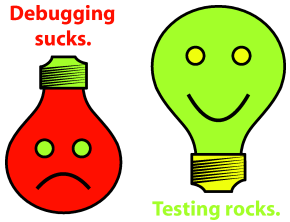

Testing on the Toilet

Know Your Test Doubles

Google

A test double is an object that can stand in for a real object in a test, similar to how a stunt double stands in for an actor in a movie. These are sometimes all commonly referred to as "mocks", but it's important to distinguish between the different types of test doubles since they all have different uses. **The most common types of test doubles are stubs, mocks, and fakes**.

A **stub** has no logic, and only returns what you tell it to return. **Stubs can be used when you need an object to return specific values in order to get your code under test into a certain state**. While it's usually easy to write stubs by hand, using a mocking framework is often a convenient way to reduce boilerplate.

```
// Pass in a stub that was created by a mocking framework.
AccessManager accessManager = new AccessManager(stubAuthenticationService);

// The user shouldn't have access when the authentication service returns false.
when(stubAuthenticationService.isAuthenticated(USER_ID)).thenReturn(false);
assertFalse(accessManager.userHasAccess(USER_ID));

// The user should have access when the authentication service returns true.
when(stubAuthenticationService.isAuthenticated(USER_ID)).thenReturn(true);
assertTrue(accessManager.userHasAccess(USER_ID));
```

A **mock** has expectations about the way it should be called, and a test should fail if it's not called that way. **Mocks are used to test interactions between objects**, and are useful in cases where there are no other visible state changes or return results that you can verify (e.g. if your code reads from disk and you want to ensure that it doesn't do more than one disk read, you can use a mock to verify that the method that does the read is only called once).

```
// Pass in a mock that was created by a mocking framework.
AccessManager accessManager = new AccessManager(mockAuthenticationService);

accessManager.userHasAccess(USER_ID);

// The test should fail if accessManager.userHasAccess(USER_ID) didn't call
// mockAuthenticationService.isAuthenticated(USER_ID) or if it called it more than once.
verify(mockAuthenticationService).isAuthenticated(USER_ID);
```

A **fake** doesn't use a mocking framework: it's a lightweight implementation of an API that behaves like the real implementation, but isn't suitable for production (e.g. an in-memory database). **Fakes can be used when you can't use a real implementation in your test** (e.g. if the real implementation is too slow or it talks over the network). You shouldn't need to write your own fakes often since fakes should usually be created and maintained by the person or team that owns the real implementation.

```
// Creating the fake is fast and easy.
AuthenticationService fakeAuthenticationService = new FakeAuthenticationService();

AccessManager accessManager = new AccessManager(fakeAuthenticationService);

// The user shouldn't have access since the authentication service doesn't
// know about the user.
assertFalse(accessManager.userHasAccess(USER_ID));

// The user should have access after it's added to the authentication service.
fakeAuthenticationService.addAuthenticatedUser(USER_ID);
assertTrue(accessManager.userHasAccess(USER_ID));
```

More information, discussion, and archives:
http://googletesting.blogspot.com