben Klasse verwendet werden.
Von außen (andere Klassen) sind sie nicht sichtbar.

Beispiele:

```

1 // Beispiel 1: Einfacher Zugriff von aussen
2 class MyClass {
3     private int secret = 42;
4 }
5
6 public class Test {
7     public static void main(String[] args) {
8         MyClass obj = new MyClass();
9         System.out.println(obj.secret); // FEHLER: secret ist private
10    }
11 }
12
13 // Beispiel 2: Private String
14 class Person {
15     private String password = "1234";
16 }
17
18 public class App {
19     public static void main(String[] args) {
20         Person p = new Person();
21         String pw = p.password; // FEHLER: password ist private
22     }
23 }
24
25 // Beispiel 3: Private Methode
26 class Calculator {
27     private int multiply(int a, int b) {
28         return a * b;
29     }
30 }
```

```

31
32 public class Main {
33     public static void main(String[] args) {
34         Calculator calc = new Calculator();
35         int result = calc.multiply(5, 3); // FEHLER: multiply ist private
36     }
37 }
38
39 // Beispiel 4: Private Array
40 class DataHolder {
41     private int[] data = {1, 2, 3};
42 }
43
44 public class Processor {
45     public static void main(String[] args) {
46         DataHolder holder = new DataHolder();
47         int first = holder.data[0]; // FEHLER: data ist private
48     }
49 }
50
51 // Beispiel 5: Änderung von private Variable
52 class Counter {
53     private int count = 0;
54 }
55
56 public class Program {
57     public static void main(String[] args) {
58         Counter c = new Counter();
59         c.count = 10; // FEHLER: count ist private
60     }
61 }
```

Worauf achten:

- **private** = nur in der eigenen Klasse nutzbar
- Von außen nicht sichtbar
- Lösung: Getter/Setter Methoden benutzen
- Oder: **private** durch **public** ersetzen

20. Zugriff auf nicht-statische Elemente aus statischem Kontext

Beschreibung:

Dies ist ein Sonderfall von #7 und #9, aber so häufig, dass er nochmal erwähnt wird: Statische Methoden können nicht direkt auf Instanzvariablen oder -methoden zugreifen.

Beispiele:

```
1 // Beispiel 1: main greift auf Instanzvariable zu
2 public class Test {
3     int number = 10;
4
5     public static void main(String[] args) {
6         System.out.println(number); // FEHLER: number ist nicht static
7     }
8 }
9
10 // Beispiel 2: Static Methode ruft Instanzmethode auf
11 public class Helper {
12     public void help() {
13         System.out.println("Helping...");
14     }
15
16     public static void doWork() {
17         help(); // FEHLER: help ist nicht static
18     }
19 }
20
21 // Beispiel 3: String-Variable in main
22 public class App {
23     String message = "Hello";
24
25     public static void main(String[] args) {
26         System.out.println(message); // FEHLER: message ist nicht static
27     }
28 }
29
30 // Beispiel 4: Instanzvariable in statischer Hilfsmethode
31 public class Calculator {
32     double result = 0;
33
34     public static void reset() {
35         result = 0; // FEHLER: result ist nicht static
36     }
37
38     public static void main(String[] args) {
```

```
39         reset();
40     }
41 }
42
43 // Beispiel 5: Array-Zugriff
44 public class Data {
45     int[] values = {1, 2, 3, 4, 5};
46
47     public static void printFirst() {
48         System.out.println(values[0]); // FEHLER: values ist nicht static
49     }
50 }
```

Worauf achten:

- Ist in **main**? Dann nur **static** Sachen benutzen
- Lösung: Schreibe **static** vor die Variable
- Oder: Erstelle ein Objekt mit **new**
- **main** ist immer static - häufigster Fehler!

Kategorie 6: Array-Fehler

21. Falsche Array-Deklaration

Beschreibung:

Die Größe eines Arrays wird bei der Deklaration nicht angegeben, sondern erst bei der Initialisierung. Die eckigen Klammern gehören zum Typ, nicht zur Variable.

Beispiele:

```

1 // Beispiel 1: Groesse bei Deklaration
2 int[5] numbers; // FEHLER: Groesse nicht bei Deklaration
3
4 // Korrekt waere:
5 int[] numbers = new int[5];
6 // oder
7 int[] numbers;
8 numbers = new int[5];
9
10 // Beispiel 2: Klammern am falschen Ort
11 int numbers[5]; // FEHLER: Groesse nicht erlaubt
12
13 // Beispiel 3: Falsche Syntax mit String
14 String[10] names; // FEHLER: Groesse nicht bei Deklaration
15
16 // Beispiel 4: Mehrere Dimensionen falsch
17 int[3][4] matrix; // FEHLER: Groessen nicht bei Deklaration
18
19 // Beispiel 5: Boolean-Array
20 boolean[20] flags; // FEHLER: Groesse nicht hier

```

Worauf achten:

- Erst sagen WAS: `int[] numbers;`
- Dann sagen WIE VIELE: `numbers = new int[5];`
- Nie Zahlen in die `[]` bei der Deklaration
- Die Zahl kommt später beim `new`

22. Falsche Array-Initialisierung

Beschreibung:

Bei der Array-Initialisierung mit Werten darf keine Größe angegeben werden, wenn man die Werte in geschweiften Klammern auflistet.

Beispiele:

```
1 // Beispiel 1: Groesse und Werte gleichzeitig
2 int[] numbers = new int[3] {1, 2, 3}; // FEHLER: entweder Groesse oder
   Werte
3
4 // Korrekt waere:
5 int[] numbers = new int[3]; // Groesse ohne Werte
6 // oder
7 int[] numbers = {1, 2, 3}; // Werte ohne Groesse/new
8 // oder
9 int[] numbers = new int[] {1, 2, 3}; // new ohne Groesse
10
11 // Beispiel 2: String-Array
12 String[] names = new String[2] {"Max", "Anna"}; // FEHLER
13
14 // Korrekt:
15 String[] names = {"Max", "Anna"};
16
17 // Beispiel 3: Double-Array
18 double[] values = new double[4] {1.1, 2.2, 3.3, 4.4}; // FEHLER
19
20 // Korrekt:
21 double[] values = {1.1, 2.2, 3.3, 4.4};
22
23 // Beispiel 4: Boolean-Array
24 boolean[] flags = new boolean[3] {true, false, true}; // FEHLER
25
26 // Korrekt:
27 boolean[] flags = {true, false, true};
28
29 // Beispiel 5: Char-Array
30 char[] letters = new char[5] {'a', 'b', 'c', 'd', 'e'}; // FEHLER
31
32 // Korrekt:
33 char[] letters = {'a', 'b', 'c', 'd', 'e'};
```

Worauf achten:

- Entweder Größe ODER Werte angeben
- Mit Werten: {1, 2, 3} - Java zählt selbst
- Mit Größe: new int[5] - leer, füllst du später
- Beides zusammen geht nicht!

Kategorie 7: Klammer und Syntax-Fehler

23. Fehlende geschweifte Klammern

Beschreibung:

Jede öffnende geschweifte Klammer { braucht eine schließende }. Dies betrifft Klassen, Methoden, Schleifen und Bedingungen.

Beispiele:

```
1 // Beispiel 1: Methode nicht geschlossen
2 public class Test {
3     public static void main(String[] args) {
4         System.out.println("Hello");
5         // FEHLER: } fehlt fuer main
6
7     public static void other() {
8         System.out.println("Other");
9     }
10
11
12 // Beispiel 2: if-Block nicht geschlossen
13 public static void check(int x) {
14     if (x > 0) {
15         System.out.println("Positive");
16         // FEHLER: } fehlt fuer if
17     }
18
19 // Beispiel 3: Verschachtelte Blöcke
20 public static void complex() {
21     for (int i = 0; i < 10; i++) {
22         if (i % 2 == 0) {
23             System.out.println(i);
24             // FEHLER: } fehlt fuer if
25     }
26 }
27
28 // Beispiel 4: Klasse nicht geschlossen
29 public class MyClass {
30     private int value;
```

```
31
32     public void setValue(int v) {
33         this.value = v;
34     }
35     // FEHLER: } fehlt fuer Klasse
36
37 // Beispiel 5: while-Schleife
38 public static void loop() {
39     int i = 0;
40     while (i < 5) {
41         System.out.println(i);
42         i++;
43         // FEHLER: } fehlt
44 }
```

Worauf achten:

- Jede { braucht eine passende }
- Zähle die Klammern: gleich viele auf und zu?
- Bei Verwirrung: jede Methode einzeln prüfen
- IDEs zeigen oft an, wo eine } fehlt

24. Fehlende Semikolons

Beschreibung:

Jede Anweisung (Statement) in Java muss mit einem Semikolon ; enden. Ausnahmen sind Klassen-, Methoden- und Block-Deklarationen.

Beispiele:

```
1 // Beispiel 1: Variable ohne Semikolon
2 public static void main(String[] args) {
3     int x = 5 // FEHLER: ; fehlt
4     System.out.println(x);
5 }
6
7 // Beispiel 2: Methodenaufruf
8 public static void test() {
9     System.out.println("Hello") // FEHLER: ; fehlt
10    System.out.println("World");
11 }
12
13 // Beispiel 3: Return-Statement
14 public static int getValue() {
15     return 42 // FEHLER: ; fehlt
16 }
17
18 // Beispiel 4: Zuweisung
19 public static void calculate() {
20     int a = 10;
21     int b = 20;
22     int sum = a + b // FEHLER: ; fehlt
23     System.out.println(sum);
24 }
25
26 // Beispiel 5: Mehrere Anweisungen
27 public static void demo() {
28     int x = 1 // FEHLER: ; fehlt
29     int y = 2 // FEHLER: ; fehlt
30     int z = x + y // FEHLER: ; fehlt
31 }
```

Worauf achten:

- Fast jede Zeile mit Code braucht ; am Ende
- Ausnahme: {, }, if, while etc.
- Nach = kommt erst der Wert, dann ;
- Vergiss nie das ; am Zeilenende!

25. Fehlende runde Klammern

Beschreibung:

Kontrollstrukturen (if, while, for) und Methodenaufrufe benötigen runde Klammern. Die Bedingung oder Parameter müssen in () stehen.

Beispiele:

```
1 // Beispiel 1: if ohne Klammern
2 public static void check(boolean flag) {
3     if flag { // FEHLER: () fehlen
4         System.out.println("True");
5     }
6 }
7
8 // Korrekt:
9 if (flag) {
10     System.out.println("True");
11 }
12
13 // Beispiel 2: while ohne Klammern
14 public static void loop() {
15     int i = 0;
16     while i < 10 { // FEHLER: () fehlen
17         i++;
18     }
19 }
20
21 // Beispiel 3: for-Schleife
22 public static void iterate() {
23     for int i = 0; i < 5; i++ { // FEHLER: () fehlen um gesamte for-
Deklaration
24         System.out.println(i);
25     }
26 }
27
28 // Beispiel 4: Methodenaufruf
29 public static void greet() {
30     System.out.println; // FEHLER: () fehlen
31 }
32
33 // Beispiel 5: Verschachtelte Bedingung
34 public static void complex(int x, int y) {
35     if x > 0 && y > 0 { // FEHLER: () fehlen
36         System.out.println("Both positive");
37 }
```

38 }

Worauf achten:

- **if, while, for:** Bedingung in () setzen
- Methoden aufrufen: immer () dahinter
- Auch ohne Parameter: **println()** nicht **println**
- Beide Klammern nicht vergessen: (und)