# 1.https://stackoverflow.com/questions/10141626/changing-private-methods-to-protected-for-testing/10141754#10141754

**T:**Changing private methods to protected for testing

**Q:**Is it a good idea to change the private methods to protected for JUNIT testing.?

**C1:**Why should it? You change something before you test, I can't see ANY reason for it?

**C2:**Almost certainly not a good idea; if you are directly testing the internal methods of a class, you should consider refactoring them into a separate class.

**C3:**I've done this in cases where a private method returns some resource which is not available in the unit testing environment, and I want to test some other (non-private) method that calls it. By making it protected I can override it, so that it returns some stub/mock instead of the unavailable resource. This has meant a unit test then becomes feasbile and I've not been able to find another way round this, but it's less than ideal.

**C4:**Yes, it's less than ideal, but go for it if other solutions would be too cumbersome. Ideally you would have a way to mock the resource without having to change your class. Making testable code does influence design choices like this one.

**C5:**default scope is better than protected.

4 **Answer**

**A1:**It's sometimes useful, yes.  
  
If the class is extendable, make sure to make the method final.  
  
Also, document the fact that the method is not supposed to be called by subclasses or external classes of the same package.  
  
I use the Guava @VisibleForTesting annotation to make it clear that the method should in fact be private.

**C1:**Thanks for opening my eyes to @VisibleForTesting. I didn't know this existed. It still feel this is a little imperfect in that it impacts the syntax of the implementation just for the purpose of testing, but it is better than exposing members and methods that should be private. And limiting testing to interface methods seems an overly strict and sometimes unpractical restriction.

**A2:**No in general not. The idea of unit testing is to test ... units. Or in other words implementations of interface methods. If you want to test a method which you can't "see" this could be a code smell. Maybe you haven't separated your business logic enough from the UI code or something.  
  
So the best idea would be to rethink your architecture. But if the alternative would be to not test your code it is a good idea to make those methods protected.

**A3:**You can make the methods package local instead.  
  
You can call private method using reflection, or you can decide that private methods shouldn't be tested directly, only indirectly.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**A4:**Although you should prefer to refactor as @user714965 suggested, PowerMock's MockPrivate can do the mocking without opening the visibility of your private methods.   
  
Writing your tests first usually leads to a design where you don't need to mock private methods.