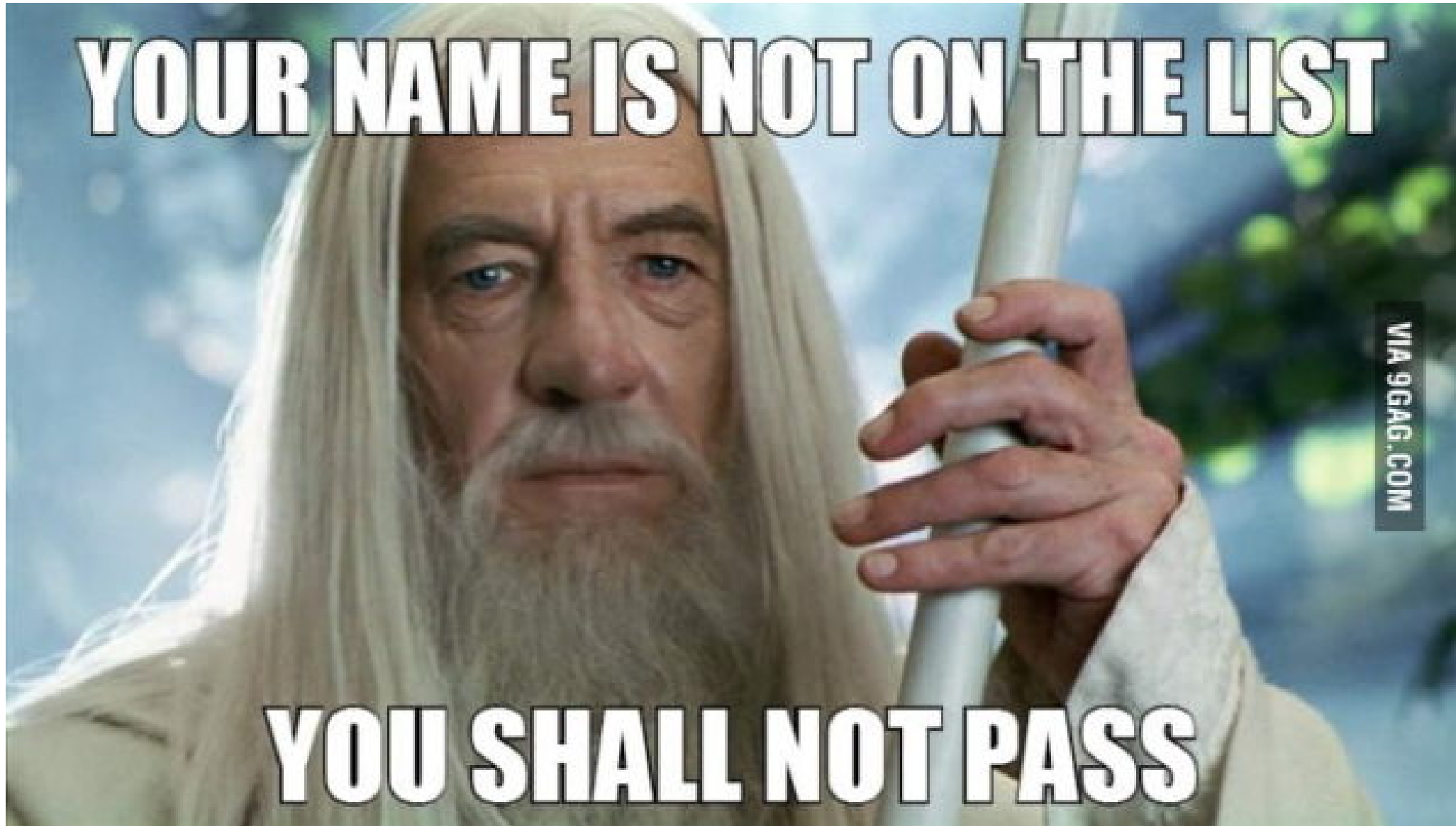


Selektion

Kontrollstrukturen

Vergleichs- und logische Operatoren kommen häufig dann zum Einsatz, wenn man etwas nur **unter einer bestimmten Bedingung ausführen** soll.





Bedingungen | Vergleichsoperatoren

Gegeben: `int a = 3; int b = 3;`

| Operator | Beschreibung | Beispiel | Resultat |
|--------------------|---|------------------------|--------------------|
| <code>==</code> | überprüft auf <u>Gleichheit</u> | <code>a == b</code> | <code>true</code> |
| <code>!=</code> | überprüft auf <u>Ungleichheit</u> | <code>a != b</code> | <code>false</code> |
| <code>></code> | ist linker Operand <u>grösser</u> | <code>a > b</code> | <code>false</code> |
| <code>>=</code> | ist linker Operand <u>grösser oder gleich</u> | <code>a >= b</code> | <code>true</code> |
| <code><</code> | ist linker Operand <u>kleiner</u> | <code>a < b</code> | <code>false</code> |
| <code><=</code> | ist linker Operand <u>kleiner oder gleich</u> | <code>a <= b</code> | <code>true</code> |

* nur bei primitiven Datentypen. Nicht bei `String` !

Bedingungen kombinieren

Gegeben: `boolean a = true; boolean b = false;`

| Operator | Beschreibung | Beispiel | Resultat |
|-------------------------|--|-----------------------------|--------------------|
| <code>&&</code> | UND: beide Ausdrücke sind <code>true</code> | <code>a && b</code> | <code>false</code> |
| <code> </code> | ODER: mindistens ein Ausdruck ist <code>true</code> | <code>a b</code> | <code>true</code> |
| <code>^</code> | XOR: genau einer der Ausdrücke ist <code>true</code> | <code>a ^ b</code> | <code>true</code> |
| <code>!</code> | NOT: wandelt ein <code>boolean</code> ins Gegenteil um | <code>!b</code> | <code>true</code> |

if / else if / else

- Wird dafür verwendet, **Bedingungen zu überprüfen**
- Als Bedingung dient ein **Bool'scher Wert** (`true` , `false`), welche über ein **Vergleichsoperator** erzeugt wird.
- Kontrolliert ob ein Codeabschnitt durchlaufen wird



if / else if / else

Beispiel

Wenn, `if`, ein Kunde einen Auftrag über 1000.-- erteilt, bekommt er 4 % Rabatt.



In Java

```
double price = StdInput.readDouble();

if (price > 1000) {
    price *= 0.96;
}

System.out.println("Your price " + price);
```

- Bedingung: `price > 1000`
 - Operator: `>` grösser als
- Anweisung: `price *= 0.96`
 - Oder: `price = price * 0.96`



if / else if / else

Mit `else if` kann priorisiert auf weitere Bedingungen reagiert werden

Schema

```
if (<Bedingung1>) {  
    <Anweisung1>  
} else if (<Bedingung2>) { // Optionaler Block  
    <Anweisung2>  
} else { // Optionaler Block  
    <Anweisung3>  
}
```

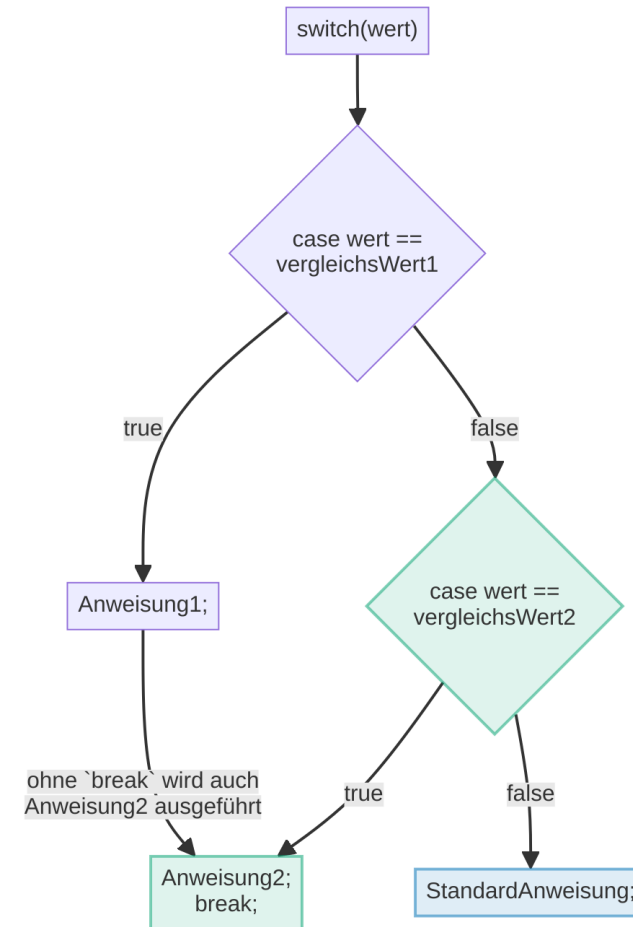
💡 if-Statements können beliebig verschachtelt werden!

Code-Beispiel

```
int age; // beliebiges alter  
double betrag; // beliebiger Betrag  
if (betrag > 10000 && age < 18) {  
    // mehr als 10000 ausgegeben  
    // UND unter 18 Jahre alt  
    betrag *= 0.9;  
} else if (betrag > 1000) {  
    betrag *= 0.96;  
} else { // Für alle andern  
    betrag *= 0.98;  
}
```

switch / case

- Wird dafür verwendet, **Gleichheit zu überprüfen**
- Als Bedingung dient die exakte Gleichheit von Werten (`==`)
- Mit `break` wird abgebrochen
- Ohne `break` wird die nächste Anweisung auch ausgeführt



switch / case

Schema

```
switch(wert) {  
    case vergleichsWert1:  
        <Anweisung1>;  
        // ohne `break` wird auch <Anweisung2>  
        // bis zum `break` ausgeführt.  
    case vergleichsWert2:  
        <Anweisung2>;  
        break;  
    default:  
        <StandardAnweisung>;  
}
```

💡 Natürlich können beliebig viele
case Blöcke folgen!

Code Beispiel

```
switch(kunde) {  
    case "Hans":  
        System.out.println("Hallo Hans!");  
    case "Fritz":  
        System.out.println("wie goots?");  
        break;  
    default:  
        System.out.println("Ciao");  
}
```

Hans: Hallo Hans! wie goots?

Fritz: wie goots?

Alle Anderen: Ciao