Mocking in Tests

TDD and mocking are like best friends

How do we write unit tests?

```
public class OrderStateTester extends TestCase {
  private static String TALISKER = "Talisker";
  private static String HIGHLAND_PARK = "Highland Park";
  private Warehouse warehouse = new WarehouseImpl();

  protected void setUp() throws Exception {
    warehouse.add(TALISKER, 50);
    warehouse.add(HIGHLAND_PARK, 25);
  }
  public void testOrderIsFilledIfEnoughInWarehouse() {
    Order order = new Order(TALISKER, 50);
    order.fill(warehouse);
    assertTrue(order.isFilled());
    assertEquals(0, warehouse.getInventory(TALISKER));
  }
  public void testOrderDoesNotRemoveIfNotEnough() {
    Order order = new Order(TALISKER, 51);
    order.fill(warehouse);
    assertFalse(order.isFilled());
    assertEquals(50, warehouse.getInventory(TALISKER));
  }
```

```
public class OrderInteractionTester extends MockObjectTestCase {
  private static String TALISKER = "Talisker";

  public void testFillingRemovesInventoryIfInStock() {
    //setup - data
    Order order = new Order(TALISKER, 50);
    Mock warehouseMock = new Mock(Warehouse.class);

    //setup - expectations
    warehouseMock.expects(once()).method("hasInventory")
      .with(eq(TALISKER),eq(50))
      .will(returnValue(true));
    warehouseMock.expects(once()).method("remove")
      .with(eq(TALISKER), eq(50))
      .after("hasInventory");

    //exercise
    order.fill((Warehouse) warehouseMock.proxy());

    //verify
    warehouseMock.verify();
    assertTrue(order.isFilled());
  }

  public void testFillingDoesNotRemoveIfNotEnoughInStock() {
    Order order = new Order(TALISKER, 51);
    Mock warehouse = mock(Warehouse.class);

    warehouse.expects(once()).method("hasInventory")
      .withAnyArguments()
      .will(returnValue(false));

    order.fill((Warehouse) warehouse.proxy());

    assertFalse(order.isFilled());
  }
```

```
public interface MailService {
  public void send (Message msg);
}
public class MailServiceStub implements MailService {
  private List<Message> messages = new ArrayList<Message>();
  public void send (Message msg) {
    messages.add(msg);
  }
  public int numberSent() {
    return messages.size();
  }
}
```

class OrderStateTester...

```
  public void testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    MailServiceStub mailer = new MailServiceStub();
    order.setMailer(mailer);
    order.fill(warehouse);
    assertEquals(1, mailer.numberSent());
  }
```

class OrderInteractionTester...

```
  public void testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    Mock warehouse = mock(Warehouse.class);
    Mock mailer = mock(MailService.class);
    order.setMailer((MailService) mailer.proxy());

    mailer.expects(once()).method("send");
    warehouse.expects(once()).method("hasInventory")
      .withAnyArguments()
      .will(returnValue(false));

    order.fill((Warehouse) warehouse.proxy());
  }
}
```

What is a mock ?

Classicist or Mockist way of testing

How your testing style impacts design

Example: Trust on Outside World Interface

```
import boto3

class AarogyaSetu(object):
    def __init__(self, name, value):
        self.name = name
        self.adhaar_number = adhaar_number

    def save(self, bucket_name):
        s3 = boto3.client('s3', region_name='us-east-1')
        s3.put_object(Bucket='mybucket', Key=self.name,
Body=self.adhaar_number)
```

```python
import boto3
from moto import mock_s3
from mymodule import MyModel


@mock_s3
def test_aarogya_setu_save():
    conn = boto3.resource('s3', region_name='us-east-1')
    # We need to create the bucket since this is all in Moto's
'virtual' AWS account
    conn.create_bucket(Bucket='mybucket')

    model_instance = AarogyaSetu('fsociety', 'xxxx-xxxxx-xxxxx-xxxx')
    model_instance.save()

    body = conn.Object('mybucket', 'fsociety').get()
['Body'].read().decode("utf-8")

    assert body == 'xxxx-xxxxx-xxxxx-xxxx'
```

Example: Coupling

```
// users.js
import axios from 'axios';

class Users {
  static all() {
    return axios.get('/users.json').then(resp => {
      return resp.data.filter((user) => user.isActive)
     });
  }
}

export default Users;
```

```
// users.test.js
import axios from 'axios';
import Users from './users';

jest.mock('axios');

test('should fetch active users', () => {
  const users = [{name: 'Bob', isActive: true}, {name: 'Alice',
isActive: false}];
  const resp = {data: users};
  axios.get.mockResolvedValue(resp);

  // or you could use the following depending on your use case:
  // axios.get.mockImplementation(() => Promise.resolve(resp))

  return Users.all().then(data => expect(data).toEqual([{name: 'Bob',
isActive: true }]));
});
```

```
   // users.js
import axios from 'axios';

class Users {
  static all() {
    return axios.get('/users.json').then(resp => {
      return resp.data
     });
  }
  static filterActive(users) {
    return users.filter((user) => user.isActive)
  }
}
```

```
   // users.test.js
import axios from 'axios';
import Users from './users';

test('should filter active users', () => {
  // Given
  const users = [{name: 'Bob', isActive: true}, {name: 'Alice',
isActive: false}];
  // When
  const activeUsers = Users.filterActive(users)
  // Then
  expect(activeUsers).toEqual([{name: 'Bob', isActive: true}])
```

Using composition

Reducing coupling

Example: Test an express app

```
const express = require('express');
const app = express();


app.get('/', function (req, res) {
  res.send('Hello World!')
});


app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
});
```

```
const express = require('express');


const hello = require('./hello.js');
const handleListen = require('./handleListen');
const log = require('./log');


const port = 3000;
const app = express();


app.get('/', hello);


app.listen(port, handleListen(port, log));
```

Mocks ain’t evil.

Principles to use when mocking

Fight your urge to mock in unit tests

Use test versions of external systems in integration tests.

Push side effects to the boundaries of your system

Dont take unit test coverage as the holy grail.

Reduce coupling