HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Typische Java-Syntaxfehler

Auflistung aus Klausuraufgaben

Autor

Luis Staudt

27. November 2025

Inhaltsverzeichnis

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:

```java
// Beispiel 1: int zu boolean
boolean flag = 42;  // FEHLER: int kann nicht zu boolean werden

// Beispiel 2: String zu int
int number = "123";  // FEHLER: String ist kein int

// Beispiel 3: double zu int ohne Cast
int x = 3.14;  // FEHLER: double kann nicht ohne Cast zu int werden

// Beispiel 4: char zu boolean
boolean isTrue = 'A';  // FEHLER: char ist kein boolean

// Beispiel 5: Array zu einzelnem Wert
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:

```
// Beispiel 1: String mit int vergleichen
String text = "Hello";
int number = 5;
if (text == number) { }  // FEHLER: String und int sind inkompatibel

// Beispiel 2: boolean mit int vergleichen
boolean flag = true;
if (flag == 1) { }  // FEHLER: boolean und int sind nicht vergleichbar

// Beispiel 3: double mit String vergleichen
double pi = 3.14;
if (pi > "3") { }  // FEHLER: double kann nicht mit String verglichen
    werden

// Beispiel 4: Array mit einzelnem Wert
int[] numbers = {1, 2, 3};
if (numbers == 1) { }  // FEHLER: Array kann nicht mit int verglichen
    werden

// Beispiel 5: char mit String
char c = 'A';
String s = "A";
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:

```
// Beispiel 1: boolean addieren
boolean a = true;
int result = a + 5;  // FEHLER: boolean kann nicht addiert werden

// Beispiel 2: String subtrahieren
String text = "Hello";
String result = text - "ll";  // FEHLER: - funktioniert nicht mit Strings

// Beispiel 3: int mit && verknuepfen
int x = 5;
int y = 10;
if (x && y) { }  // FEHLER: && braucht boolean-Operanden

// Beispiel 4: String multiplizieren
String s = "abc";
String result = s * 3;  // FEHLER: * funktioniert nicht mit Strings

// Beispiel 5: boolean modulo
boolean b1 = true;
boolean b2 = false;
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:

```java
// Beispiel 1: int-Methode mit double aufrufen
public static boolean isEven(int number) {
    return number % 2 == 0;
}
// Aufruf:
double x = 4.5;
boolean result = isEven(x);  // FEHLER: double passt nicht zu int

// Beispiel 2: String-Methode mit int aufrufen
public static void printMessage(String msg) {
    System.out.println(msg);
}
// Aufruf:
printMessage(42);  // FEHLER: int ist kein String

// Beispiel 3: Mehrere Parameter in falscher Reihenfolge
public static void display(String name, int age) {
    System.out.println(name + " ist " + age);
}
// Aufruf:
display(25, "Anna");  // FEHLER: Reihenfolge ist falsch

// Beispiel 4: boolean statt int
public static int multiply(int a, int b) {
    return a * b;
}
// Aufruf:
int result = multiply(5, true);  // FEHLER: boolean statt int

// Beispiel 5: Array statt einzelner Wert
public static void printNumber(int num) {
    System.out.println(num);
}
// Aufruf:
int[] numbers = {1, 2, 3};
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:

```
// Beispiel 1: Variable nie deklariert
public static void main(String[] args) {
    System.out.println(name);  // FEHLER: name wurde nie deklariert
}

// Beispiel 2: Tippfehler im Variablennamen
int counter = 0;
counter++;
System.out.println(countr);  // FEHLER: Tippfehler (countr statt counter)

// Beispiel 3: Variable ausserhalb des Scopes
if (true) {
    int localVar = 5;
}
System.out.println(localVar);  // FEHLER: localVar existiert hier nicht
   mehr

// Beispiel 4: Falsche Gross-/Kleinschreibung
int myValue = 10;
System.out.println(MyValue);  // FEHLER: Java ist case-sensitive

// Beispiel 5: Variable in falscher Reihenfolge verwendet
System.out.println(result);  // FEHLER: result wird erst spaeter deklariert
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:

```
// Beispiel 1: Einfache nicht initialisierte Variable
public static void main(String[] args) {
    int x;
    int y = x + 5;  // FEHLER: x wurde nicht initialisiert
}

// Beispiel 2: Bedingte Initialisierung
int value;
if (Math.random() > 0.5) {
    value = 10;
}
System.out.println(value);  // FEHLER: value koennte uninitialisiert sein

// Beispiel 3: Variable in Schleife
int sum;
for (int i = 0; i < 10; i++) {
    sum += i;  // FEHLER: sum wurde nicht initialisiert
}

// Beispiel 4: String ohne Initialisierung
String message;
message = message.toUpperCase();  // FEHLER: message hat keinen Wert

// Beispiel 5: Array-Element
int[] numbers = new int[5];
int first = numbers[0];  // OK: Arrays werden mit 0 initialisiert
int standalone;
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:

```java
// Beispiel 1: Direkte Verwendung in main
public class Test {
    int instanceVar = 10;

    public static void main(String[] args) {
        System.out.println(instanceVar);  // FEHLER: nicht-static in static
    }
}

// Beispiel 2: In statischer Methode
public class Calculator {
    double result = 0;

    public static void reset() {
        result = 0;  // FEHLER: result ist nicht static
    }
}

// Beispiel 3: String-Variable
public class Program {
    String name = "Test";

    public static void printInfo() {
        System.out.println(name);  // FEHLER: name ist nicht static
    }
}

// Beispiel 4: Array-Variable
public class Data {
    int[] numbers = {1, 2, 3};

    public static void process() {
        int first = numbers[0];  // FEHLER: numbers ist nicht static
    }
}

// Beispiel 5: boolean-Variable
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:

```
// Beispiel 1: if-Block
public static void main(String[] args) {
    if (true) {
        int x = 5;
    }
    System.out.println(x);  // FEHLER: x existiert hier nicht
}

// Beispiel 2: for-Schleife
public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
        int sum = i * 2;
    }
    System.out.println(sum);  // FEHLER: sum existiert hier nicht
}

// Beispiel 3: while-Schleife
public static void main(String[] args) {
    while (true) {
        String message = "Loop";
        break;
    }
    System.out.println(message);  // FEHLER: message existiert hier nicht
}

// Beispiel 4: Verschachtelte Bloecke
public static void main(String[] args) {
    {
        int temp = 100;
    }
    int value = temp;  // FEHLER: temp existiert hier nicht
}

// Beispiel 5: Switch-Case
public static void main(String[] args) {
    int choice = 1;
    switch (choice) {
        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:

```
// Beispiel 1: In main-Methode
public class Test {
    public void printMessage() {
        System.out.println("Hello");
    }

    public static void main(String[] args) {
        printMessage();  // FEHLER: printMessage ist nicht static
    }
}

// Beispiel 2: Von statischer Hilfsmethode
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public static int calculate() {
        return add(5, 3);  // FEHLER: add ist nicht static
    }
}

// Beispiel 3: Getter-Methode
public class Person {
    private String name = "Max";

    public String getName() {
        return name;
    }

```

```java
    public static void displayName() {
        System.out.println(getName());  // FEHLER: getName ist nicht static
    }
}

// Beispiel 4: Setter-Methode
public class Counter {
    private int count = 0;

    public void increment() {
        count++;
    }

    public static void resetAndIncrement() {
        increment();  // FEHLER: increment ist nicht static
    }
}

// Beispiel 5: Void-Methode aufrufen
public class Helper {
    public void doSomething() {
        System.out.println("Doing...");
    }

    public static void execute() {
        doSomething();  // FEHLER: doSomething ist nicht static
    }
}
```

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:

```
// Beispiel 1: Zu wenige Parameter
public static int add(int a, int b) {
    return a + b;
}
// Aufruf:
int result = add(5);  // FEHLER: 2 Parameter erwartet, nur 1 uebergeben

// Beispiel 2: Zu viele Parameter
public static void printName(String name) {
    System.out.println(name);
}
// Aufruf:
printName("Max", "Mustermann");  // FEHLER: 1 Parameter erwartet, 2
    uebergeben

// Beispiel 3: Keine Parameter erwartet
public static void greet() {
    System.out.println("Hello!");
}
// Aufruf:
greet("World");  // FEHLER: 0 Parameter erwartet, 1 uebergeben

// Beispiel 4: Drei Parameter, zwei uebergeben
public static double calculate(int a, int b, int c) {
    return (a + b) * c;
}
// Aufruf:
double result = calculate(5, 10);  // FEHLER: 3 Parameter erwartet

// Beispiel 5: Ein Parameter fehlt
public static boolean isInRange(int value, int min, int max) {
    return value >= min && value <= max;
}
// Aufruf:
boolean check = isInRange(50, 0);  // FEHLER: 3 Parameter erwartet, 2
    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:

```java
// Beispiel 1: String statt int
public static void printNumber(int num) {
    System.out.println("Number: " + num);
}
// Aufruf:
printNumber("42");  // FEHLER: String statt int

// Beispiel 2: int statt boolean
public static void setFlag(boolean flag) {
    System.out.println("Flag: " + flag);
}
// Aufruf:
setFlag(1);  // FEHLER: int statt boolean

// Beispiel 3: double statt String
public static void displayMessage(String message) {
    System.out.println(message);
}
// Aufruf:
displayMessage(3.14);  // FEHLER: double statt String

// Beispiel 4: Array statt einzelner Wert
public static int square(int x) {
    return x * x;
}
// Aufruf:
int[] numbers = {2, 3, 4};
int result = square(numbers);  // FEHLER: Array statt int

// Beispiel 5: Mehrere Parameter mit falschen Typen
public static void register(String name, int age, boolean active) {
    System.out.println(name + ", " + age + ", " + active);
}
// Aufruf:
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:

```java
// Beispiel 1: Komplett fehlendes return
public static int getValue() {
    int x = 5;
    // FEHLER: kein return
}

// Beispiel 2: return nur in if, nicht in else
public static int getNumber(boolean condition) {
    if (condition) {
        return 10;
    }
    // FEHLER: was wenn condition false ist?
}

// Beispiel 3: return nach unerreichbarem Code
public static double calculate() {
    if (true) {
        return 3.14;
    }
    // Dieser Code ist unerreichbar, aber return fehlt formal
}

// Beispiel 4: return in Schleife, aber nicht ausserhalb
public static String findName() {
    for (int i = 0; i < 10; i++) {
        if (i == 5) {
            return "Found";
        }
    }
    // FEHLER: was wenn die Schleife durchlaeuft ohne return?
}

// Beispiel 5: switch ohne default
public static int getDays(int month) {
    switch (month) {
        case 1: return 31;
        case 2: return 28;
        // ...
```

```
39      }
40      // FEHLER: was bei ungueltigen 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:

```java
// Beispiel 1: if ohne else
public static int getResult(boolean flag) {
    if (flag) {
        return 1;
    }
    // FEHLER: wenn flag false ist, kein return
}

// Beispiel 2: mehrere if ohne finales return
public static String getGrade(int points) {
    if (points >= 90) {
        return "A";
    }
    if (points >= 80) {
        return "B";
    }
    // FEHLER: was bei points < 80?
}

// Beispiel 3: while-Schleife
public static int findValue() {
    int i = 0;
    while (i < 10) {
        if (i == 5) {
            return i;
        }
        i++;
    }
    // FEHLER: was wenn 5 nie erreicht wird?
}

// Beispiel 4: try-catch ohne return ausserhalb
public static int parseValue(String s) {
    try {
        return Integer.parseInt(s);
    } catch (Exception e) {
        System.out.println("Error");
    }
```

```
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:

```java
// Beispiel 1: int statt String
public static String getNumber() {
    return 42;  // FEHLER: int wird zurueckgegeben, String erwartet
}

// Beispiel 2: String statt int
public static int getValue() {
    return "123";  // FEHLER: String statt int
}

// Beispiel 3: double statt boolean
public static boolean check() {
    return 1.0;  // FEHLER: double statt boolean
}

// Beispiel 4: Array statt einzelner Wert
public static int getFirst() {
    int[] numbers = {1, 2, 3};
    return numbers;  // FEHLER: Array statt int
}

// Beispiel 5: boolean statt int
public static int calculate() {
    return true;  // FEHLER: boolean statt int
}
```

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:

```java
// Beispiel 1: String zurueckgeben
public static void printMessage() {
    return "Hello";  // FEHLER: void-Methode kann keinen Wert zurueckgeben
}

// Beispiel 2: int zurueckgeben
public static void calculate() {
    int result = 5 * 5;
    return result;  // FEHLER: void kann keinen int zurueckgeben
}

// Beispiel 3: boolean zurueckgeben
public static void checkValue(int x) {
    if (x > 0) {
        return true;  // FEHLER: void kann keinen boolean zurueckgeben
    }
}

// Beispiel 4: double zurueckgeben
public static void computeAverage(int a, int b) {
    double avg = (a + b) / 2.0;
    return avg;  // FEHLER: void kann keinen double zurueckgeben
}

// Beispiel 5: Object zurueckgeben
public static void createObject() {
    Object obj = new Object();
    return obj;  // FEHLER: void kann kein Object zurueckgeben
}
```

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:

```java
// Beispiel 1: Methode in Methode
public static void outerMethod() {
    public static void innerMethod() {  // FEHLER: nicht verschachtelt erlaubt
        System.out.println("Inner");
    }
}

// Beispiel 2: Hilfsmethode innerhalb
public static int calculate(int x) {
    public static int helper(int y) {  // FEHLER: nicht erlaubt
        return y * 2;
    }
    return helper(x);
}

// Beispiel 3: main mit innerer Methode
public static void main(String[] args) {
    public static void greet() {  // FEHLER: nicht innerhalb main
        System.out.println("Hello");
    }
    greet();
}

// Beispiel 4: Mehrfach verschachtelt
public static void methodA() {
    public static void methodB() {  // FEHLER
        public static void methodC() {  // FEHLER
            System.out.println("C");
        }
    }
}

// Beispiel 5: return-Typ-Methode verschachtelt
public static int getValue() {
    public static int compute() {  // FEHLER
        return 42;
    }
```

```
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

addcontentslinetocchapterKategorie 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:

```
// Beispiel 1: int statt boolean
int x = 5;
if (x) {  // FEHLER: x ist int, nicht boolean
    System.out.println("true");
}

// Beispiel 2: String-Bedingung
String name = "Max";
if (name) {  // FEHLER: String ist nicht boolean
    System.out.println("Name exists");
}

// Beispiel 3: Zuweisung statt Vergleich
int value = 10;
if (value = 5) {  // FEHLER: = ist Zuweisung, nicht Vergleich (sollte ==
   sein)
    System.out.println("Five");
}

// Beispiel 4: Null-Wert
Object obj = null;
if (obj) {  // FEHLER: Object ist nicht boolean
    System.out.println("Exists");
}

// Beispiel 5: Arithmetischer Ausdruck
int a = 10, b = 20;
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:

```java
// Beispiel 1: int statt boolean
int count = 5;
while (count) {  // FEHLER: count ist int, nicht boolean
    System.out.println(count);
    count--;
}

// Beispiel 2: Zahl als Bedingung
while (10) {  // FEHLER: 10 ist kein boolean
    System.out.println("Loop");
}

// Beispiel 3: String-Bedingung
String text = "running";
while (text) {  // FEHLER: String ist nicht boolean
    System.out.println(text);
}

// Beispiel 4: Zuweisung statt Vergleich
int x = 0;
while (x = 5) {  // FEHLER: = statt == (und dann ist es auch noch int)
    x++;
}

// Beispiel 5: Array als Bedingung
int[] numbers = {1, 2, 3};
while (numbers) {  // FEHLER: Array ist nicht boolean
    System.out.println("Array");
}
```

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:

```java
// Beispiel 1: Einfacher Zugriff von aussen
class MyClass {
    private int secret = 42;
}

public class Test {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.secret);  // FEHLER: secret ist private
    }
}

// Beispiel 2: Private String
class Person {
    private String password = "1234";
}

public class App {
    public static void main(String[] args) {
        Person p = new Person();
        String pw = p.password;  // FEHLER: password ist private
    }
}

// Beispiel 3: Private Methode
class Calculator {
    private int multiply(int a, int b) {
        return a * b;
    }
}
```

```java

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        int result = calc.multiply(5, 3);  // FEHLER: multiply ist private
    }
}

// Beispiel 4: Private Array
class DataHolder {
    private int[] data = {1, 2, 3};
}

public class Processor {
    public static void main(String[] args) {
        DataHolder holder = new DataHolder();
        int first = holder.data[0];  // FEHLER: data ist private
    }
}

// Beispiel 5: Aenderung von private Variable
class Counter {
    private int count = 0;
}

public class Program {
    public static void main(String[] args) {
        Counter c = new Counter();
        c.count = 10;  // FEHLER: count ist private
    }
}
```

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:

```java
// Beispiel 1: main greift auf Instanzvariable zu
public class Test {
    int number = 10;

    public static void main(String[] args) {
        System.out.println(number);  // FEHLER: number ist nicht static
    }
}

// Beispiel 2: Static Methode ruft Instanzmethode auf
public class Helper {
    public void help() {
        System.out.println("Helping...");
    }

    public static void doWork() {
        help();  // FEHLER: help ist nicht static
    }
}

// Beispiel 3: String-Variable in main
public class App {
    String message = "Hello";

    public static void main(String[] args) {
        System.out.println(message);  // FEHLER: message ist nicht static
    }
}

// Beispiel 4: Instanzvariable in statischer Hilfsmethode
public class Calculator {
    double result = 0;

    public static void reset() {
        result = 0;  // FEHLER: result ist nicht static
    }

    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:

```java
// Beispiel 1: Groesse bei Deklaration
int[5] numbers;  // FEHLER: Groesse nicht bei Deklaration

// Korrekt waere:
int[] numbers = new int[5];
// oder
int[] numbers;
numbers = new int[5];

// Beispiel 2: Klammern am falschen Ort
int numbers[5];  // FEHLER: Groesse nicht erlaubt

// Beispiel 3: Falsche Syntax mit String
String[10] names;  // FEHLER: Groesse nicht bei Deklaration

// Beispiel 4: Mehrere Dimensionen falsch
int[3][4] matrix;  // FEHLER: Groessen nicht bei Deklaration

// Beispiel 5: Boolean-Array
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:

```
// Beispiel 1: Groesse und Werte gleichzeitig
int[] numbers = new int[3] {1, 2, 3};  // FEHLER: entweder Groesse oder
   Werte

// Korrekt waere:
int[] numbers = new int[3];  // Groesse ohne Werte
// oder
int[] numbers = {1, 2, 3};  // Werte ohne Groesse/new
// oder
int[] numbers = new int[] {1, 2, 3};  // new ohne Groesse

// Beispiel 2: String-Array
String[] names = new String[2] {"Max", "Anna"};  // FEHLER

// Korrekt:
String[] names = {"Max", "Anna"};

// Beispiel 3: Double-Array
double[] values = new double[4] {1.1, 2.2, 3.3, 4.4};  // FEHLER

// Korrekt:
double[] values = {1.1, 2.2, 3.3, 4.4};

// Beispiel 4: Boolean-Array
boolean[] flags = new boolean[3] {true, false, true};  // FEHLER

// Korrekt:
boolean[] flags = {true, false, true};

// Beispiel 5: Char-Array
char[] letters = new char[5] {'a', 'b', 'c', 'd', 'e'};  // FEHLER

// Korrekt:
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:

```java
// Beispiel 1: Methode nicht geschlossen
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello");
    // FEHLER: } fehlt fuer main

    public static void other() {
        System.out.println("Other");
    }
}

// Beispiel 2: if-Block nicht geschlossen
public static void check(int x) {
    if (x > 0) {
        System.out.println("Positive");
        // FEHLER: } fehlt fuer if
}

// Beispiel 3: Verschachtelte Bloecke
public static void complex() {
    for (int i = 0; i < 10; i++) {
        if (i % 2 == 0) {
            System.out.println(i);
        // FEHLER: } fehlt fuer if
    }
}

// Beispiel 4: Klasse nicht geschlossen
public class MyClass {
    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:

```java
// Beispiel 1: Variable ohne Semikolon
public static void main(String[] args) {
    int x = 5  // FEHLER: ; fehlt
    System.out.println(x);
}

// Beispiel 2: Methodenaufruf
public static void test() {
    System.out.println("Hello")  // FEHLER: ; fehlt
    System.out.println("World");
}

// Beispiel 3: Return-Statement
public static int getValue() {
    return 42  // FEHLER: ; fehlt
}

// Beispiel 4: Zuweisung
public static void calculate() {
    int a = 10;
    int b = 20;
    int sum = a + b  // FEHLER: ; fehlt
    System.out.println(sum);
}

// Beispiel 5: Mehrere Anweisungen
public static void demo() {
    int x = 1  // FEHLER: ; fehlt
    int y = 2  // FEHLER: ; fehlt
    int z = x + y  // FEHLER: ; fehlt
}
```

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:

```java
// Beispiel 1: if ohne Klammern
public static void check(boolean flag) {
    if flag {  // FEHLER: () fehlen
        System.out.println("True");
    }
}

// Korrekt:
if (flag) {
    System.out.println("True");
}

// Beispiel 2: while ohne Klammern
public static void loop() {
    int i = 0;
    while i < 10 {  // FEHLER: () fehlen
        i++;
    }
}

// Beispiel 3: for-Schleife
public static void iterate() {
    for int i = 0; i < 5; i++ {  // FEHLER: () fehlen um gesamte for-
    Deklaration
        System.out.println(i);
    }
}

// Beispiel 4: Methodenaufruf
public static void greet() {
    System.out.println;  // FEHLER: () fehlen
}

// Beispiel 5: Verschachtelte Bedingung
public static void complex(int x, int y) {
    if x > 0 && y > 0 {  // FEHLER: () fehlen
        System.out.println("Both positive");
    }
```

```
38 }
```

Worauf achten:

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