1.
We've initially set up the mock to receive any signals of method calls to the IDatabase interface. The using(mocks.Record()) block specifies which particular methods to listen to, their arguments, and tells it that it returns something. Then, we set up the Hotel object like we have before and simply run the test.

2.
```
LastCall.Throw(new Exception("whatever"));
```

3.
A stub is a special case of the DynamicMock (or was it the other way around...).

If there is nothing to be returned, then one should use the DynamicMock and call mock.VerifyAll() or a similar method.

4.
Similar to the previous example, we've set up the mock to act as our IDatabase. However, we aren't directly testing a method on the database. Instead, we're asserting the state of a property. So we set up what our expectation should be. We then let the mock know of that expectation. We arrange and run as we did before.

5.
The problem with testing methods involving the ServiceLocator is that the fields of that class are private without setters. However, we bypass that obstacle with a reflective approach. We set up the field the way we want it to be, and then use reflection voodoo magic to set the field from outside the class. Then we follow with the typical arrange, act, assert.