Relational Operators in Java 21

A Beginner's Guide

Relational operators are one of the most fundamental building blocks in Java. They let you compare two values and ask questions like "Is this number greater than that one?" or "Are these two values equal?" The result of any relational expression is always a **boolean** — either **true** or **false**.

The Six Relational Operators

Operator	Meaning	Example	Result
==	Equal to	5 == 5	true
!=	Not equal to	5 != 3	true
>	Greater than	7 > 4	true
<	Less than	2 < 9	true
>=	Greater than or equal	5 >= 5	true
<=	Less than or equal	3 <= 6	true

A Simple Example

The program below demonstrates each operator in action. Notice that each comparison produces either **true** or **false**.

```java
public class RelationalDemo {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;

        System.out.println(a == b);   // false — 10 is not equal to 20
        System.out.println(a != b);   // true  — 10 is not equal to 20
        System.out.println(a > b);    // false — 10 is not greater than 20
        System.out.println(a < b);    // true  — 10 is less than 20
        System.out.println(a >= 10);  // true  — 10 is equal to 10
        System.out.println(b <= 20);  // true  — 20 is equal to 20
    }
}
```

Output:

```
false
true
false
true
true
true
```

Using Relational Operators with if Statements

Relational operators are most commonly used inside **if** statements to make decisions in your code.

```java
public class AgeCheck {
    public static void main(String[] args) {
        int age = 18;

        if (age >= 18) {
            System.out.println("You are eligible to vote.");
        } else {
            System.out.println("You are not old enough to vote yet.");
        }
    }
}
```

Output: You are eligible to vote.

Using Relational Operators in Loops

You can also use relational operators to control how many times a loop runs.

```
public class CountDown {
    public static void main(String[] args) {
        int count = 5;

        while (count > 0) {
            System.out.println("Count: " + count);
            count--;
        }
        System.out.println("Done!");
    }
}
```

A Common Mistake: == vs =

One of the most frequent errors beginners make is confusing the **assignment operator** = with the **equality operator** ==.

- = **assigns** a value: `int x = 5;`
- == **compares** two values: `x == 5`

```
// Wrong — will NOT compile
if (a = 5) { ... }

// Correct
if (a == 5) { ... }
```

■ *Using = inside a condition causes a **compile error** in Java — which is actually helpful, as it prevents this mistake from going unnoticed.*

Comparing Strings: Don't Use ==

When comparing **String** objects, always use the `.equals()` method rather than ==. The == operator checks whether two variables point to the **same object in memory**, not whether their contents are the same.

```
String s1 = new String("hello");
String s2 = new String("hello");

System.out.println(s1 == s2);       // false — different objects in memory
System.out.println(s1.equals(s2));  // true  — same content
```

■ *This is a classic trap for Java beginners. Always use `.equals()` for String comparisons.*

Combining with Logical Operators

You can combine multiple comparisons using **logical operators**: && (AND), `||` (OR), and `!` (NOT).

```
int score = 75;

// Both conditions must be true (AND)
if (score >= 60 && score <= 100) {
    System.out.println("You passed!");
}

// At least one must be true (OR)
if (score < 50 || score > 100) {
    System.out.println("Invalid or failing score.");
}
```

Quick Recap

- Relational operators compare two values and always return **true** or **false**.
- The six operators are: `==, !=, >, <, >=, <=`.
- They are most useful inside **if** statements, loops, and conditional expressions.
- Never use == to compare Strings — use `.equals()` instead.
- Don't confuse = (assignment) with == (comparison).

Once you're comfortable with relational operators, you'll find them everywhere in Java code. They're the foundation of logic and decision-making in almost every program you'll ever write.