Quiz yourself: Use primitives such as the % operator

Test your knowledge of Java’s primitive operators.

by Simon Roberts and Mikalai Zaikin

December 24, 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.

The objective of this Java SE 11 quiz is to use primitives—in this case, the % operator.

Imagine that your department is developing an application that accesses an external web service by host name. High demand has led to a requirement for load balancing across a pool of servers, and your colleague has implemented this by rotation. At login, each application user is assigned to one server from the pool. This is done based on a randomly generated single sign-on (SSO) token generated by the following code:

```java
public static final List<String> HOSTS = List
private Optional<String> getRotatedHost(BigInteger ssoToken.has
if (HOSTS.size() > 1) {
    return Optional.of(HOSTS.get(ssoToken.has
} return Optional.empty();
```

Which of the following statements is correct? Choose one.
A. The code compiles and properly rotates servers, distributing the load approximately evenly between two of them.

The answer is A.

B. The code compiles, but always returns `abc.xyz`.

The answer is B.

C. The code compiles, but always returns `def.xyz`.

The answer is C.

D. The code compiles, but sometimes throws an exception.

The answer is D.

**Answer.** The quiz code is built around several concepts: accessing list elements by index, using the `hashCode()` method on an object, and using Java’s remainder operator (`%`). Let’s look at what it does.

In its current form, the size of the `HOSTS` list is hardcoded as two items, and the list is created using the `List.of` factory. Such a list is unmodifiable, meaning that the list structure itself cannot be changed. This is not the same as an immutable list, however, since if the list contained mutable objects such as `StringBuilder` instances, the content of those items could still be changed.

In this case, it’s fairly evident that the expectation is that the number of items might be increased over time as more servers are brought online. However, for this quiz question, that’s irrelevant—you know the size of the list will never be less than two, and the only interesting code is inside the `if` construct.

Looking at that line of code, you can see several things:

First, the question uses `Optional.of` rather than `Optional.ofNullable`. The `Optional.of` method throws an exception if its argument is `null`, so you must evaluate whether that situation might ever occur.

Second, notice that the argument to the `Optional.of` call is an invocation of `HOSTS.get()`. The `get` call on a `List` can sometimes return a `null` if a null-tolerant list is in use and that list happens to contain a `null` at the specified index. However, in this code, two elements are hardcoded into the list, neither of which is `null`. Further, the list is unmodifiable, so there’s no chance of a `null` here, and you can rule out the concern raised above.

Third, the `get` method of a list will always return a value for any index that’s in range. The valid range will be zero to one less than the number of elements in the list. In this code, the index is
calculated from the `hashCode` value and that `%` remainder operator. The `hashCode()` method returns an `int` primitive, and the value can be anywhere in the plus-or-minus-two-billion range. This list requires an index of either `0` or `1`, so clearly, it’s important that you do not use the `hashCode` values directly, as the vast majority of them would be outside the acceptable range for this list.

The final observation is that the code uses the `%` operator, in what might seem like a fairly common pattern, to try to bring the `hashCode` values down into the acceptable range for the `get` method. However, closer inspection shows that there is a problem here.

It’s common to hear the `%` operator referred to as the modulo operator. Indeed, in some other languages (for example Python), `%` is exactly that. However, in Java, `%` signifies the remainder. Mathematically, the remainder and modulo operations are not the same—and the difference is critical to this question.

If you have two operands that are both positive, there is no difference between the remainder and modulo operations. However, if any operands are negative, the two are not the same.

Importantly for this quiz, recall that the value of a `hashCode` can be anywhere in the valid range of an `int` and, therefore, negative values will occur sometimes.

Notice also that the question states that the SSO value is randomly generated, which suggests that on average it's equally likely to be positive or negative.

So, what’s the difference between the two operations?

- With a remainder operation, the sign of the result will always be the same as the sign of the first operand.
- With a modulo operation, the sign will always be the same as the second operand.

In the code in this question, if there is a negative value from the `hashCode`, you will get a negative value from the remainder operation, and the code will attempt to access list elements using an illegal index value. The result, of course, is that an `IndexOutOfBoundsException` exception will be thrown.

Given these observations, it’s clear that option D is the correct answer, and the others must therefore be incorrect.

**More about remainder versus modulo**

One way to think about the difference between the remainder and modulo operators is this. The modulo operator is sometimes referred to as “clock arithmetic” in that it reduces a value in the same way that a clock reduces a hundred hours into a number of days, which are lost, and the remaining count of hours.
However, instead of only reducing to hours within a day, the
modulo operation creates a result that is between zero and the
second operand; hence, the sign must be the same as that of
the second operand.

A remainder, however, calculates what’s left after a sharing
operation. If you share -7 items between 3 people, you can
account for -6 items by giving each person -2. However, you
have -1 items left over. Hence, the remainder takes its sign from
the first operand.

The really important point here, however, is that not all
programming languages perform the same mathematics for the
% operator. It’s critical to know how a specific project’s language
will behave.

The Java code below demonstrates the error. The BigInteger
is generated randomly, so to witness the problem, you should
run this code multiple times:

```java
public static void main(String[] args) {
    NetClient client = new NetClient();
    BigInteger number = new BigInteger(1024, new SecureRandom());
    System.out.println(number);
    var v = client.getRotatedHost(number);
    System.out.println(v.get());
}
```

Three overloaded Math.floorMod(x, y) methods were
introduced in Java 8, and they work as modulo operators. In this
quiz, the second operator is the size of the list. Because the size
is always positive, you can be sure that using this operation
instead of the % operator will give a positive result. So, you can
fix the code as follows:

```java
return Optional.of(HOSTS.get(Math.floorMod(ss
```

Finally, there’s an interesting (and more complete) discussion of
how programming languages implement the modulo and
remainder operations on Wikipedia.

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

---

**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.