Quiz Yourself: Variable Declaration
(Intermediate)

The scope rules of Java variables and an examination of shadowing

by Simon Roberts and Mikalai Zaikin

November 21, 2019

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. The "intermediate" and "advanced" designations refer to the exams, rather than to the questions, although in almost all cases, "advanced" questions will be harder. 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 code snippets:

A.```java
class C1 {
    void foo(int a) {
        for (int a = 0; a < 5; a++) {
        }
    }
}
```

B.```java
class C2 {
    int a = 0;
    { int a = 1; }
}
```

C.```java
class C3 {
    { int a = 0; }
    { int a = 1; }
}
```

D.```java
class C4 {
    { int a = 0;
        for (int a = 0; a < 5; a++) {
        }
    }
}
```

E.```java
class C5 {
    { for (int a = 0; a < 5; a++) {
        int a = 0;
    }
}
```
Which snippets successfully compile? Choose three.

**Answer.** This question investigates the scope rules of Java along with an effect referred to as *shadowing*, where two identifiers of the same name exist in the same scope, but referring to the simple, unqualified name reaches one and ignores the other. In general, a local-scoped variable may be defined in such a way that it shadows another variable in a class or instance scope. However, this may not be done to local-scoped variables. Notice this choice is a useful one, because you can always refer explicitly to the class or instance field using the class name or the implicit variable `this`. Consider this example:

```java
public class MyClass {
    static int x = 99;
    int y = 100;
    public static void showX() {
        int x = 9;
        System.out.println("x is "+x); // prints 9
        System.out.println("MyClass.x is "+MyClass.x); // prints 99
    }
    public void showY() {
        int y = 10;
        System.out.println("y is "+y); // prints 10
        System.out.println("this.y is "+this.y); // prints 10
    }
}
```

Let’s take a look at the options.

In option A, you see a method parameter called `a`. Method parameters are local variables and have a scope that starts from the argument list and continues to the closing curly brace that ends the method declaration. But there is also another local variable named `a` that is declared in the `for` loop. Because of this, and the rule that prohibits having two local variables with the same name and overlapping scope, this code will not compile, and option A is incorrect.

Option B defines an instance variable called `a` and also an instance initializer block that declares a local variable of the same name. However, because the variable defined in the instance initializer is a local variable (just as it would be in a method), that local variable shadows the instance-scoped variable successfully. Therefore, the code compiles correctly and option B is correct.

Option C declares two independent instance initializer blocks and each has a local variable named `a`, but their scopes, limited by their enclosing initializer blocks, do not overlap in any way so the variables do not conflict. This code is useless because variables inside instance initializer blocks are not visible anywhere else in the class and are immediately lost after the initializer completes. You could be forgiven for expecting the compiler to object in the same way that it does with unreachable code, but it does not; the syntax is valid, and option C is correct.

In option D the code declares a local variable and immediately declares another with the same name in the loop. A variable declared in a `for` loop is a local variable having a scope that starts at the point of declaration and ends with the end of the loop. Of course, this means that two local variables of the same name appear to be in scope through the body of the loop, and this is prohibited. This situation is closely parallel to the code in option A, with the difference that in option A, the first-declared local variable was a method parameter, and in this option, the first-declared local variable is a simple local variable. However, you can see from this that option D is also incorrect.

In option E, the code is somewhat similar to option D, but the loop and local variable declarations are in the opposite order. As a result, the loop variable is out of scope before the second declaration. Therefore, the two variables do not conflict, the code is valid, and option E is correct.
The topic of the scope of declarations is detailed in the *Java Language Specification*, particularly section 6.3, “Scope of a Declaration.”

From that section, two quotations are particularly relevant. The following one pertains to the scope of method parameters:

The scope of a formal parameter of a method (§8.4.1), constructor (§8.8.1), or lambda expression (§15.27) is the entire body of the method, constructor, or lambda expression.

And this one pertains to variables declared in `for` loops:

The scope of a local variable declared in the `ForInit` part of a basic `for` statement (§14.14.1) includes all of the following:

- Its own initializer
- Any further declarators to the right in the `ForInit` part of the `for` statement
- The `Expression` and `ForUpdate` parts of the `for` statement
- The contained `Statement`

The correct answer is options B, C, and E.

---

**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.

---

**Share this Page**

Facebook  
Twitter  
LinkedIn  
Email