Production level Test Driven Development

- Dagger 2
- Firebase Test Labs
- MVVM + DataBinding
- Robolectric
- Espresso
About Me

KAPIL BAKSHI

LOGISTICS + FINTECH + EDTECH

Software Engineer

akapil167
kapilbakshi167@gmail.com
About Me

International Conferences

- DroidCon Beijing 2017
- HeisenBug Russia 2018
- Android Makers France 2018
- DroidKaigi Tokyo 2018
- DroidCon Berlin 2018

kapilbakshi167@gmail.com

akapil167
The Major **Release** is Coming

And The **Bugs** Come With It
The War is Between The Features And The Bugs
And Make No Mistake

The Bugs Are Coming
And That is Why We Have Gathered Here

To Find A Solution
This talk will clear all your confusions

Which **Framework** to choose?
This talk will clear all your confusions

How to write Testable Code
This talk will clear all your confusions

How to Test on **Multiple Devices**
This talk will clear all your confusions

Unit Testing, Integration Testing or End To End Tests
Flakiness and Its Mitigation

Ferocious Flaky

Same Code

Passing

Failing
Flakiness and Its Mitigation

Ferocious Flaky

Same Code

Passing

Failing

Concurrency

Flaky third party code

Infrastructure problems
Flakiness and Its Mitigation

**Ferocious Flaky**

- Mocking Dependencies
- Relying More on Unit Tests

**Hero Hermetic to the Rescue**
Different Types Of Testing

- **Unit**: Least Flaky
- **Integration**: More Flaky
- **E2E**: Most Flaky
Different Types Of Testing

- **Fast**
- **Slower**
- **Slowest**

Diagram:
- Unit
- Integration
- E2E
This talk will clear all your confusions on Unit Testing, Integration Testing or End To End Tests.
Genuine Production
Level Scenarios
Testing Error Handling

Not Connected to Internet
Please connect and try again

Servers are Temporarily Down
Please try again after sometime
Testing Error Handling

Would you actually Turn Off the internet on your device ????

Would you actually stop your server ????

Not Connected to Internet
Please connect and try again

Servers are Temporarily Down
Please try again after sometime
This would simply **Defeat** the purpose of **Automation** and make testing **Cumbersome**
An App Accepting Different Types Of Orders

Your Orders

Sony Play Station 4 $600
Placed on 20th Oct 2017 8:00 PM
Processing

Roadster Mens’ Shirt $45
Dispatched on 17th Aug 2017 4:30 PM
Dispatched

Google Pixel(Silver, 32 GB) $735
Delivered on 1st May 2017 8:00 PM
Delivered

Your Orders

iPhone 7 (Black, 32 GB) $735
Cancelled on 29th April 2016 11:00 AM
Cancelled

Sony BRAVIA X9300E $1120
Cancelled on 29th April 2016 11:00 AM
Refund Initiated
What would happen if you
Don’t test this Hermetically
Handle Complex Order Lifecycle

Order Placed  Packed  Dispatched  Delivered

App  Packing team Dashboard  Dispatchment team Dashboard  Delivery App
You would have to

Order Placed
Packed
Dispatched
Delivered

External
From
Your
TEST CODE

HIT

APIs

App
Packing team Dashboard
Dispatchment team Dashboard
Delivery App
Then you’ll realize
It’s taking much longer to “Make Arrangements” to write Test Cases than to actually Write Test Cases.
It’s taking much longer to Write Tests to write than to actually Develop Features
You are testing what you haven’t even coded
The goal to test the code you have written gets farther far far farther.
Then solution is quite simple

Let the Code Take Control Of Everything
Let’s Explore How
Repository Pattern

Domain Layer

Business Logic

Persist

Query

Repository

Order

getOrders()

fetchOrderDetails()

updateOrderDetails()

Data Layer

Data Source

Cloud

Local Storage

Mock Data Source
Repository Pattern - Advantages

Provides Abstraction of Data
Repository Pattern - Advantages

Makes the code Highly Maintainable and Extensible
Repository Pattern - Advantages

Repository

Order
getOrders()
fetchOrderDetails()
updateOrderDetails()

Data Source

Cloud
Local Storage
Mock Data Source

Business Logic

Persist
Query

Makes the code Highly Configurable and Testable
Repository Pattern - In Action
OrdersDataSource

```java
public interface OrdersDataSource {

    Observable<Order> getOrdersResponse();

}
```

Interface which would be implemented by All the Data Sources and the Repository
OrdersDataSource

public interface OrdersDataSource {

    Observable<Order> getOrdersResponse();

}

Method which would be overridden by all data sources
Repository Pattern - In Action

Let's see the Repository Code
public class OrdersRepository implements OrdersDataSource {

private final OrdersDataSource ordersDataSource;

private OrdersRepository (@NonNull OrdersDataSource ordersDataSource) {
    this.ordersDataSource = ordersDataSource;
}

@override
public Observable<AllOrdersResponse> getOrdersResponse() {
    return ordersDataSource.getOrdersResponse();
}
OrdersRepository

public class OrdersRepository implements OrdersDataSource {

private final OrdersDataSource ordersDataSource;

private OrdersRepository (@NonNull OrdersDataSource ordersDataSource) {
    this.ordersDataSource = ordersDataSource;
}

@Override
public Observable<AllOrdersResponse> getOrdersResponse() {
    return ordersDataSource.getOrdersResponse();
}
public class OrdersRepository implements OrdersDataSource {

    private final OrdersDataSource ordersDataSource;

    private OrdersRepository(
            @NonNull OrdersDataSource ordersDataSource) {
        this.ordersDataSource = ordersDataSource;
    }

    @Override
    public Observable<AllOrdersResponse> getOrdersResponse() {
        return ordersDataSource.getOrdersResponse();
    }
}
Repository Pattern: Advantages

Let's see OrderRemoteDataSource Code
public class OrdersRemoteDataSource implements OrdersDataSource {

    @Inject
    NetworkApis networkApis;

    @Override
    public Observable<AllOrdersResponse> getOrdersResponse() {
        networkApis.getOrders();
    }
}

Implementing OrdersDataSource
public class OrdersRemoteDataSource implements OrdersDataSource {

    @Inject
    NetworkApis networkApis;

    @Override
    public Observable<AllOrdersResponse> getOrdersResponse() {
        networkApis.getOrders();
    }
}
Repository Pattern - Advantages

Let’s see FakeDataSource Code
public class FakeOrderDataSource implements OrdersDataSource {

    public static Observable<AllOrdersResponse> ALL_ORDER_RESPONSE_OBSERVABLE;

    @Override
    public Observable<Order> getOrdersResponse() {
        ALL_ORDER_RESPONSE_OBSERVABLE;
    }

    We’ll be modifying ALL_ORDER_RESPONSE_OBSERVABLE as per different scenarios
public class FakeOrderDataSource implements OrdersDataSource

    public static void createAll_Order_Response() {

        String errorMessage = null;
        boolean success = true;
        List<Order> orderList = new ArrayList<Order>();
        ALL_ORDER_RESPONSE = new AllOrdersResponse (success,
        errorMessage, orderList);

    }

Creating a default response
public class FakeOrderDataSource implements OrdersDataSource {

public void createAllOrderResponseWithServerErrorObservable(String errorMessage) {
    reCreateAll_Order_Response();

    ALL_ORDER_RESPONSE.setErrorMessage(errorMessage);
    ALL_ORDER_RESPONSE.setSuccess(false);

    ALL_ORDER_RESPONSE_OBSERVABLE = Observable.just(ALL_ORDER_RESPONSE);
}

Creates All Orders Observable with an Error to mock Server Error
public class FakeOrderDataSource implements OrdersDataSource {

public void create_Exception_Error_Observable(String exceptionMessage) {

    ALL_ORDER_RESPONSE_OBSERVABLE = Observable.<AllOrdersResponse>
error(new NullPointerException(exceptionMessage));
}

Creates All Orders Observable with an Exception
public class FakeOrderDataSource implements OrdersDataSource

    . . .

public void createOrdersObservable(String... statuses) {
    reCreateAll_Order_Response();

    for (String status : statuses) {

        Order order = createOrderBasedOnStatus(status, new Random().nextInt(Integer.MAX_VALUE));
        addOrders(order);
    }

    ALL_ORDER_RESPONSE_OBSERVABLE = Observable.just(getAllOrderResponse());
}

Creating order response
As per different order statuses

    ALL_ORDER_RESPONSE_OBSERVABLE = Observable.just(getAllOrderResponse());
}
Repository Pattern

Now we know how to use different Data Sources.
Now How do we interchange these Data Sources while Running our Tests ??
Now How do we interchange these Data Sources while Running our Tests ??

Dependency Injection is the way to go!!
Dependency Injection

The client delegates the responsibility of providing its dependencies to external code (The Injector) Without The client having to build it.
Dependency Injection:

Advantages

The client becomes highly **Configurable** and **Reusable**.

The Code becomes **Decoupled**.
Dependency Injection Using Dagger 2 - In Action
Dagger 2: Constituents

- Module
- Component
- Activity
- Fragment
- Any Other Class

Source of providing objects to be injected
Dagger 2: Constituents

Module

Generates code which helps in injection

Component

Activity
Fragment
Any Other Class

Source of providing objects to be injected
Dagger 2: Constituents

Module

Component

Generates code which helps in injection

Activity
Fragment
Any Other Class

Source of providing objects to be injected

This is where injection takes place
@Module

public class OrdersModule {

@Provides
@Singleton

public OrdersRepository providesNotesRepository() {
    return OrdersRepository.getInstance(OrdersRemoteDataSource.getInstance());
}

}
@Module
public class OrdersModule {

@Provides
@Singleton
public OrdersRepository providesNotesRepository() {
    return OrdersRepository.getInstance(OrdersRemoteDataSource.getInstance());
}

}
@Module
public class OrdersModule {

@Provides
@Singleton
public OrdersRepository providesNotesRepository() {
    return OrdersRepository.getInstance(
        OrdersRemoteDataSource.getInstance());
}

Notice Remote Order Data Source is being used here
@Module
public class OrdersTestModule extends OrdersModule {

    @Provides
    @Singleton
    public OrdersRepository providesNotesRepository() {
        return OrdersRepository.getInstance(FakeOrdersDataSource.getInstance());
    }
}

Notice Fake Order Data Source is being used here
Dagger 2: Constituents

- **Module**: Source of providing objects to be injected
- **Component**
- **Activity**
  - Fragment
  - Any Other Class
Components In Dagger 2

@Singleton
@Component (modules = {
    NotesModule.class, NetworkModule.class, OrdersModule.class
})
public interface AppComponent {

    void inject(AddEditNoteActivity addEditNoteActivity);

    void inject(AllNotesActivity allNotesActivity);

    void inject(OrdersRemoteDataSource ordersRemoteDataSource);

    void inject(AllOrdersActivity allOrdersActivity);
}

This interface is used by Dagger 2 to generate code which uses the modules to fulfill the requested dependencies.
**Dagger 2: Constituents**

- **Module**: Source of providing objects to be injected.
- **Component**: Generates code which helps in injection.
- **Activity, Fragment, Any Other Class**: Constituents generated by Dagger 2.
How does Injection Take Place

```java
public class MyApplication extends Application {

    private static AppComponent component;

    public static AppComponent getComponent() {
        return component;
    }

    public AppComponent createComponent() {
        return DaggerAppComponent.builder()
            .networkModule(new NetworkModule(this))
            .ordersModule(new OrdersModule())
            .build();
    }
}
```

DaggerAppComponent contains the generated code to Configure Modules
public class MyApplication extends Application {

    private static AppComponent component;

    public static AppComponent getComponent() {
        return component;
    }

    public AppComponent createComponent() {
        return DaggerAppComponent.builder()
            .networkModule(new NetworkModule(this))
            .ordersModule(new OrdersModule())
            .build();
    }

    Modules getting configured
public class MyApplication extends Application {

    private static AppComponent component;

    public static AppComponent getComponent() {
        return component;
    }

    public AppComponent createComponent() {
        return DaggerAppComponent.builder()
            .networkModule(new NetworkModule(this))
            .ordersModule(new OrdersTestModule())
            .build();
    }
}
public class AllOrdersActivity extends AppCompatActivity {

@Inject
OrdersRepository ordersRepository;

private AllOrdersViewModel allOrdersViewModel;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    MyApplication.getComponent().inject(this);
    activityAllOrdersBinding = DataBindingUtil.setContentView(this, R.layout.activity_all_orders);
}
}
Dagger 2: Constituents

- **Module**: Source of providing objects to be injected
- **Component**: Generates code which helps in injection
- **Activity**
- **Fragment**
- **Any Other Class**: This is where injection takes place
This talk will clear all your confusions

How to write Testable Code
This talk will clear all your confusions

How to write Testable Code
Now That We Have The Tools Ready

Let’s Start Writing Test Cases
Three Approaches To Testing

Unit Instrumentation And Integration Testing Using Espresso

Unit Testing Using Roboelectric

Pure JVM Testing Using MVVM

Which to Choose?
Performance Analysis
Before That

Let’s talk about Appium a little
Before That

Black Box Testing
Before That

Black Box Testing

Code Agnostic
Before That

Black Box Testing

Code Agnostic

Only UI interactions
Before That

Black Box Testing

Only UI interactions

QA Centric

Code Agnostic
What we will be focusing on

White Box Testing

functions
Classes
UI as well
Variables
What we will be focusing on

White Box Testing

Code Aware

functions
Classes
UI as well
Variables
What we will be focusing on

White Box Testing

Unit Testing

Code Aware

functions
Classes
UI as well
Variables
What we will be focusing on

White Box Testing

Unit Testing
Dev Centric

Code Aware

functions
Classes

UI as well
Variables
Why Espresso?

Closely Integrated With Android Studio
Why Espresso?

Closely Integrated With Android Studio

No External Dependency eg. Selenium Server in case of Appium
Why Espresso?

Closely Integrated With Android Studio

No External Dependency eg. Selenium Server in case of Appium

Can be used both for Unit and Integration Testing
Why Espresso?

Closely Integrated With Android Studio

No External Dependency eg. Selenium Server in case of Appium

Can be used both for Unit and Integration Testing

Removes **Flakiness** By Mocking Intents
Hypnotic Effect of Espresso Intents
Let’s Mock’em
Instrumentation Tests Setup

Configures Test Application to replace Main Application

Test Runner

Test Application

TestModule

FakeDataSource
Instrumentation Tests Setup

Test Runner

Points to the Test Module
Instrumentation Tests Source Set

- androidTest
  - java
    - com.homelane.notetaking
      - custom
      - di
      - notes
      - orderlifecycle
    - orders
      - AllOrdersTest
      - Constants
      - ExampleInstrumentedTest
      - MyTestRunner
    - TestMyApplication
Setting Up An Instrumentation Runner

In build.gradle

defaultConfig {
	namespace applicationId "com.sinca.shoppy"
	namespace minSdkVersion 19
	namespace targetSdkVersion 25
	namespace versionCode 1
	namespace versionName "1.0"

testInstrumentationRunner "com.sinca.shoppy.MyTestRunner"

}
What does the Test Runner Do??

```java
public class MyTestRunner extends AndroidJUnitRunner {

@Override
    public Application newApplication(ClassLoader classLoader,
            String className, Context context)
        throws InstantiationException,
                IllegalAccessException, ClassNotFoundException {
            return super.newApplication(classLoader,
                    TestMyApplication.class.getName(), context);
        }
}
```

Replacing the application class With a Test Application Class
What does the Test Application Do??

```java
public class TestMyApplication extends MyApplication {

    @Override
    public AppComponent.createComponent() {

        return DaggerAppComponent.builder()
            .networkModule(new NetworkModule(this))
            .ordersModule(new OrdersTestModule())
            .build();
    }
}
```

Setting Up Test Modules
Instrumentation Tests Setup

Test Runner

Test Application

TestModule

FakeDataSource
Espresso Commands At a Glance

```java
onView(ViewMatcher)
  .perform(ViewAction)
  .check(ViewAssertion);

onData(ObjectMatcher)
  .DataOptions
  .perform(ViewAction)
  .check(ViewAssertion);
```
public class AllOrdersTest {

    @Rule
    public ActivityTestRule<AllOrdersActivity> mActivityTestRule =
    new ActivityTestRule<AllOrdersActivity>(AllOrdersActivity.class, true, false);

    @Before
    public static void setUp() {
        FakeOrderDataSource.createALL_ORDER_RESPONSE_OBSERVABLE();
    }

    Provides functional testing of a single Activity
public class AllOrdersTest {

    @Rule
    public ActivityTestRule<AllOrdersActivity> mActivityTestRule =
    new ActivityTestRule<AllOrdersActivity>(AllOrdersActivity.class, true, false);

    @Before
    public static void setUp() {
        FakeOrderDataSource.createALL_ORDER_RESPONSE_OBSERVABLE();
    }

    Creating the default response
public class AllOrdersTest {

    public void onExceptionError_checkIfSnackbarIsDisplayed() {

        FakeOrderDataSource.getInstance().create_Exception_Error_Observable( "Internet Security Exception");
        reloadOrdersActivity();

        String text = mActivityTestRule.getActivity().getString(R.string.some_error_occurred);

        onView(allOf(withId(android.support.design.R.id.snackbar_text),
                withText(text)))
                .check(matches(isDisplayed()));
    }
}
public class AllOrdersTest {

    public void onExceptionError_checkIfSnackBarIsDisplayed () {

        FakeOrderDataSource.getInstance().create_Exception_Error_Observable("Internet Security Exception");

        reloadOrdersActivity();

        String text = mActivityTestRule.getActivity().getString(R.string.some_error_occurred);

        onView(allOf(withId(android.support.design.R.id.snackbar_text),
                        withText(text)))
                        .check(matches(isDisplayed()));
    }
}
Replicating the entire scenario altogether

```java
FakeOrderDataSource.getInstance().create_Exception_Error_Observable("Internet Security Exception");
```
public class AllOrdersTest {

    public void onExceptionError_checkIfSnackBarIsDisplayed () {

        FakeOrderDataSource.getInstance().create_Exception_Error_Observable("Internet Security Exception");

        reloadOrdersActivity();

        String text = mActivityTestRule.getActivity().getString(R.string.some_error_occurred);

        onView(allOf(withId(android.support.design.R.id.snackbar_text),
            withText(text)))
            .check(matches(isDisplayed()));

    }

}
public class AllOrdersTest {

    public void onExceptionError_checkIfSnackBarIsDisplayed () {

        FakeOrderDataSource.getInstance().create_Exception_Error_Observable("Internet Security Exception");

        reloadOrdersActivity();

        String text = mActivityTestRule.getActivity().getString(R.string.some_error_occured);

        onView(allOf(withId(android.support.design.R.id.snackbar_text), withText(text)))
            .check(matches(isDisplayed()));
    }

    Fetching string which would be displayed on the snackbar
public class AllOrdersTest {

    public void onExceptionError_checkIfSnackBarIsDispalyed() {
        FakeOrderDataSource.getInstance().create_Exception_Error_Observable("Internet Security Exception");
        reloadOrdersActivity();

        String text = mActivityTestRule.getActivity().getString(R.string.some_error_occurred);

        onView(allOf(withId(android.support.design.R.id.snackbar_text), withText(text)))
            .check(matches(isDisplayed()));
    }

    Checking If a SnackBar gets displayed with an appropriate text
```java
ViewMatcher

onView(allOf(withId(android.support.design.R.id.snackbar_text),
    withText(text)))
    .check(matches(isDisplayed()));
```
Clicking on A Cancelled Order

@Test
public void onCancelledOrderClick_checkIfCancelledOrderPageIsOpened() {

    FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);

    reloadOrdersActivity();

    onView(withText(OrderLifeCycleConstants.STATUS_ORDER_CANCELLED)).perform(click());

    onView(withId(R.id.order_cancelled_text_view)).check(matches(isDisplayed()));
}

Creating Orders List Observable
You would have to rewrite application code for external APIs from your test code.
Replicating the entire scenario altogether

FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);
@Test
public void onCancelledOrderClick_checkIfCancelledOrderPageIsOpened() {

FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);

reloadOrdersActivity();

onView(withId(R.id.order_cancelled_text_view)).check(matches(isDisplayed()));

ViewMatcher

onView(withText(OrderLifeCycleConstants.STATUS_ORDER_CANCELED)).perform(click());

onView(withId(R.id.order_cancelled_text_view)).check(matches(isDisplayed()));
}
@Test
def onCancelledOrderClick_checkIfCancelledOrderPageIsOpened() {

FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);

reloadOrdersActivity();

onView(withText(OrderLifeCycleConstants.STATUS_ORDER_CANCELLED)).perform(click());

onView(withId(R.id.order_cancelled_text_view)).check(matches(isDisplayed()));
}
@Test
public void onCancelledOrderClick_checkIfCancelledOrderPageIsOpened() {

FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);

reloadOrdersActivity();

onView(withText(OrderLifeCycleConstants.STATUS_ORDER_CANCELLED)).perform(click());

onView(withId(R.id.order_cancelled_text_view)).check(matches(isDisplayed()));
}

Checking if the correct page has opened
@Test
def onServerError_checkIfSnackBarIsDisplayedWithCorrectMessage():

FakeOrderDataSource.getInstance().createAllOrderResponseWithServerErrorObservable(SERVER_BUSY_MESSAGE);

reloadOrdersActivity();

onView(allOf(withId(android.support.design.R.id.snackbar_text), withText(SERVER_BUSY_MESSAGE))).
  .check(matches(isDisplayed()));
Replicating the entire scenario altogether

FakeOrderDataSource.getInstance().createAllOrderResponseWithErrorObservable(SERVER_BUSY_MESSAGE);
It took 45 secs to build & Install the app + 4 secs to run the 6 test cases = 49 secs
When to Use Espresso

- For Integration Testing
- To Test On Multiple Devices
- To Test With Actual Data Sources
Robolectric

Not Required

Android SDK
To Run Tests Directly On

Mock JVM
Robolectric Tests Setup

Configures Test Application to replace Main Application
Robolectric Tests Setup

Test Runner

Points to the Test Module

Test Application

TestModule

FakeDataSource
Unit Tests Source Set

- test
  - java
    - com.homelane.notetaking
      - di
      - support
        - AllOrdersActivityTest
        - AllOrdersViewModelTest
        - Constants
        - ExampleUnitTest
        - UnitTestingApplication
      - rx.plugins
@RunWith(RobolectricTestRunner.class)

@org.robolectric.annotation.Config(constants = BuildConfig.class, sdk = 21,
    shadows = {ShadowSnackbar.class}, application = UnitTestingApplication.class)

public class AllOrdersActivityTest {

}
Robolectric Test Class

```java
@RunWith(RobolectricTestRunner.class)
@org.robolectric.annotation.Config(constants = BuildConfig.class,
sdk = 21,
    shadows = {ShadowSnackbar.class},
    application = UnitTestingApplication.class)

public class AllOrdersActivityTest {
}
```

Setting Up Test Application to Inject Mocked Modules
@RunWith(RobolectricTestRunner.class)

public class AllOrdersActivityTest {

    private void reloadOrdersActivity() {

        activity = Robolectric.setupActivity(AllOrdersActivity.class);

        ordersRecyclerView = (RecyclerView) activity.findViewById(R.id.orders_rc);

        allOrdersViewModel = activity.getAllOrdersViewModel();
    }

}
Initialisation Before Every Test

```java
@RunWith(RobolectricTestRunner.class)

public class AllOrdersActivityTest {
    private void reloadOrdersActivity() {
        activity = Robolectric.setupActivity(AllOrdersActivity.class);

        ordersRecyclerView = (RecyclerView) activity.findViewById(R.id.orders_rc);

        allOrdersViewModel = activity.getAllOrdersViewModel();
    }
}
```

Referencing RecyclerViews
@Test
public void onExceptionErrorWhileFetchingOrders_checkIfSnackBarIsDisplayed() {
    FakeOrderDataSource.getInstance().create_Exception_Error_Observable("Internet Security Exception");
    reloadOrdersActivity();
    assertThat(activity.getString(R.string.some_error_occurred), equalsTo(ShadowSnackbar.getTextOfLatestSnackbar()));
}
Replicating the entire scenario altogether
@Test
public void onExceptionErrorWhileFetchingOrders_checkIfSnackBarIsDisplayed() {

FakeOrderDataSource.getInstance().create_Exception_Error_Observable("Internet Security Exception");

reloadOrdersActivity();

assertThat(activity.getString(R.string.some_errorOccurred), equalTo(ShadowSnackbar.getTextOfLatestSnackbar()));
}

Checking If Snackbar displays the correct text or not
@Test
public void onOrdersLoaded_checkIfStatusLabellingOfOrderItemsIsCorrect() {
    
    for (int i = 0; i < OrderLifeCycleConstants.ORDER_STATUSES_ARRAY.length; i++) {
        View itemView = ordersRecyclerView.getChildAt(i);
        TextView statusTextView = (TextView) itemView.findViewById(R.id.order_status_text_view);
        assertTrue(statusTextView.getText().toString().equals(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY[i]));
    }
}
@Test
public void onOrdersLoaded_checkIfStatusLabellingOfOrderItemsIsCorrect() {
    ...

    for (int i = 0; i < OrderLifeCycleConstants.ORDER_STATUSES_ARRAY.length; i++) {
        View itemView = ordersRecyclerView.getChildAt(i);

        TextView statusTextView = (TextView) itemView.findViewById(R.id.order_status_text_view);

        assertTrue(statusTextView.getText().toString().equals(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY[i]));
    }
}

Checking If Every Order Displays The Correct Status Or Not
@Test
public void onDeliveryOrderClick_checkIfDeliveryOrderPageIsOpened() {

FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);

reloadOrdersActivity();

ordersRecyclerView.getChildAt(0).performClick();

assertNextActivity(activity, DeliveryActivity.class);
}
@Test
public void onDeliveryOrderClick_checkIfDeliveryOrderPageIsOpened() {

FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);

reloadOrdersActivity();

ordersRecyclerView.getChildAt(0).performClick();

assertNextActivity(activity, DeliveryActivity.class);
}

Checking if correct activity has been opened or not
FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);
It took 18 secs to Shadow Android Code To JVM + 5 secs to run the 6 test cases = 23 secs
When to Use Robolectric

- For Testing Directly On JVM
When to Use Roboelectric

- For Testing Directly On JVM

- Very Useful When App Is Not Well Architectured
When to Use Robolectric

- For Testing Directly On JVM
- Very Useful When App Is Not Well Architectured
- Also very Helpful for testing view properties like colour, style etc.
This talk will clear all your confusions

Which **Framework** to choose?
This talk will clear all your confusions

Which Framework to choose?
Testing The View Model
Any Guesses

How much time it took ??
It took 180 ms To Run The Same Test Cases
How Did It Happen???

Lets See...!!!
public class AllOrdersViewModel {

    public AllOrdersViewModel(
        OrdersRepository repository) {
        ordersRepository = repository;
    }

    private void loadOrders(final boolean showLoadingUI) {
        if (showLoadingUI) {
            dataLoading.set(true);
        }

        ordersRepository.getOrdersResponse(new OrdersDataSource.LoadOrdersCallback() {
            
        });
    }
}
public class AllOrdersViewModel {

    public AllOrdersViewModel(
        OrdersRepository repository) {
        ordersRepository = repository;
    }

    private void loadOrders(final boolean showLoadingUI) {
        if (showLoadingUI) {
            dataLoading.set(true);
        }

        ordersRepository.getOrdersResponse(new OrdersDataSource.LoadOrdersCallback() {
            Orders Are being fetched from the Repository
        });
    }
}
@Override
public void onCompleted() {
}

@Override
public void onError(Throwable e) {
    dataLoading.set(false);
    snackbarText.set(exceptionErrorText);
}

@Override
public void onNext(AllOrdersResponse allOrdersResponse) {
    dataLoading.set(false);
    if (allOrdersResponse.isSuccess()) {
        ordersList.clear();
        ordersList.addAll(allOrdersResponse.getOrders());
    }

    else {
        snackbarText.set(allOrdersResponse.getError_message());
    }

    Observable Which Would directly Update Views In The Activity
@Override
public void onCompleted() {
}

@Override
public void onError(Throwable e) {

dataLoading.set(false);

snackbarText.set(exceptionErrorText);
}

@Override
public void onNext(AllOrdersResponse allOrdersResponse) {

dataLoading.set(false);
if (allOrdersResponse.isSuccess()) {

    ordersList.clear();
    ordersList.addAll(allOrdersResponse.getOrders());
}

else {
    snackbarText.set(allOrdersResponse.getError_message());
}
```java
@Override
public void onCompleted() {
}

@Override
public void onError(Throwable e) {
    dataLoading.set(false);
    snackbarText.set(exceptionErrorText);
}

@Override
public void onNext(AllOrdersResponse allOrdersResponse) {
    dataLoading.set(false);
    if (allOrdersResponse.isSuccess()) {
        ordersList.clear();
        ordersList.addAll(allOrdersResponse.getOrders());
    } else {
        snackbarText.set(allOrdersResponse.getError_message());
    }
}
```
The Magic Of MVVM + DataBinding

Visibility Of This ProgressBar would depend on ObservableBoolean variable
That Means I can Directly Test The View Model

And See Whether The Business Logic Works Fine Or Not
Yesssssss

And Since The View Model is Simply a Java Class

Without Any Android Specific Code

The Tests Run Very Fast On

JVM
@Test
test void afterSuccessFullOrdersLoading_CheckIfProgressBarIsNotDisplayed() {

    FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);

    AllOrdersViewModel allOrdersViewModel = constructAndGetAllOrdersViewModel(EXCEPTION_ERROR_SNACKBAR_TEXT);

    allOrdersViewModel.loadOrders();

    assertFalse(allOrdersViewModel.getDataLoading().get());
}
@Test
public void afterSuccessFullOrdersLoading_CheckIfProgressBarIsNotDisplayed() {

    FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);

    AllOrdersViewModel allOrdersViewModel = constructAndGetAllOrdersViewModel(EXCEPTION_ERROR_SNACKBAR_TEXT);

    allOrdersViewModel.loadOrders();

    assertFalse(allOrdersViewModel.getDataLoading().get());
}
@Test
def afterSuccessFullOrdersLoading_CheckIfProgressBarIsNotDisplayed() {
    FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifeCycleConstants.ORDER_STATUSES_ARRAY);

    AllOrdersViewModel allOrdersViewModel = constructAndGetAllOrdersViewModel(EXCEPTION_ERROR_SNACKBAR_TEXT);

    allOrdersViewModel.loadOrders();

    assertFalse(allOrdersViewModel.getDataLoading().get());
}

Checking of dataLoading is false
What’s New ??
Local DB Testing

Domain Layer

Presentation Layer

Data Layer
Local DB Testing

Room

Test Database Migrations

Test Database Queries
Local DB Testing

- Domain Layer
- Presentation Layer
- Data Layer
Check out my in-depth blog posts on medium

https://medium.com/@kapil.bakshi
JUnitParams Support

@RunWith(JUnitParamsRunner.class)
public class PersonTest {
    @Test
    @Parameters({"17, false", "22, true" })
    public void personIsAdult(int age, boolean valid) throws Exception {
        assertThat(new Person(age).isAdult(), is(valid));
    }
}
Testing On Multiple Devices
Testing On Multiple Devices

Happy and Relaxed

After An Important Release

✅ Testing On Stage
✅ Testing On PreProd
✅ Testing On 3-4 Devices
Testing On Multiple Devices

- Investigation: Recommended
- Adoption: 41.4% of Total DAU
- Stability: 97.0% Crash-free Users
- Median Session Length: 0:38 sec
- Investigate this release's top issues in Crashlytics.

### Devices
- Samsung: 55%
- OPPO: 41%

### Operating Systems
- 5
- 85%
- 4
- 15%
Testing On Multiple Devices

Fire Base Test Labs

Device Farm

SAUCE LABS
Robo Tests

Max depth
The deepest level that Robo will traverse within an app UI.
50

Test account credentials (Optional)
If your app requires custom login, enter the resource names of the login elements and the login credentials.

Enter username resource
Enter username

Enter password resource
Enter password

Additional fields (Optional)
If your app has additional elements that require input text, enter the resource names and input strings below.

Enter resource name
Enter value

Randomly Tests App’s UI
Can Supply Inputs for EditTexts
Can Choose Maximum Depth of Test Traversal

START TESTS
Run From Android Studio

Deployment Target Options

Target: Firebase Test Lab Device Matrix

Matrix configuration: Sample Spark configuration (4)

Run NotePadTest

- Nexus 7 (2013), ASUS | Android 5.0.x, API Level 21 (Lollipop) | English | Portrait
  - com.example.android.notepad.NotePadTest
    - testAddNote

- Nexus 7 (2013), ASUS | Android 4.4.x, API Level 19 (KitKat) | English | Portrait
  - com.example.android.notepad.NotePadTest
    - testAddNote

- Nexus 5, LG | Android 5.1.x, API Level 22 (Lollipop) | English | Portrait
  - com.example.android.notepad.NotePadTest
    - testAddNote
Get Very Detailed Reports

Device Details:

- Manufacturer: unknown
- Brand: generic
- Model: GCE x86 phone
- Device: gce_x86
- Release version: 6.0.1
- SDK version: 23
- Play Services Version Code: 11060470
Cons

1. Fewer Of Devices
2. Supports Only Android Instrumentation Tests And Robo Tests
3. Network Speed Throttling Not Supported
AWS Device Farm

- Built-in: Explorer
- Built-in: Fuzz
- Appium Java JUnit
- Appium Java TestNG
- Appium Python
- Calabash
- Instrumentation
- UI Automator
AWS Device Farm

Testing At Different Network Speeds
# AWS Device Farm

## More Devices To Test On

<table>
<thead>
<tr>
<th>Device</th>
<th>OS</th>
<th>Version</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTC One M8 (AT&amp;T)</td>
<td>Android</td>
<td>4.4.4</td>
<td>Phone</td>
</tr>
<tr>
<td>LG G Flex (AT&amp;T)</td>
<td>Android</td>
<td>4.2.2</td>
<td>Phone</td>
</tr>
<tr>
<td>LG G2 (AT&amp;T)</td>
<td>Android</td>
<td>4.4.2</td>
<td>Phone</td>
</tr>
<tr>
<td>LG Optimus L70 (MetroPCS)</td>
<td>Android</td>
<td>4.4.2</td>
<td>Phone</td>
</tr>
<tr>
<td>Motorola DROID Ultra (Verizon)</td>
<td>Android</td>
<td>4.4.4</td>
<td>Phone</td>
</tr>
<tr>
<td>Samsung Galaxy Note 3 (AT&amp;T)</td>
<td>Android</td>
<td>4.4.2</td>
<td>Phone</td>
</tr>
<tr>
<td>Samsung Galaxy Note 3 (Verizon)</td>
<td>Android</td>
<td>4.4.4</td>
<td>Phone</td>
</tr>
<tr>
<td>Samsung Galaxy Note 4 (AT&amp;T)</td>
<td>Android</td>
<td>5.0.1</td>
<td>Phone</td>
</tr>
<tr>
<td>Samsung Galaxy Note 4 (Verizon)</td>
<td>Android</td>
<td>5.0.1</td>
<td>Phone</td>
</tr>
<tr>
<td>Samsung Galaxy S3 (T-Mobile)</td>
<td>Android</td>
<td>4.3</td>
<td>Phone</td>
</tr>
<tr>
<td>Samsung Galaxy S3 (Verizon)</td>
<td>Android</td>
<td>4.4.2</td>
<td>Phone</td>
</tr>
<tr>
<td>Samsung Galaxy S3 LTE (T-Mobile)</td>
<td>Android</td>
<td>4.3</td>
<td>Phone</td>
</tr>
<tr>
<td>Samsung Galaxy S2 Mini (AT&amp;T)</td>
<td>Android</td>
<td>4.4.2</td>
<td>Phone</td>
</tr>
</tbody>
</table>
The Only Con

Not Able To Run Specific TestNG Test Suites
Sauce Labs

Supports Different Testing Frameworks
Sauce Labs

Sauce Labs Acquired TestObject to enable testing on Real Devices
DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability("deviceName", deviceName);
capabilities.setCapability("platformName", AppConfig.INSTANCE.get("platformName"));
capabilities.setCapability("platformVersion", androidVersion);
capabilities.setCapability("appPackage", appPackage);
capabilities.setCapability("resetKeyboard", true);
capabilities.setCapability("testobject_api_key", "89HG598ZXSD6YH78BEF9E5796C108A0F");

MobileDriver mobileDriver = new AndroidDriver(new URL("https://eu1.appium.testobject.com/wd/hub"), capabilities);

Just have to change The Url To Appium Hosted On TestObject
Cons

Network Speed Throttling Is Not Supported
This talk will clear all your confusions

How to Test on Multiple Devices
This talk will clear all your confusions

How to Test on Multiple Devices
Oh Lord Of Test Driven Development

Cast Your Light Upon Us
For The Release Is Critical

And Prone To Bugs
Entire Content Shared

@akapil167
Entire Content Shared

@akapil167

Follow to get latest updates
GREAT INDIAN DEVELOPER SUMMIT 2019
Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com
@greatindiandev
bit.ly/gidslinkedind
facebook.com/gids19
bit.ly/saltmarchyoutube
flickr.com/photos/saltmarch/