Java Magazine

Java SE, Quiz

Quiz yourself: Initializing standard and final variables in Java


Mikalai Zaikin


Simon Roberts

November 1, 2021

Generally, every *field* in an object or class is initialized to a zero-like value during the allocation of memory—but not always.

Given the `SuperChar` class

```
public class SuperChar {
  static final char zero; // line n2
  private transient final char one = '1'; // line n3

  public SuperChar() {
    System.out.print("'" + zero + one + "'");
```

```
  }

  public static void main(String... args) {
    new SuperChar();
  }
}
```

Copy code snippet

If the numeric value of the `char '1'` is `49`, what is the result? Choose one.

A. `' 1'` — The answer is A.

B. `'01'` — The answer is B.

C. `'1'` — The answer is C.

D. `' 49'` — The answer is D.

E. `'49'` — The answer is E.

F. Compilation fails due to line n2. — The answer is F.

G. Compilation fails due to line n3. — The answer is G.

Answer. Let's take care of the lowest-hanging fruit first. The set of modifiers on line n3 is entirely valid and the line does not cause a compilation error. Thus, option G is incorrect.

Default initialization of variables is a popular topic for the Java certification exam. Generally, every *field* in an object or class is initialized to a zero-like value during the allocation of memory. This aspect is, in fact, inseparable from the action of `new` and ensures that numeric primitive types start out with zero values, object types start out as `null`, and `Boolean` types start out `false`. In most situations, if no further explicit initialization is provided by the code, a field will simply remain that zero-like value.

However, there is an exception to this behavior for `final` fields, which are required to be *explicitly* initialized. If this is not done, the code will fail to compile.

In many cases, `final` variables are initialized immediately in the declaration statement; this approach is convenient and obviously correct. However, if this is not done, a `final` field must be initialized later. If such delayed initialization is used, the field is referred to as a *blank final*. This is discussed in *Java*

Language Specification section 8.3.1.2, which makes two particular points.

> A blank final class variable must be definitely assigned by a static initializer of the class in which it is declared, or a compile-time error occurs.
>
> A blank final instance variable must be definitely assigned and moreover not definitely unassigned at the end of every constructor of the class in which it is declared, or a compile-time error occurs.

The first of these points describes blank `final` class variables. In the quiz code, line n2 declares just such a `final` class variable.

```
static final char zero;
```

Copy code snippet

Clearly, this declaration does not initialize the `zero` variable in the declaration, but further, there is nothing in any `static` initializer block that does so either. Because of this, the `zero` field is never explicitly assigned a value, and the class will not compile. This means option F is correct and the remaining options incorrect.

Let's consider some variations. First, imagine that the `final` modifier was not included. In that case, normal default initialization would apply, the code would compile successfully, and the value of `zero` would, in fact, be numerically `0`.

If that change were made

- The string concatenation that precedes printing the output would first concatenate the opening single quote mark with the value of `zero`, which is the `char` with numeric value `0`.
- The second concatenation would append the character `1` and then the closing single quote mark.

The representation of the `zero` character is platform-dependent. Sometimes it is represented as a space, sometimes it produces no output, and other possibilities exist. Of the options listed, two possibilities are shown: in option A, `' 1'` (which has a space before the 1), and in option C, (which does not have a space). The `char`-type variable `zero` will not be presented as a digit `0`, nor will the value be promoted to an `int` since it is explicitly a `char` type. Therefore, option B would still be incorrect.

Side note: When a value explicitly typed as a `char` is concatenated with a string, the *character* is concatenated—the `char` is *not* promoted to `int`. By contrast, if two `char` values are added together, they will both be promoted to `int` as part of the normal numeric operand promotion. This means that you will not see `49` in the output with the code as it is. However, what if the print line were changed to look like this?

```
```

```
System.out.print("'" + (zero + one) + "'");
```

Copy code snippet

Then the operands `zero` and `one` would both be promoted to `int` type, their values would be summed to 49, that sum would be converted from an `int` back into a textual representation of that number, and the output would be `'49'`. However, since that's not the case here, options D and E are incorrect.

Conclusion. The correct answer is option F.

Related quizzes

- Quiz yourself: String manipulation
- Quiz yourself: The Optional class and null values in Java
- Quiz yourself: Final classes

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

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.

‹ Previous Post