15787 users solved this topic. Latest completion was **about 1 hour ago**.

Theory: Relational operators

21 minutes 5 / 13 problems solved

Start practicing

§1. List of relational operators

Java provides six relational operators to compare numbers:

- `==` (equal to)
- `!=` (not equal to)
- `>` (greater than)
- `>=` (greater than or equal to)
- `<` (less than)
- `<=` (less than or equal to)

The result of applying a relational operator to its operands takes the **boolean** type (`true` or `false`) regardless of the types of operands.

§2. Comparing integer numbers

Relational operators allow you to easily compare, among other things, two integer numbers. Here are some examples below:

```
int one = 1;
int two = 2;
int three = 3;
int four = 4;

boolean oneIsOne = one == one; // true

boolean res1 = two <= three; // true
boolean res2 = two != four;  // true

boolean res3 = two > four;   // false

boolean res4 = one == three; // false
```

Relational operators can be used in mixed expressions together with arithmetic operators. In such expressions, relational operators have lesser priorities than arithmetic operators.

In the following example, first of all, two sums are calculated, and then they are compared using the operator `>`.

```
int number = 1000;
boolean result = number + 10 > number + 9; // 1010 > 1009 is true
```

The `result` is `true`.

§3. Joining relational operations using logical operators

In Java, you cannot write an expression like a <= b <= c. Instead of it, you should join two boolean expressions using logical operators like `||` and `&&`.

Here is an example:

```
number > 100 && number < 200; // it means 100 < number < 200
```

Also, we can write parts of the expression in parentheses to improve readability:

```
(number > 100) && (number < 200);
```

But parentheses are not necessary here because relational operators have a higher priority than logical operators.

Current topic:

✓ Relational operators — Stage 2

Topic depends on:

✓ Integer types and operations — Stage 1

✓ Boolean and logical operations — Stage 2

Topic is required for:

✓ Conditional statement — Stage 2

Table of contents:

Here is a more general example of variables.

```
int number = ...              // it has a value
int low = 100, high = 200;    // borders


boolean inRange = number > low && number < high;  // joining two expressions using
 AND.
```

The code checks if the value of `number` belongs to a range.

So, logical operators allow you to join a sequence of relational operations into one expression.

§4. An example of a program

Suppose there are three children in the sports class. You want to check if they are arranged in the descending order. The following program reads three integer numbers `h1`, `h2`, `h3` and then checks if `h1 >= h2` and `h2 >= h3`. Note that **h** means **the height of a child**.

```
import java.util.Scanner;

public class CheckAscOrder {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int h1 = scanner.nextInt();
        int h2 = scanner.nextInt();
        int h3 = scanner.nextInt();



        boolean descOrdered = (h1 >= h2) && (h2 >= h3);



        System.out.println(descOrdered);

    }

}
```

There are several input-output pairs:

Input 1

```
185 178 172
```

Output 1

```
true
```

Input 2

```
181 184 177
```

Output 2

```
false
```

It is possible not to use an additional variable to store the boolean result before output:

```
System.out.println((h1 >= h2) && (h2 >= h3));
```

But when your condition is quite long, it is hard to understand what code does without some explanations. A variable with a good name provides such an explanation.

Report a typo

1230 users liked this theory. **13** didn't like it. What about you?

Start practicing

Comments (16) Hints (0) Useful links (0) Show discussion