GROUPON

15 tips to improve your unit tests

Droidcon 2016 Barcamp

What makes a clean test?

Three things.

Readability,

readability,

and readability

Uncle Bob, Clean Code

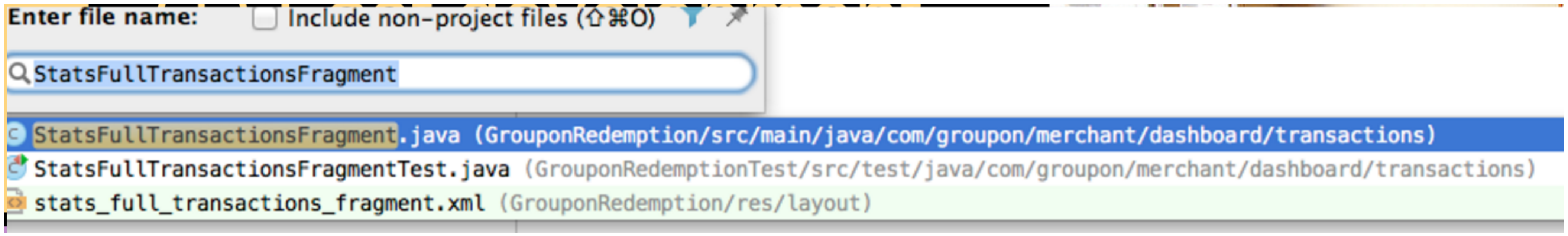
#1 Don‘t start your test methods with „***test***“

- That‘s Junit3 way
- Junit4 was released 10 years ago!
- Prevents good method names

```
@Test
public void should_stop_sync_if_error()
```

#2 Have consistent naming

- Have one name for the instance under test
  Don‘t let the reader guess which class is tested



Enter file name: Include non-project files (⇧⌘O)

StatsFullTransactionsFragment

StatsFullTransactionsFragment.java (GrouponRedemption/src/main/java/com/groupon/merchant/dashboard/transactions)
StatsFullTransactionsFragmentTest.java (GrouponRedemptionTest/src/test/java/com/groupon/merchant/dashboard/transactions)
stats_full_transactions_fragment.xml (GrouponRedemption/res/layout)

```
  @Test
  public void should_write_to_parcel() {
      tested.writeToParcel(parcel, 0);
...
```

#3 Don‘t use pure junit asserts!

- Nobody remembers actual and expected order

- Hamcrest matchers:

```
assertThat("test", anyOf(is("testing"), containsString("est")));
```

- FEST assertions:

```
assertThat(yoda).isInstanceOf(Jedi.class)
                .isEqualTo(foundJedi)
                .isNotEqualTo(foundSith);
```

#4 The test method is the most import part

- **Move as much code** out of test as possible to **setup or init-block**
- 90% of readers won‘t need that

```
@Test
public void should_not_be_empty() {
    when(login.isLoggedIn()).thenReturn(true);
    when(options.showDashboardValues()).thenReturn(true);
    assertFalse(tested.isEmptyDashboardShown());
}
```

#4 The test method is the most import part

- Move as much code out of test as possible to setup or init-block
- 90% of readers won‘t need that

```
@Before
public void setup() {
    login = mock(Login.class);
    when(login.isLoggedIn()).thenReturn(true);
    options = mock(DisplayOptions.class);
    displayUtil = mock(DisplayUtil.class);
    when(options.showDashboardValues()).thenReturn(true);
    tested = new ContentFragmentBuilderPhone(login, options, displayUtil);
}
```

#4 The test method is the most import part

- Move as much code out of test as possible to setup or init-block
- **Don‘t do both:**

```
Place place = new Place("1", "one", Location.NONE);
Merchant merchant1 = new Merchant("1", "one", MerchantAccessLevel.FULL_ACCESS);
Merchant merchant2 = new Merchant("2", "two", MerchantAccessLevel.FULL_ACCESS);
PlaceListTabletFragment tested = spy(new PlaceListTabletFragment());

@Before
public void setup() {
    tested.places = new Places(Arrays.asList(place));
    tested.merchant = merchant1;
    tested.interactor = mock(LoginInteractor.class);
    tested.loggedInUser =
       new LoggedInUser(NO_USER,NO_TOKEN, Arrays.asList(merchant1, merchant2), NO_PLACE);
}
```

#5 No javadoc to explain the test steps

```
@Test
public void testGetParentScope_shouldReturnRootScope_whenAskedForSingleton() {

    //GIVEN
    Scope parentScope = Toothpick.openScope("root");
    Scope childScope = Toothpick.openScopes("root", "child");

    //WHEN
    Scope scope = childScope.getParentScope(Singleton.class);

    //THEN
    assertThat(scope, is(parentScope));
  }
```

#1

#4

#5 No javadoc to explain the test steps

```
Scope parentScope = Toothpick.openScope("root");
Scope childScope = Toothpick.openScopes("root", "child");

@Test
public void should_return_root_scope () {
     Scope scope = childScope.getParentScope(Singleton.class);
     assertThat(scope, is(parentScope));
}
```

#6 Reduce the noise: readability first

- No finals, nor private fields
- No final on methods or variables
- Test instances don‘t see each other
  it is **safe to violate encapsulation** in tests!

```
~~private final ~~Place place = new Place("1", "one“);
~~private final ~~Merchant merchant = new Merchant("1", "one“);
```

#7 Always implement equals!

Keeps test short and readable

```
assertThat(tested.getUser())
 .isEqualTo(new User(“danny”));
```

#8 Add matcher for your models

- `anyDeal()` instead of `any(Deal.class)`
- `mockDeal()` instead of `mock(Deal.class)`

- *For android check out github.com/dpreussler/mockitoid*

#9 Use the Null-Object pattern

- Prefer Optionals over `null`
- Prefer Null-Object over optionals
- Prefer Null-Object over mocks as mocks dont respect nonnull contract
- Often in tests you just need the right type, not a specific new instance
  -> reduce test garbage

```
tested.redeem(Deal.NO_DEAL, mock(EventBus.class));
```

#10 Have default implemenations

For tests and equal checks

```
@Singleton
public class UiFlavors {

    public static final UiFlavors TABLET = new UiFlavors(true);
    public static final UiFlavors PHONE = new UiFlavors(false);

    private final boolean isTablet;

    @Inject
    public UiFlavors(Resources resources) {
        this(detectTablet(resources));
    }
...
```

Filled on runtime

#10 Have default implemenations

```
@Test
public void should_add_custom_dimension_for_phone() {
   TrackingImpl tested = new TrackingImpl(UiFlavors.PHONE, mockContext());
   tested.setCustomDimensions(eventBuilder);
   verify(eventBuilder).setCustomDimension(1, "phone");
}
```

```
 ...
```

#11 Write readable Reflection code

```
SuperReflect.on(
 Build.VERSION.class)
 .set("SDK_INT", 14);
```

```
SuperReflect.on(tested)
   .get(“privateField“);
```

Fluent interface, good for setting values (incl. finals) in API/ 3rd party/ code

github.com/dpreussler/SuperReflect

#11 Write readable Reflection code

```
@BoundBox( boundClass = A.class )
private BoundBoxOfA tested;

@Test
public void test() {
  tested = new BoundBoxOfA( new A("bb") );
  tested.boundbox_privateField()
}
```

Generate getters for private fields (compiletime-safe)

github.com/stephanenicolas/boundbox

#12 Don‘t be too strict on „units“

Real world API JSON parsing and conversion to models
is a large unit but very useful test

```
@Test
public void can_read_from_json() throws IOException {
    String values = inputStreamToString(JsonDealsTest.class.getResourceAsStream("/deals.json"));
    JsonDeals out = new Gson().fromJson(values, JsonDeals.class);

    List<Deal> deals= out.asDealList();
    assertThat (deals.get(0).getTitle()).isEqualTo("First TestTitle");
}
```

#13 If you can not test, then it‘s not good code

Do you know the rule about encapsulation and tests?

Uh, no. What rule is that?

Tests trump Encapsulation.

What does that mean?

That means that tests win.
No test can be denied access to a variable
simply to maintain encapsulation.

Uncle Bob (https://blog.8thlight.com/uncle-bob/2015/06/30/the-little-singleton.html)

#14 Inversion of Control

If you create objects (especially views) in a constructor, overload another constructor for testing

```
public OptionsViewHolder(View itemView) {
    this(itemView, new DealAnalyticsView(itemView, formatter));
}

@VisibleForTesting
OptionsViewHolder(View itemView, DealAnalyticsView analyticsView){
...
```

#15 Make it fail!

The pupil went to the master programmer and said:

"All my tests pass all the time. Don't I deserve a raise?"

The master slapped the pupil and replied:

"If all your tests pass, all the time, you need to write better tests."

Good tests fail

The Way of Testivus, Alberto Savoia

http://www.agitar.com/downloads/TheWayOfTestivus.pdf

GROUPON

Thank you!

Unit testing is no rocket science!

@PreusslerBerlin