Prefer Testing Public APIs Over Implementation-Detail Classes

Does this class need to have tests?

```
class UserInfoValidator {
  public void validate(UserInfo info) {
    if (info.getDateOfBirth().isInFuture()) { throw new ValidationException(); }
  }
}
```

Its method has some logic, so it may be good idea to test it. But **what if its only user looks like this?**

```
public class UserInfoService {
  private UserInfoValidator validator;
  public void save(UserInfo info) {
    validator.validate(info); // Throw an exception if the value is invalid.
    writeToDatabase(info);
  }
}
```

The answer is: **it probably doesn't need tests,** since all paths can be tested through `UserInfoService`. The key distinction is that **the class is an implementation detail, not a public API**.

A public API can be called by any number of users, who can pass in any possible combination of inputs to its methods. **You want to make sure these are well-tested**, which ensures users won't see issues when they use the API. Examples of public APIs include classes that are used in a different part of a codebase (e.g., a server-side class that's used by the client-side) and common utility classes that are used throughout a codebase.

An implementation-detail class exists only to support public APIs and is called by a very limited number of users (often only one). **These classes can sometimes be tested indirectly** by testing the public APIs that use them**.**

Testing implementation-detail classes is still useful in many cases, such as if the class is complex or if the tests would be difficult to write for the public API class. When you do test them, **they often don't need to be tested in as much depth as a public API,** since some inputs may never be passed into their methods (in the above code sample, if `UserInfoService` ensured that `UserInfo` were never `null`, then it wouldn't be useful to test what happens when `null` is passed as an argument to `UserInfoValidator.validate`, since it would never happen).

Implementation-detail classes can sometimes be thought of as private methods that happen to be in a separate class, since you typically don't want to test private methods directly either. **You should also try to restrict the visibility of implementation-detail classes**, such as by making them package-private in Java.

Testing implementation-detail classes too often leads to a couple problems:

- **Code is harder to maintain since you need to update tests more often**, such as when changing a method signature of an implementation-detail class or even when doing a refactoring. If testing is done only through public APIs, these changes wouldn't affect the tests at all.

- **If you test a behavior only through an implementation-detail class, you may get false confidence in your code,** since the same code path may not work properly when exercised through the public API. **You also have to be more careful when refactoring,** since it can be harder to ensure that all the behavior of the public API will be preserved if not all paths are tested through the public API.

More information, discussion, and archives:

http://googletesting.blogspot.com