Java Magazine

Java SE, Quiz

Quiz yourself: The plus + and equals-equals == operators in Java


Simon Roberts | December 22, 2021


Mikalai Zaikin | December 22, 2021

These common Java operators work differently with numeric and string operands.

Given the class

```
public class StringComparison {
  public static void main(String[] args) {
    var str1 = "Java 11";
    var str2 = "Java 11";
    var res = "The" + " " + "same:" + str1 == str2;
    System.out.print(res);
  }
}
```

Copy code snippet

What is the output? Choose one.

A. `The same:true` The answer is A.

B. `The same:false` The answer is B.

C. `true` The answer is C.

D. `false` The answer is D.

Answer. This question investigates operators and their precedence with particular attention to the plus (+) and equals-equals (==) operators.

- The plus operator performs numeric addition if both operands have a numeric type. Otherwise, it performs string concatenation and requires that *at least one* of the operands have the `String` type (the other argument is converted to `String` if it is not already so). If neither of these situations applies, using the + operator raises an error during compilation.
- The equals-equals operator tests *primitive values* for equivalence, and it tests *object references* for identity. That is, if it's used with two object references, == tells whether those references both refer to the *same* object.

The code in the question evaluates an expression that has three uses of the plus operator and one use of the equals-equals operator. Additive operations have higher precedence than equality operations (see the documentation for operators). This means that all three + operations will be performed, from left to right, before the == operator is evaluated.

If you take the initial expression

```
"The" + " " + "same:" + str1 == str2
```

You can add parentheses to clarify the order of execution. The result would look like this.

```
((("The" + " ") + "same:") + str1) == str2
```

Because the first two concatenations involve literal text, it's easy to see that the expression quickly reduces to this.

```
(("The same:") + str1) == str2
```

Next, the concatenation with `str1` is performed resulting in

```
("The same:Java 11") == str2
```

Finally, the identity of the computed `String` on the left of the == is tested against the identity of the

Finally, the identity of the computed `String` on the left of the `==` is tested against the identity of the `String` referred to by `str2`. Of course, these strings must be different objects in memory since they have different contents, so the result is false. Consequently `false` is printed to the console, making option D correct and the other options incorrect.

It's worth considering some variations on this expression. First, consider that in Java, if multiple `String` *literals* in a program have the same text, they will all be reduced to a single object in memory. This optimization is performed by the compiler but is also performed by the class loader, so even if the literals are in different source files that are compiled separately, this will still be true.

Thus, if you simply performed the comparison `str1 == str2` in isolation, the result would be true. However, if you use parentheses to force the == test to happen before the string concatenation, like this

```
var res = "The" + " " + "same:" + (str1 == str2);
```

The output would be

```
The same:true
```

As a final note, observe that the left side of the == comparison is a `String` object with the following value:

```
"The same:Java 11"
```

Now imagine that the value of `str2` were initialized to that same text. In that case, the output would still be `false`, because the two `String` objects, despite having the same textual contents, are *different objects in memory*. Thus, they would fail the identity check that == performs.

Conclusion. The correct answer is option D.

Related quizzes

- Quiz yourself: The scope of variables and dividing by zero
- Quiz yourself: Use primitives such as the % operator
- Quiz yourself: String manipulation
- Quiz yourself: Initializing standard and final variables in Java

Simon Roberts

Simon Roberts

Simon Roberts joined Sun Microsystems in time to teach Sun's first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance educator who publishes recorded and live video training through Pearson InformIT (available direct and through the O'Reilly Safari Books Online service). He remains involved with Oracle's Java certification projects.

Mikalai Zaikin

Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous *Sun Certified Programmer for Java* study guides by Kathy Sierra and Bert Bates.

‹ Previous Post