Quiz yourself: Local variables and unary operators

Test your knowledge of Java—and your debugging skills too.

by Simon Roberts and Mikalai Zaikin

December 14, 2020

If you have worked on our quiz questions in the past, you know none of them is easy. They model the difficult questions from certification examinations. We write questions for the certification exams, and we intend that the same rules apply: Take words at their face value and trust that the questions are not intended to deceive you but to straightforwardly test your knowledge of the ins and outs of the language.

Given the following method:

```java
public void unaryOps() {
    int x,a = 20;  // line n1
    a = a-- - --a + a++;
    x = --a + --a + ++x; // line n2
    System.out.print(a + " " + x);
}
```

What is the result? Choose one.

A. Compilation fails due to line n1. The answer is A.
B. Compilation fails due to line n2. The answer is B.
C. 18 38 The answer is C.
D. 18 58 The answer is D.

Answer. The objective of this Java SE 11 question looks complex at first glance; it's set up so you might think the focus is the behavior of the unary increment and decrement operators,
but in fact there's a much simpler key to it. Everything hinges on the declaration, initialization, and usage rules for local variables.

By the time you've taken this code apart, you could be forgiven for thinking it's a silly evaluation of test-taking skills, which is an objection often leveled at exams of all kinds. A better way to think about it is as an evaluation of debugging skill. There is a strong human tendency to focus on the first idea, ignoring other possibilities. That will distract you in a question like this, and the same tendency will make you miss things during debugging.

Enough justification: Let's look at the question.

The construction of line n1 is valid; it's equivalent to this more verbose—and probably stylistically preferred—construction:

```java
int x;
int a = 20;
```

This tells you that option A is incorrect.

As you look at that expanded form of line n1, notice that the `int` variable `x` is not initialized. That's always a possible issue in any language. In Java, local variables cannot be read from until they are definitely initialized. In other languages, strange and unrepeatable behavior is likely.

Next, notice that line n2 attempts to increment `x`. This is on the right side of the assignment to `x`. The right side of any assignment expression must be evaluated before the assignment can occur, which means that at the point the code attempts to increment `x`, no initial value has been assigned. Since incrementing is really a read-increment-assign sequence, this must fail. From this you can determine that the `++x` subexpression in line n2 will cause compilation to fail and that option B is correct.

It's perhaps useful to consider what would happen if `x` were initialized. Two possible interpretations of the code on line 1 might seem reasonable even though they're incorrect. One is that `x` might have picked up a default value of zero, and the other is that it might have been assigned the same value as `a`, which is 20.

If `x` were initialized to zero, the correct option would be C. The first line performing computations, which is this:

```java
a = a-- - --a + a++;
```

would evaluate as shown in Figure 1:
And the assignment expression results in this:

\[
a = 20 - 18 + 18
\]

Thus \(a\) contains the value 20 after this line.

The second line of computations (line n2), which is this:

\[
x = --a + --a + ++x;
\]

would evaluate as shown in Figure 2:

\[
x = --a + --a + ++x;
\]

Thus \(x\) contains the value 38 when the statement is complete.

In case \(x\) was initialized to 20, the only difference from the explanation above would be in the value of \(++x\) in the second line. That subexpression would have the value 21, and the final result would be 19 + 18 + 21, which produces the value of 58.

By the way, although the style is not desirable, Java does allow you to declare and initialize multiple variables of the same type in a single statement, for example:

```java
int x, a = x = 20; // both a and x are assigned 20
```

With the initialization above, then as described previously, the correct option would have been option D.

However, since \(x\) was not initialized, option C and option D cannot be correct.

**Conclusion:** The correct answer is option B.
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.