JAVA SE

Quiz yourself: The scope of variables and dividing by zero

If a mathematical equation's arguments aren't accessible, what happens to the math operation?

by Mikalai Zaikin and Simon Roberts

May 17, 2021

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 two methods, which are declared in the same class

```
public static float divide(float arg1, float
    throws ArithmeticException { // line n1
        return arg1/arg2;
}

public static void main(String[] args) {
    try {
        int arg1 = 10;
        int arg2 = 0;
        System.out.printf("Result: %f", divid
    } catch (RuntimeException e) {
        System.out.printf(
            "Bad arguments: %d and %d", arg1,
    }
}
```

What is the result? Choose one.

A.

`Result: Infinity` — The answer is A.

B. `Result: Nan` — The answer is B.

C. `Bad arguments: 10 and 0` — The answer is C.

D. Compilation fails at line n1. — The answer is D.

E. Compilation fails at line n2. — The answer is E.

Answer. This is an uncomfortable question because compilation fails. In real-world daily coding, the problem would be reported immediately by your development environment. This might make it seem like an unreasonable question to ask.

Let's be clear: Exams are *not* daily coding. The objective is to probe your understanding. As such, a little care and attention to detail should lead you to the right answer and, in the process, allow you to demonstrate an element of core knowledge that is being legitimately tested. If you're still uneasy by the end of the discussion, know that the exam creators try hard to limit the number of questions that fall into this category, and the information in the question—specifically "fails at line n2"—should be used to help you spot the right answer.

Let's look at the question. The setup has all the hallmarks of being about the way Java handles division by zero—but it's not. It's about the scope of variables.

In general, a local variable, such as `arg1` and `arg2` in this sample, is visible from the point of declaration to the end of the immediately enclosing block; that's the region bounded by curly braces. As a result, `arg1` and `arg2` are not accessible in the `catch` block, and line n2 fails to compile. This—along with the assurance that there's only one correct answer—tells you that option E is correct.

An important note is that the description of visibility just given isn't complete and, therefore, isn't fully correct. Formal parameters (such as the variables in the argument list of a method) will be visible from the point of declaration to the end of the block that is associated with whatever those variables are formal parameters to. By way of examples, the variable `args`, which is the formal parameter of the `main` method, is visible throughout the `main` method body. The variable `e`, which is the formal parameter of the `catch` block, is visible throughout that `catch` block. Similar rules apply to similar situations, including variables declared in the resources section of a try-with-resources structure and those declared in `for` loops.

You could fix the compilation error simply by moving the declarations of the two variables further up in the source code so they are directly above the `try` keyword. In that case, the code would compile and run.

Now, to make this question and its discussion more interesting, consider what would happen if that were the case. After all, the

distractors (the wrong answers) were chosen to be at least tempting, which should be true of all multiple-choice exam questions.

Perhaps the best starting point is to consider what happens if you perform a division by zero. It turns out that the result depends on the type of the expression. If an integer division expression has zero in the divisor (the bottom part of a fraction), the code throws an `ArithmeticException`.

In practice, this means that both the divisor and the dividend (the top) must be of integral types: If either has a floating-point type, the whole expression has a floating-point type, and no exceptions are possible. Here are the possible results.

Floating-point expressions with division by zero produce one of three special values: `Infinity`, `-Infinity`, and `NaN` ("Not a Number"), as follows:

- If the dividend is nonzero and it has the same sign as the zero divisor (floating-point arithmetic distinguishes positive and negative zero), you get `Infinity`.
- If the signs are different, you get `-Infinity`.
- If both the dividend and divisor are zero, you get NaN.

Because the code could not possibly produce `NaN`, option B must be incorrect.

The next consideration is that the variables `arg1` and `arg2` are declared as `int`, but the `divide` method takes two `float` arguments. Would this division be handled in floating-point or integer arithmetic format? The former. The integer arguments are promoted to `float` for the method invocation, so a floating-point expression is evaluated and, again, no exception will be thrown. This tells you that option C cannot be correct, even if the scope problem were fixed.

From the previous discussion, you can tell that if the variable scope issue were fixed and the code were compiled, the output would be in the form of option A. However, in the code's current form, option A is incorrect.

You might then ask the following: If the expression cannot throw an exception, is it an error that the method declares an exception that will definitely never arise? The answer is no, and in fact, it's a general rule that methods are permitted to declare exceptions that they never throw.

One reason this is important is that overriding methods are not permitted to throw checked exceptions that are not permissible from the method being overridden. On this basis, abstract methods in interfaces regularly declare exceptions that, given that they have no implementation, they obviously cannot throw.

It is worth noting that `ArithmeticException` is an unchecked exception, so there's never a *requirement* to declare it on any

method. However, it is perfectly *permissible* to do so, even if it's unusual and not recommended style. From this, you can determine that line n1 does not cause a compilation error and option D is incorrect.

Conclusion. The correct answer is option E.

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.

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.

Share this Page