

Condicionais e Fluxo de Controle

outra declaração

A `else` instrução executa um bloco de código quando a condição dentro da `if` instrução é `false`. A `else` declaração é sempre a última condição.

```
boolean condition1 = false;

if (condition1){
    System.out.println("condition1 is
true");
}
else{
    System.out.println("condition1 is not
true");
}
// Prints: condition1 is not true
```

else if Declarações

`else - if` as instruções podem ser encadeadas para verificar várias condições. Quando uma condição for `true`, um bloco de código será executado e a instrução condicional será encerrada. Pode haver várias instruções `else - if` em uma única instrução condicional.

```
int testScore = 76;
char grade;

if (testScore >= 90) {
    grade = 'A';
} else if (testScore >= 80) {
    grade = 'B';
} else if (testScore >= 70) {
    grade = 'C';
} else if (testScore >= 60) {
    grade = 'D';
} else {
    grade = 'F';
}

System.out.println("Grade: " + grade); //
Prints: C
```

Se Declaração

Uma `if` instrução executa um bloco de código quando uma expressão booleana especificada é avaliada como `true`.

```
if (true) {
    System.out.println("This code
executes");
}
// Prints: This code executes

if (false) {
    System.out.println("This code does
not execute");
}
// There is no output for the above
statement
```

Declarações condicionais aninhadas

Uma instrução condicional aninhada é uma instrução condicional aninhada dentro de outra instrução condicional. A instrução condicional externa é avaliada primeiro; se a condição for `true`, a instrução condicional aninhada será avaliada.

```
boolean studied = true;
boolean wellRested = true;

if (wellRested) {
    System.out.println("Best of luck
today!");
    if (studied) {
        System.out.println("You are prepared
for your exam!");
    } else {
        System.out.println("Study before your
exam!");
    }
}

// Prints: Best of luck today!
// Prints: You are prepared for your
exam!
```

E Operador

The AND logical operator is represented by `&&`. This operator returns `true` if the `boolean` expressions on both sides of the operator are `true`; otherwise, it returns `false`.

```
System.out.println(true && true); //
Prints: true
System.out.println(true && false); //
Prints: false
System.out.println(false && true); //
Prints: false
System.out.println(false && false); //
Prints: false
```

NOT Operator

The NOT logical operator is represented by `!`. This operator negates the value of a boolean expression.

```
boolean a = true;
System.out.println(!a); // Prints: false

System.out.println(!false) // Prints:
true
```

The OR Operator

The logical OR operator is represented by `||`. This operator will return `true` if at least one of the `boolean` expressions being compared has a `true` value; otherwise, it will return `false`.

```
System.out.println(true || true); //
Prints: true
System.out.println(true || false); //
Prints: true
System.out.println(false || true); //
Prints: true
System.out.println(false || false); //
Prints: false
```

Conditional Operators – Order of Evaluation

If an expression contains multiple conditional operators, the order of evaluation is as follows:
Expressions in parentheses → NOT → AND → OR.

```
boolean foo = true && (!false || true);
// true
/*
(!false || true) is evaluated first
because it is contained within
parentheses.
```

Then `!false` is evaluated as `true` because it uses the NOT operator.

Next, `(true || true)` is evaluated as `true`.

Finally, `true && true` is evaluated as `true` meaning `foo` is `true`. */