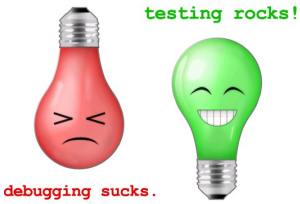

Testing on the Toilet
Partial Mocks using Forwarding Objects

Feb 19, 2009

A ***Partial Mock*** is a mock that uses **some behavior from a real object** and **some from a mock object.** It is useful when you need bits of both. One way to implement this is often a ***Forwarding Object*** (or *wrapper*) which **forwards calls to a delegate**.

For example, when writing an Olympic swimming event for ducks, you could create a simple forwarding object to be used by multiple tests:

```
interface Duck {
  Point getLocation();
  void  quack();
  void  swimTo(Point p);
}
```

```
class ForwardingDuck implements Duck {
  private final Duck d;
  ForwardingDuck(Duck delegate) { this.d = delegate; }
  public Point getLocation()  { return d.getLocation(); }
  public void quack()         { d.quack(); }
  public void swimTo(Point p) { d.swimTo(p); }
}
```

And then create a test that uses all of the real ***OlympicDuck*** class's behavior except quacking.

```
public void testDuckCrossesPoolAndQuacks() {
  final Duck mock = EasyMock.createStrictMock(Duck.class); // enforces call order
  mock.swimTo(FAR_SIDE);
  mock.quack();            // quack after the race
  EasyMock.replay(mock);

  Duck duck = OlympicDuck.createInstance();
  Duck partialDuck = new ForwardingDuck(duck) {
    @Override public void quack()         { mock.quack(); }
    @Override public void swimTo(Point p) { mock.swimTo(p); super.swimTo(p); }
    // no need to @Override "Point getLocation()"
  }

  OlympicSwimmingEvent.createEventForDucks()
                      .withDistance(ONE_LENGTH)
                      .sponsoredBy(QUACKERS_CRACKERS) // their slogan: "quack"
                      .addParticipant(partialDuck)
                      .doRace();

  MatcherAssert.assertThat(duck.getLocation(), is(FAR_SIDE));
  EasyMock.verify(mock);}
```

`partialDuck` is a complex example of a partial mock – it combines real and mock objects in three different ways:

- `quack()` **calls the mock object**. It verifies that the duck doesn't promote the sponsor (by quacking) until after the race. (We skip the real quack() method so that our continuous build doesn't drive us crazy.)
- `getLocation()` **calls the real object**. It allows us to use the `OlympicDuck`'s location logic instead of rewriting/simulating the logic from that implementation.
- `swimTo(point)` **calls both objects**. It allows us to verify the call to the real duck before executing it.

There is some debate about whether you should forward to the real or mock Duck by default. If you use the mock duck by default, any new calls to the mock will break the test, making them brittle. If you use the real duck, some very sensitive calls like `submitToDrugTest()` might get called by your test if your duck happens to win.

Consider using a ***Partial Mock*** **in tests** when you need **to leverage** the implementation of **the real object**, but want to **limit, simulate or verify method calls using** the power of **a mock object**.

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