Quiz yourself: Accessibility of private methods in Java

What happens when some classes define private methods—and other classes attempt to access them?

by Mikalai Zaikin and Simon Roberts

June 28, 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 code

```java
class C1 {
    private int m1() {
        return 0;
    }
}

class S1 {
    public int doIt() {
        return new C1().m1() + 1;
    }
}

class S2 extends C1 {
    public int doIt() {
        return m1() + 1;
    }
}

class S3 {
    class C3 {
        private int m3() {
            return 0;
        }
    }
    public int doIt() {
        return new C3().m3() + 1;
    }
}
```
Which classes will compile? Choose two.

A. Class S1
   The answer is A.

B. Class S2
   The answer is B.

C. Class S3
   The answer is C.

D. Class S4
   The answer is D.

Answer. This question explores the accessibility of private members. Each of the classes C1, C3, and C4 defines a private method, and each of the candidates—S1, S2, S3, and S4—attempts to access them.

In option A, the two classes have no relationship to one another, other than perhaps being members of the same package (the code shows nothing on this point, so no assumptions can be made). In this situation, private members of such classes are inaccessible to the other class, so S1 will be denied access to m1(), the code will fail to compile, and option A is incorrect.

In option B, the class S2 is a subclass of C1, which does create a relationship that can grant privileged access to certain members. So, S2 now attempts to access the m1() method using the implicit this prefix. However, this will be denied, because the inheritance relationship is not sufficient to grant access to private members of either parent classes or subclasses. Because of this, the code fails to compile and option B is also incorrect.

At this point, you have rejected two options, and only two remain. Since you are required to select two options, it might seem that options C and D must be correct. In an exam, if you are crushingly short of time, that might be a reasonable approach if you are confident of those rejections. Let’s take a proper look at the two remaining options, however.

Option C has an outer S3 class and an inner class C3 that contains the private method you are trying to access from the outer class.

Option D has two classes, C4 and C4A, that are defined inside the outer class S4. In this example, the private member is inside C4, and the access is being made from C4A.

Both options C and D demonstrate the accessibility of private members where nested classes are concerned.
In view of this rule, it's clear that the accesses proposed by options C and D will be valid. The only remaining concern is whether you have the right prefixes to allow access to object instances or static members. In this case, both options C and D do, in fact, provide valid prefixes.

In option C, an instance method of class S₃ creates an instance of the inner (instance) class C₃, and then it uses that instance as the prefix to access the m₃ method.

In option D, the method m₄ is a static method in a static class, and the invocation is prefixed with the class name. This again is correct.

From these observations you can conclude that options C and D are both correct.

By the way, option A would compile successfully if method m₁() had default (package-level) access. Further, option B would compile successfully if method m₁() had protected access even if the classes C₁ and S₂ were defined in different packages.

More details on accessing members from other classes can be found in the specification for Java SE 11 under "Determining accessibility," which says the following about private members:

A member (class, interface, field, or method) of a reference type, or a constructor of a class type, is accessible only if the type is accessible and the member or constructor is declared to permit access:

If the member or constructor is declared public, then access is permitted. All members of interfaces lacking access modifiers are implicitly public.

Otherwise, the member or constructor is declared private, and access is permitted if and only if it occurs within the body of the top level type that encloses the declaration of the member or constructor.

Notice that this phrasing does not mention any of the nested classes; it simply says access is permitted from inside the body of the top-level type. Since everything in the nested classes is (by definition) enclosed by the top-level type, access is permitted.

Conclusion. The correct answers are options C and D.
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