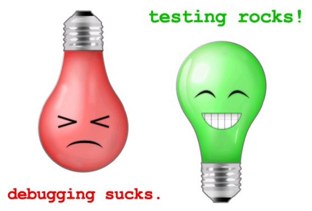

Testing on the Toilet

Dec 4, 2008

Use EasyMock for, well, easier mocks

Most of us are aware that mock and stub objects can make testing easier by isolating the class under test from external dependencies. This goes hand-in-hand with dependency injection. Writing all these classes can be a pain though.

EasyMock provides an alternative. It dynamically implements an interface which records and replays your desired behavior. Let's say you want to model an `ATM` interface:

```
public interface Atm {
  boolean enterAccount(String accountNumber);
  boolean enterPin(String pin);
  boolean enterWithdrawalAmount(int dollars);
}
```

It is pretty easy to mock this interface. Still, every mock has to implement all three methods, even if you only need one. You also need a separate mock for each set of inputs. With EasyMock, you can create mocks as you need them, recording and replaying your expectations:

```
public void testAtmLogin() {
  Atm mockAtm = createMock(Atm.class);                                    // 1
  EasyMock.expect(mockAtm.enterAccount("MyAccount")).andReturn(true);   // 2
  EasyMock.expect(mockAtm.enterPin("1234")).andReturn(true);              // 3
  EasyMock.replay(mockAtm);                                               // 4
  Account account = new Account();
  account.login(mockAtm);                                                 // 5
  assertTrue(account.isLoggedIn();
  EasyMock.verify(mockAtm);                                               // 6
}
```

We tell EasyMock to create a dynamic proxy implementing `Atm` (**1**), which starts in record mode. Then we record two method calls along with the expected results (**2** and **3**). The `replay()` call tells EasyMock to stop recording (**4**). After that, calls on the object return the set values. If it gets a call it does not expect, it throws an Exception to fail fast. `Account` now uses the mock as if it were the real thing (**5**). The `verify()` method checks to see if the mock actually received all the calls you expect (**6**). It really is that simple. If we want to simulate failure, we can set up another test to return `false` from one of the method calls.

EasyMock has lots more capabilities as well. It can throw exceptions. It also can record multiple calls to the same method returning the same or different results. You also can create *stub* expectations and *nice* mocks so you don't have to record every expected call. You also can create several mocks, and even nest them to test classes with complex dependencies. Beware, though, this often creates brittle tests, and is a sign the class under test needs refactoring.

Basic EasyMock only mocks interfaces, but there is an EasyMockClassExtension that mocks non-final classes when you really must. See the EasyMock documentation at the link below for details.

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