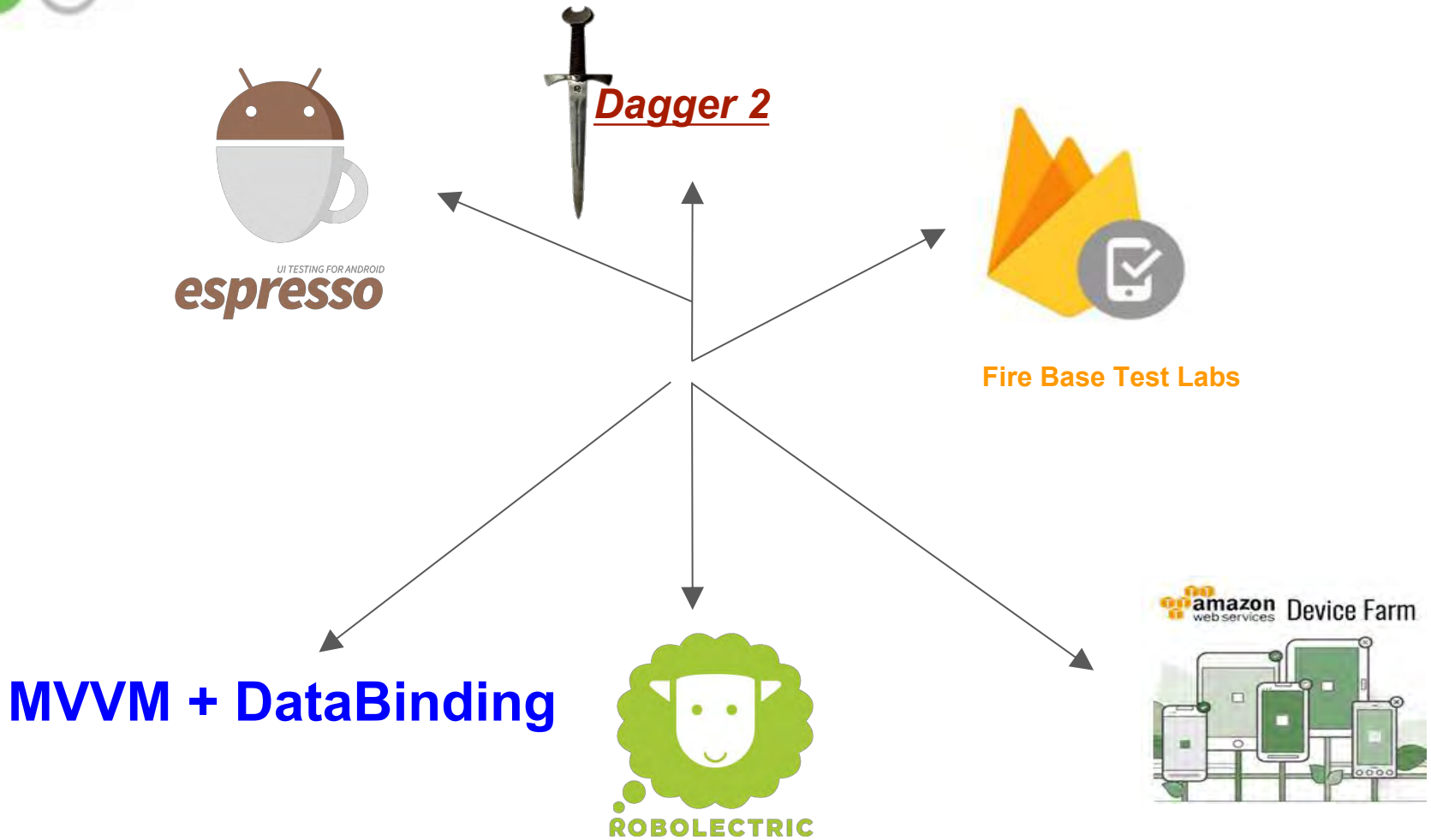




Production level Test Driven Development





About Me

KAPIL BAKSHI

FOODIE

TRAVELLER

MUSIC LOVER

SUSPENSE WATCHER



LOGISTICS + FINTECH + EDTECH

Software Engineer



akapil167



kapilbakshi167@gmail.com

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 ?

Writing **Testable**
Code

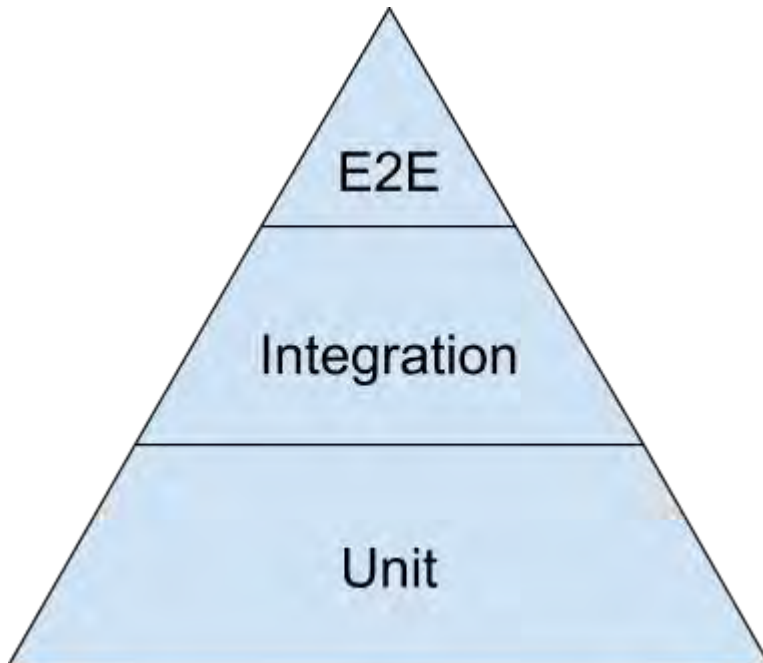
Running Tests On
Different Devices

Umm.. **Unit Testing**,
Instrumentation
Testing or **End To**
End Tests ??





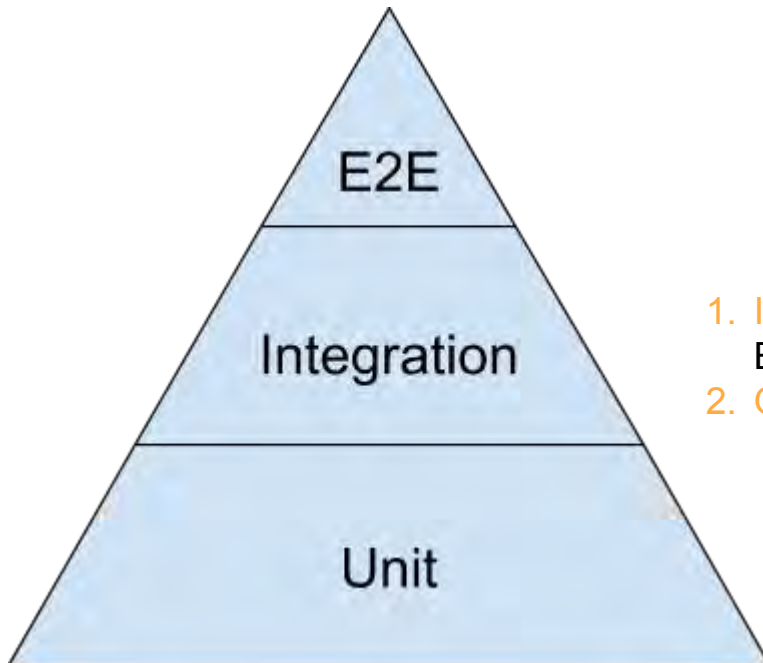
Different Types Of Testing



1. **Idea** :- Testing Business Logic
2. **No external dependency**
3. **Options** :- Robolectric, JUnit, Mockito



Different Types Of Testing

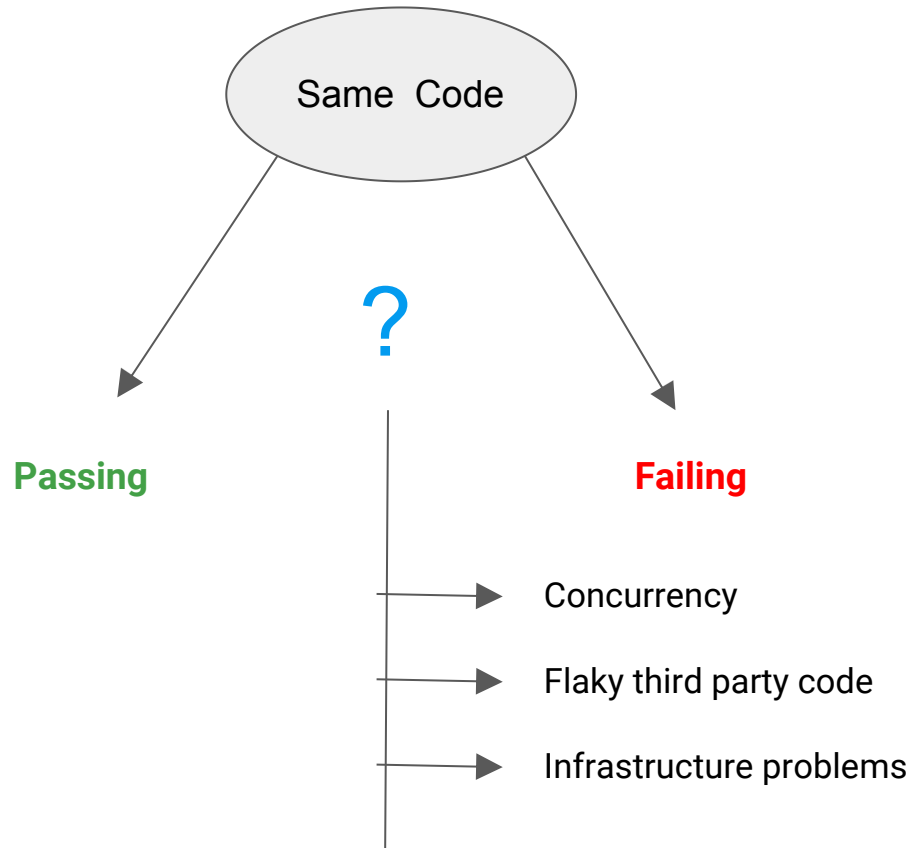


1. **Idea** :- Testing User Dependencies, Behaviour, Integration
2. **Options** :- Espresso, Robotium, Appium



Flakiness and Its Mitigation

Ferocious Flaky





Flakiness and Its Mitigation



Ferocious Flaky

Same Code

?

Passing

Failing

- Concurrency
- Flaky third party code
- Infrastructure problems

Mocking Dependencies
Relying More on Unit Tests

Hero Hermetic to the Rescue





Genuine Production Level Scenarios



Testing Error Handling



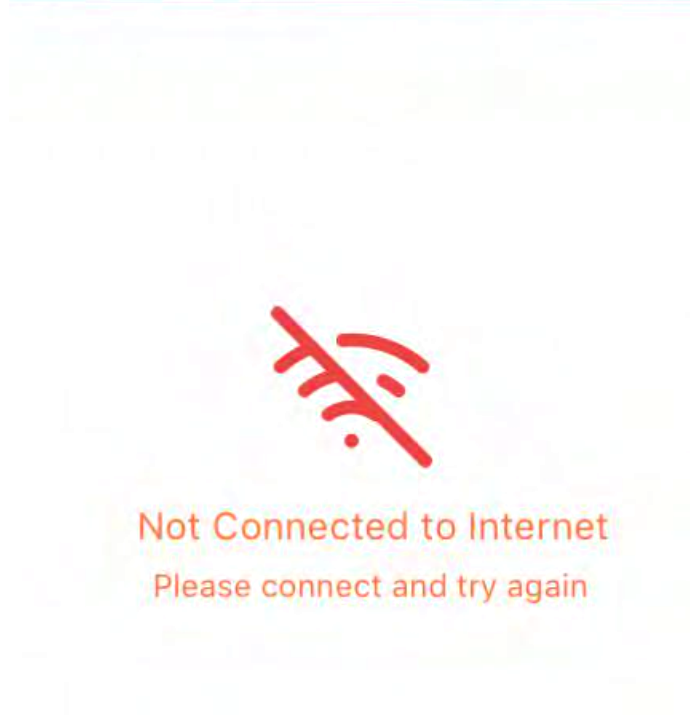
Not Connected to Internet
Please connect and try again



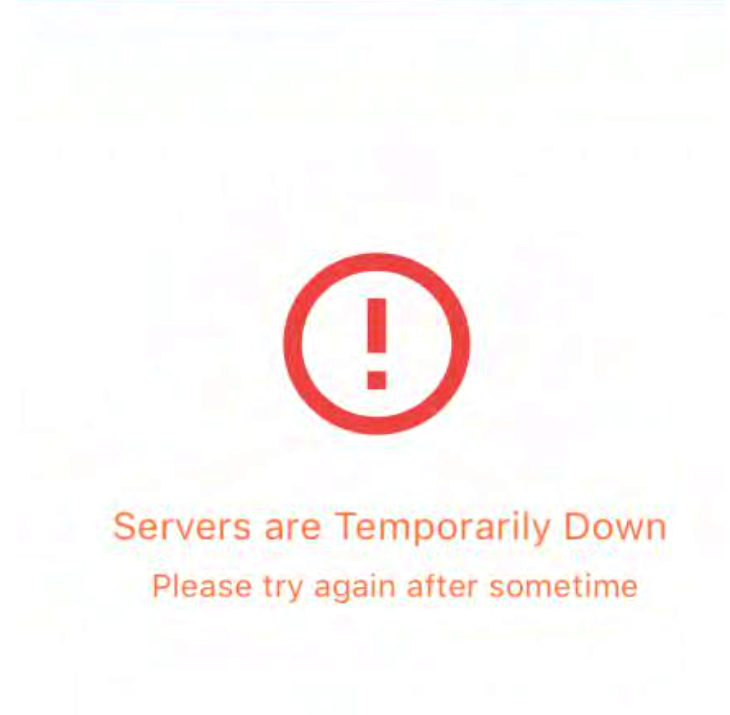
Servers are Temporarily Down
Please try again after sometime



Now to Test this



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



Would you actually stop your server ????



Now to Test this



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




iPhone 7 (Black, 32 GB) \$735

☰ Your Orders 



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



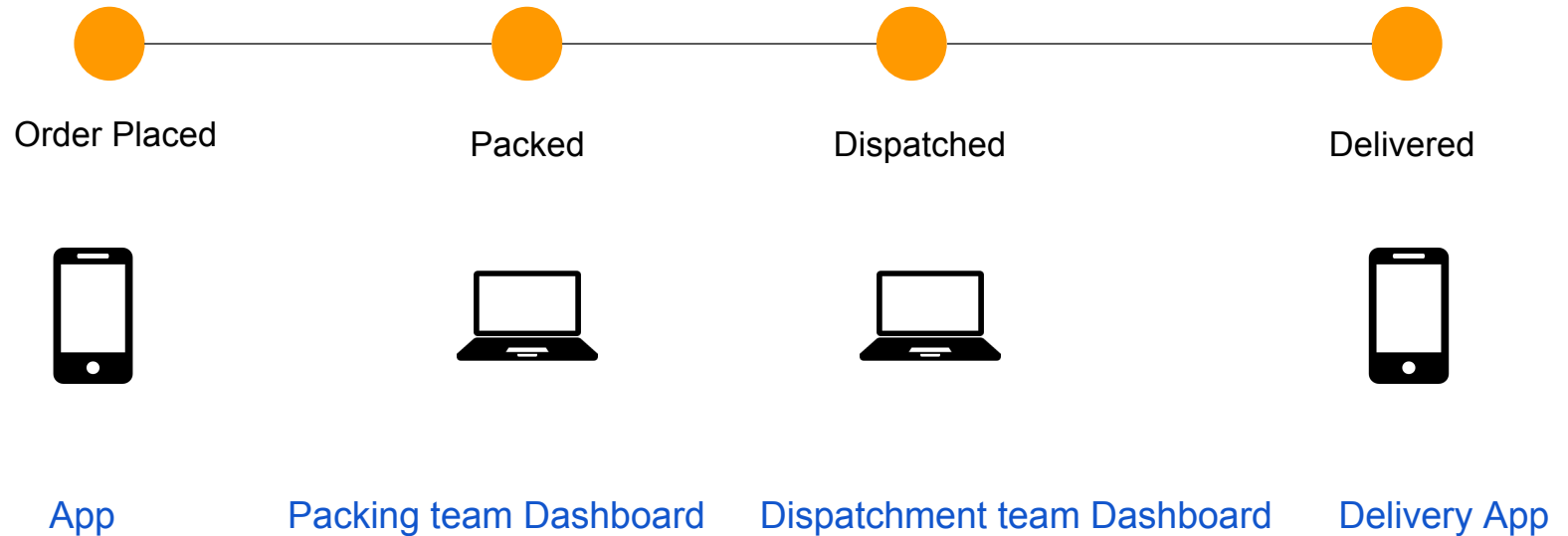
Sony BRAVIA X9300E \$1120
Cancelled on 29th April 2016 11:00 AM
Refund Initiated



What would happen if you
Don't test this **Hermetically**

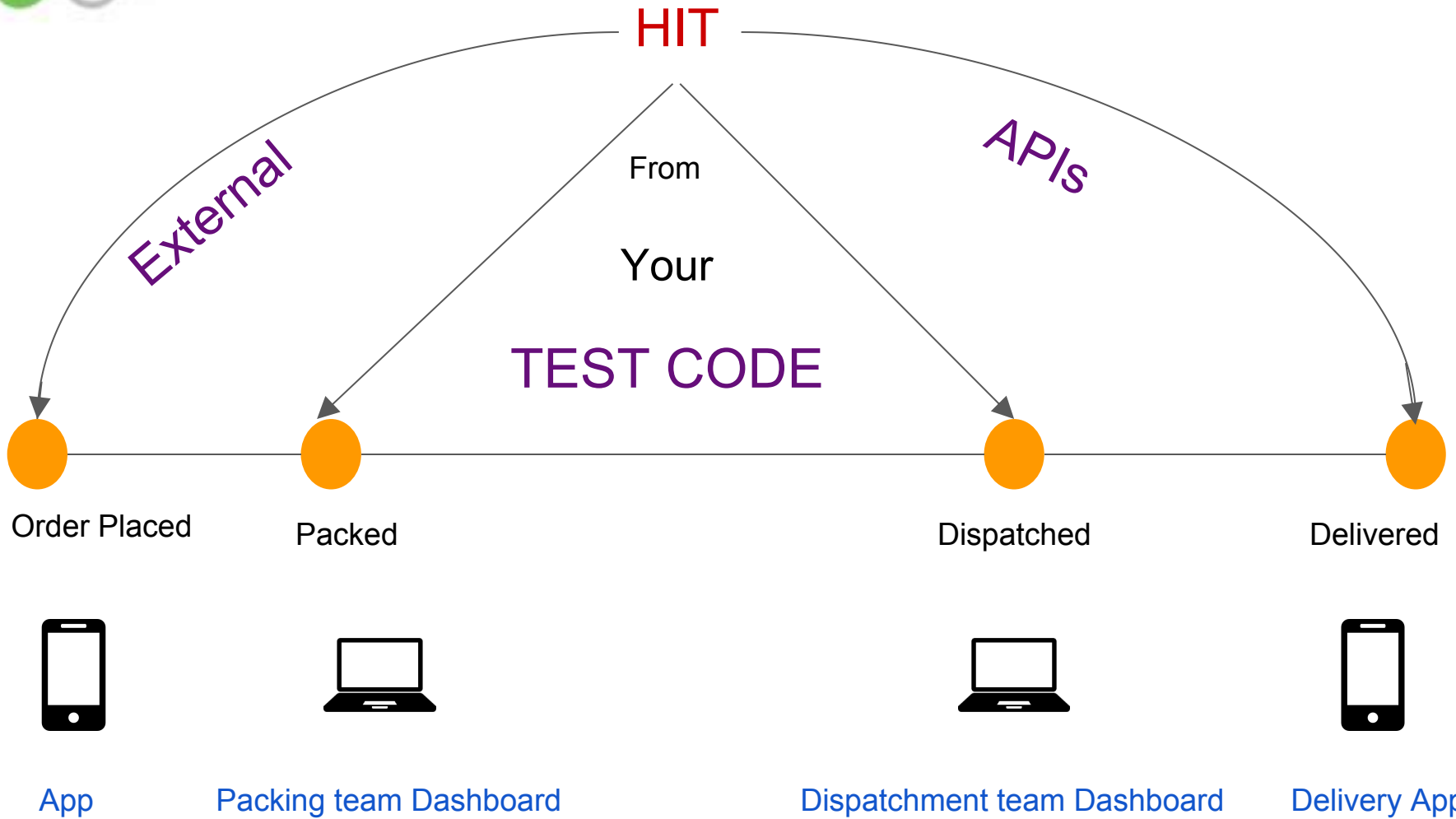


Handle Complex Order Lifecycle





You would have to





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 Test cases than to develop features



Then you'll realize

You are testing **What You Haven't Even
Coded**



Then you'll realize

The goal of testing **The Code** you have actually written gets

Farther .. Farther Farther..... Farther **Away**





Then solution is quite simple

Let the Code Take Control Of Everything

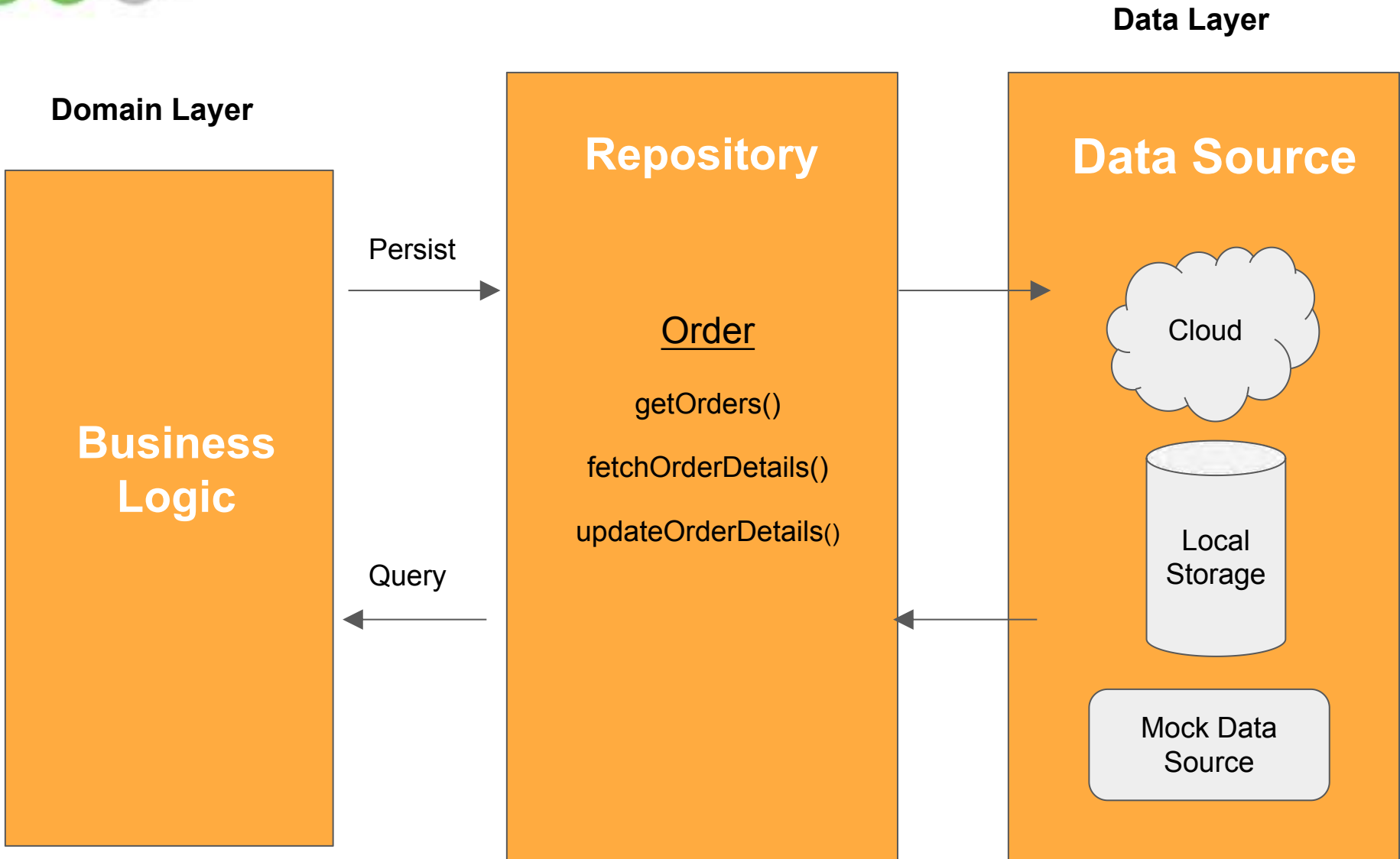


Let's Explore How



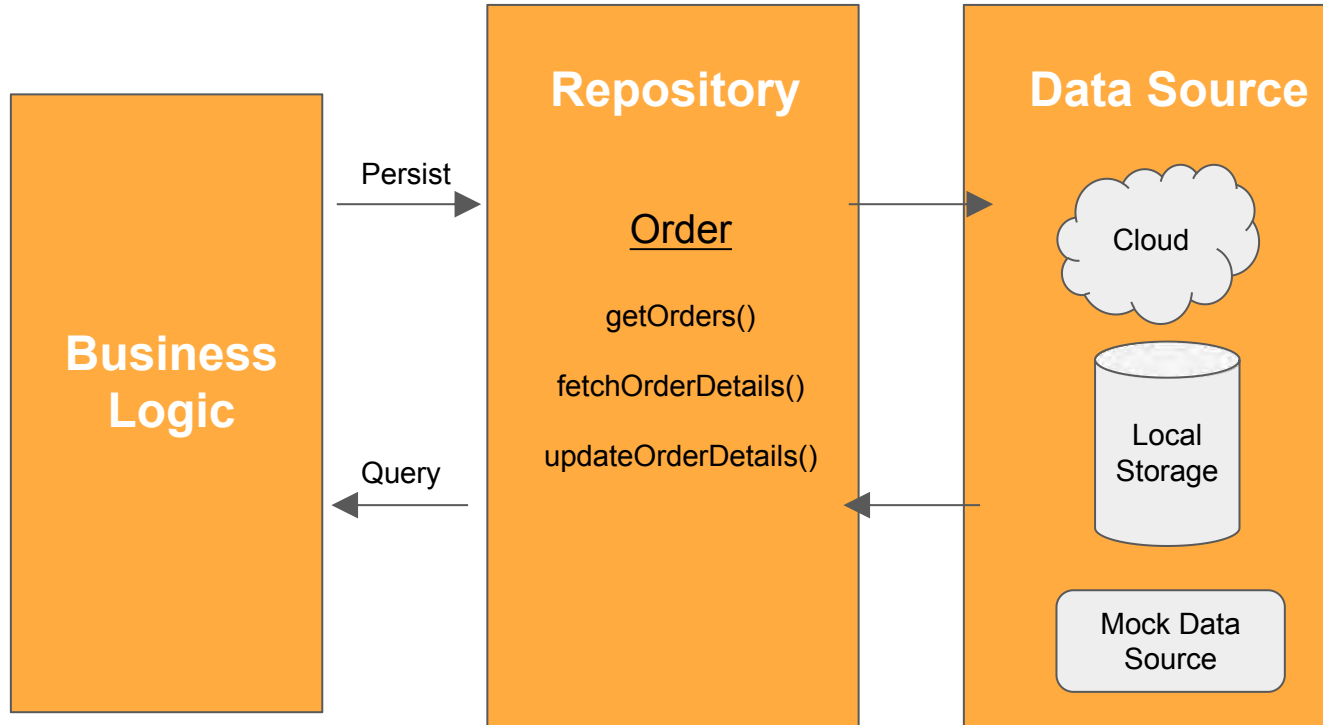


Repository Pattern





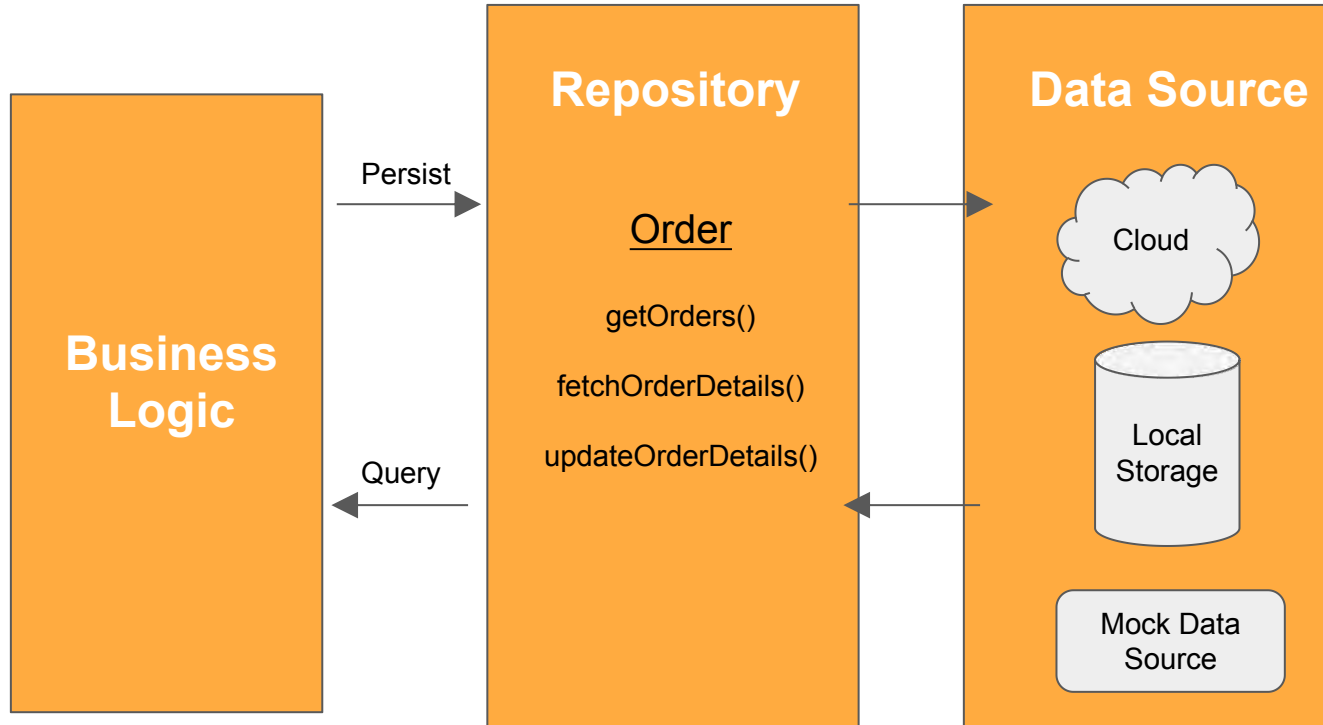
Repository Pattern - Advantages



Provides Abstraction of Data



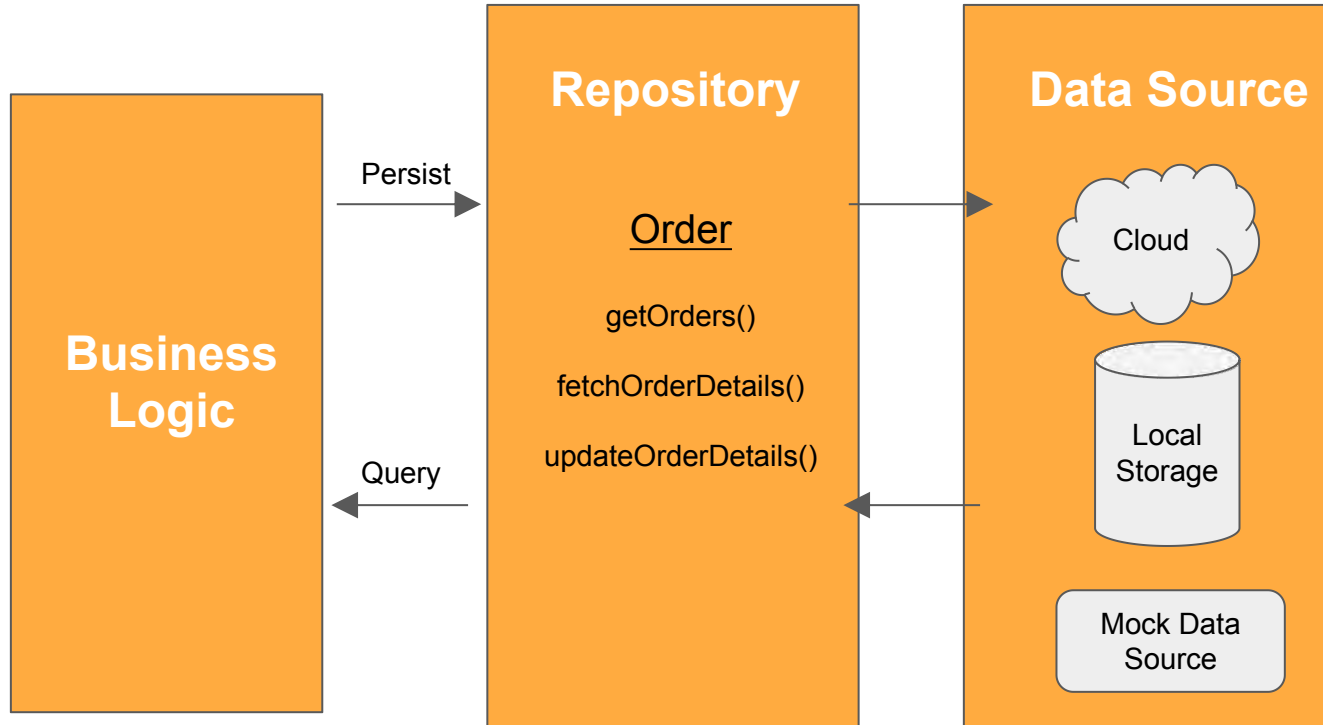
Repository Pattern - Advantages



Makes the code Highly Maintainable and Extensible



Repository Pattern - Advantages



Makes the code Highly Configurable and Testable



Repository Pattern - In Action



OrdersDataSource

```
public interface OrdersDataSource {  
  
    interface LoadOrdersCallback {  
  
        void onGetOrdersResponse(Observable<AllOrdersResponse>  
ordersResponseObservable);  
    }  
  
    void getOrdersResponse(@NonNull OrdersDataSource.LoadOrdersCallback  
callback);  
  
}
```

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



OrdersRepository

```
public class OrdersRepository implements OrdersDataSource {
```

```
private final OrdersDataSource ordersDataSource;
```

```
private OrdersRepository (
```

```
    @NonNull OrdersDataSource ordersDataSource ) {
```

```
    this.ordersDataSource = ordersDataSource;
```

```
}
```

```
@Override
```

```
public void getOrdersResponse(@NonNull final OrdersDataSource.LoadOrdersCallback  
callback) {
```

```
    ordersDataSource.getOrdersResponse(new OrdersDataSource.LoadOrdersCallback() {
```

```
        @Override
```

```
        public void onGetOrdersResponse(Observable<AllOrdersResponse>
```

```
ordersResponseObservable) {
```

```
            callback.onGetOrdersResponse(ordersResponseObservable);
```

```
        }
```

```
    });
```

```
}
```

← Accepting a
Data Source



OrdersRemoteDataSource

```
public class OrdersRemoteDataSource implements OrdersDataSource {
```

```
    private static OrdersRemoteDataSource INSTANCE;
```

```
    @Inject  
    Retrofit retrofit;
```

```
    @Override
```

```
    public void getOrdersResponse(@NonNull LoadOrdersCallback  
callback) {
```

```
        MyApplication.getComponent().inject(this);
```

```
        NetworkApis networkApis = retrofit.create(NetworkApis.class);
```

```
        callback.onGetOrdersResponse(networkApis.getOrders());
```

```
    }
```

```
    public static OrdersRemoteDataSource getInstance() {
```

```
        if (INSTANCE == null) {
```

```
            INSTANCE = new OrdersRemoteDataSource();
```

```
        }
```

```
        return INSTANCE;
```

```
    }
```

```
}
```

Fetching Orders From
The Server Using
Retrofit



FakeDataSource

```
public class FakeOrderDataSource implements OrdersDataSource {
```

```
    @Override
```

```
    public void getOrdersResponse(@NonNull LoadOrdersCallback  
callback) {
```

```
        callback.onGetOrdersResponse(getAllOrderResponseObservable());
```

```
    }
```

```
    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);  
    }
```

↑
Fetching an Observable
Of Mocked Orders



FakeDataSource

```
public class FakeOrderDataSource implements OrdersDataSource {
```

```
    @Override
```

```
    public void getOrdersResponse(@NonNull LoadOrdersCallback callback) {  
        callback.onGetOrdersResponse(getAllOrderResponseObservable());  
    }
```

```
    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);
```

```
    }
```

```
}
```

Creates All Orders
response which can
be modified further





FakeDataSource

```
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());  
    }  
  
}
```

Fetching an Observable Of
Mocked Orders as per the
given statuses



FakeDataSource

```
public class FakeOrderDataSource implements OrdersDataSource {  
    public void createAllOrderResponseWithServerErrorObservable(String errorMessage) {  
        reCreateAll_Order_Response();  
        addErrorToAllOrdersResponse(errorMessage);  
        toggleSuccess(false);  
        ALL_ORDER_RESPONSE_OBSERVABLE = Observable.just(getAllOrderResponse());  
    }  
}
```

Creates All Orders
Observable with an Error
to mock Server Error



FakeDataSource

```
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



**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





Modules In Dagger 2

@Module

```
public class OrdersModule {
```

Responsible for providing
objects which can be
injected

@Provides

@Singleton

```
public OrdersRepository providesNotesRepository() {
```

```
    return OrdersRepository.getInstance(  
        OrdersRemoteDataSource.getInstance());
```

```
}
```

```
}
```

Used for methods which provide
objects for dependencies injection

Notice Remote Order Data Source
is being used here



Modules In Dagger 2

Test Order Module

```
public class OrdersTestModule extends OrdersModule
{
    @Override
    public OrdersRepository providesNotesRepository() {
        return OrdersRepository.getInstance(
            FakeOrderDataSource.getInstance());
    }
}
```

←
Notice Mocked Order
Data Source is being used
here



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.



How does Injection Take Place

```
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();
```

```
}
```

```
@Override
```

```
public void onCreate() {
```

```
    super.onCreate();
```

```
    component = createComponent();
```

```
}
```

```
}
```

DaggerAppComponent contains
the generated code to Configure
Modules



How does Injection Take Place

```
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();
```

```
}
```

```
@Override
```

```
public void onCreate() {
```

```
    super.onCreate();
```

```
    component = createComponent();
```

```
}
```

```
}
```

← Modules getting configured



How does Injection Take Place While Testing

```
public class TestMyApplication extends Application {
```

```
    @Override
```

```
    public AppComponent createComponent() {
```

```
        return DaggerAppComponent.builder()
```

```
            .networkModule(new NetworkModule(this))
```

```
            .ordersModule(new OrdersTestModule())
```

```
            .build();
```

```
    }
```

Notice Test Module
Getting Configured





@Inject Annotation

```
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);  
    }  
    private void setViewModel() {  
        allOrdersViewModel = findOrCreateViewModel();  
        activityAllOrdersBinding.setAllOrdersViewModel(allOrdersViewModel);  
    }  
  
    @Override  
    public void onResume() {  
        super.onResume();  
        allOrdersViewModel.loadOrders();  
    }  
}
```

Injection Taking Place



View Model

```
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() {
```

Accepting a
Repository



Orders Are being
fetched from the
Repository





```
ordersResponseObservable.subscribeOn(Schedulers.io())  
    .observeOn(AndroidSchedulers.mainThread())  
    .subscribe(new Observer<AllOrdersResponse>() {  
        @Override  
        public void onCompleted() {  
        }  
    });
```

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

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

Handling
Exceptions

Orders Are being
fetched from the
Repository



Now That We Have The
Tools Ready



Let's Start Writing **Test Cases**



Three Approaches To Testing



Unit Instrumentation And
Integration Testing Using
Espresso



Which to Choose ?



Unit Testing Using
Robolectric



Performance
Analysis



Pure JVM Testing Using
MVVM



So, Why Espresso ?



Closely Integrated With Android
Studio



No External Dependency eg.
Selenium Server in case of Appium



So, Why Espresso ?



Can be used both for Unit and
Integration Testing



Removes **Flakiness** By Mocking
Intents



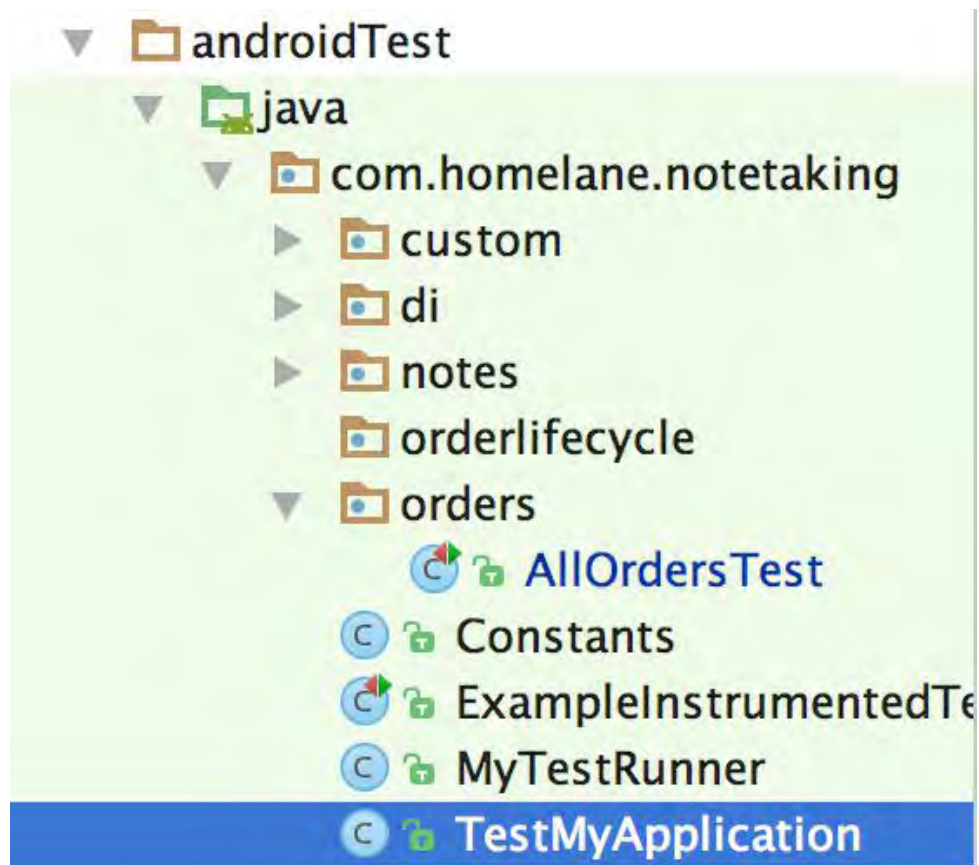
Hypnotic Effect of Espresso Intents

Let's Mock'em





Instrumentation Tests Source Set





Setting Up An Instrumentation Runner



In build.gradle

```
defaultConfig {  
    applicationId "com.sinca.shoppay"  
    minSdkVersion 19  
    targetSdkVersion 25  
    versionCode 1  
    versionName "1.0"  
    testInstrumentationRunner "com.sinca.shoppay.MyTestRunner"  
}
```

Test Runner
Class



What does the Test Runner Do ??



```
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 ??



```
public class TestMyApplication extends MyApplication {
```

```
    @Override
```

```
    public AppComponent createComponent() {
```

```
        return DaggerAppComponent.builder()
```

```
            .networkModule(new NetworkModule(this))
```

```
            .ordersModule(new OrdersTestModule())
```

```
            .build();
```

```
    }
```

```
}
```

Setting Up Mock Modules





Espresso Commands At a Glance

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

```
onData(ObjectMatcher)  
    .DataOptions  
    .perform(ViewAction)  
    .check(ViewAssertion);
```



View Matchers

USER PROPERTIES

```
withId(...)  
withText(...)  
withTagKey(...)  
withTagValue(...)  
hasContentDescription(...)  
withContentDescription(...)  
withHint(...)  
withSpinnerText(...)  
hasLink()   
hasEllipsisText()  
hasMultiLineText()
```

UI PROPERTIES

```
isVisible()  
isCompletelyDisplayed()  
isEnabled()  
hasFocus()  
isClickable()  
isChecked()
```

HIERARCHY

```
withParent(Matcher)  
withChild(Matcher)  
hasDescendant(Matcher)  
isDescendantOfA(Matcher)  
hasSibling(Matcher)  
isRoot()
```

INPUT

```
supportsInputMethods(...)  
hasDPEAction(...)
```

CLASS

```
isAssignableFrom(...)  
withClassName(...)
```

ROOT MATCHERS

Data Options

```
ListAdapter(Matcher)  
atPosition(Integer)  
onChildView(Matcher)
```

View Actions

CLICK/PRESS

```
click()  
doubleClick()  
longClick()  
pressBack()  
pressDPEActionButton()  
pressKey([int/EspressoKey])  
pressMenuKey()  
closeSoftKeyboard()  
openLink()
```

GESTURES

```
scrollTo()  
swipeLeft()  
swipeRight()  
swipeUp()  
swipeDown()
```

TEXT

```
clearText()  
typeText(String)
```



Espresso Commands At a Glance

USER PROPERTIES

- withId(...)
- withText(...)
- withTagKey(...)
- withTagValue(...)
- hasContentDescription(...)
- withContentDescription(...)
- withHint(...)
- withInnerText(...)
- has(Regex)
- hasEllipsizedText()
- hasMultiLineText()

UI PROPERTIES

- isVisible()
- isCompletelyDisplayed()
- isEnabled()
- hasFocus()
- isClickable()
- isChecked()
- isNotChecked()
- withEffectiveness(...)
- isSelected()

OBJECT MATCHER

- allOf(Matchers)
- anyOf(Matchers)
- is(...)
- not(...)
- endsWith(String)
- startsWith(String)
- instanceOf(Class)

VIEW MATCHERS

- withParent(Matcher)
- withChild(Matcher)
- hasDescendant(Matcher)
- isDescendantOfA(Matcher)
- hasSibling(Matcher)
- isRoot()

INPUT

- perform(InputActions(...))
- hasIMEAction(...)

CLASS

- isAssignableFrom(...)
- withClassName(...)

ROOT MATCHERS

- isFocusable()
- isTouchable()
- isDialog()
- withDecorView()
- isPlatformPopup()

SEE ALSO

- Preference matchers
- Cursor matchers
- Layout matchers

atPosition(Integer)

onChildView(Matcher)

View Actions

CLICK/PRESS

- click()
- doubleClick()
- longClick()
- pressBack()
- pressActionButton()
- pressKey(Int, EspressoKey)
- pressKeyCode()
- closeSoftKeyboard()
- openInk()

GESTURES

- scrollTo()
- swipeLeft()
- swipeRight()
- swipeUp()
- swipeDown()

TEXT

- clearText()
- typeText(String)
- typeTextIntofocusView(String)
- replaceText(String)

View Assertions

matches(Matcher)

- doesNotExist()
- selectedDescendantsMatch(...)

LAYOUT ASSERTIONS

- hasEllipsizedText(Matcher)
- hasMultiLineButtons()
- hasNoLaps([Matcher])

POSITION ASSERTIONS

- isLeftOf(Matcher)
- isRightOf(Matcher)
- isLeftAlignedWith(Matcher)
- isRightAlignedWith(Matcher)
- isAbove(Matcher)
- isBelow(Matcher)
- isBottomAlignedWith(Matcher)
- isTopAlignedWith(Matcher)



Espresso Testing In Action



Provides functional testing
of a single Activity

```
public class AllOrdersTest {
```

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

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

```
    @Test  
    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()));
```

```
    }
```

Creating Order Observable
With An Exception



Espresso Testing In Action



```
public class AllOrdersTest {
```

```
    @Rule
```

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

```
    @BeforeClass
```

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

```
    @Test
```

```
    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()));
```

```
    }
```

Checking If a SnackBar gets displayed with an appropriate text





Espresso Testing In Action



ViewMatcher



```
onView(allOf(withId(android.support.design.R.id.snackbar_text),  
withText(text)))
```

```
.check(matches(isDisplayed()));
```

```
}
```



ViewAssertion



Clicking on A Cancelled Order



Creating Orders List
Observable

```
@Test  
public void onCancelledOrderClick_checkIfCancelledOrderPagelsOpened() {
```

```
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()));
```

```
}
```



Clicking on A Cancelled Order



```
@Test
public void onCancelledOrderClick_checkIfCancelledOrderPagelsOpened() {

    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()));
}
```

View Action



Clicking on A Cancelled Order



```
@Test
public void onCancelledOrderClick_checkIfCancelledOrderPagelsOpened() {

    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

Clicking on a
Cancelled Order



Testing Server Error



@Test

```
public void 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()));
```

```
}
```



It took



45 secs to build
&
Install the app

=

49 secs

+

4 secs to run the **6**
test cases



UI TESTING FOR ANDROID
espresso





When to Use Espresso



For Integration Testing



To Test On Multiple Devices



To Test With Actual Data Sources





X *Not Required*

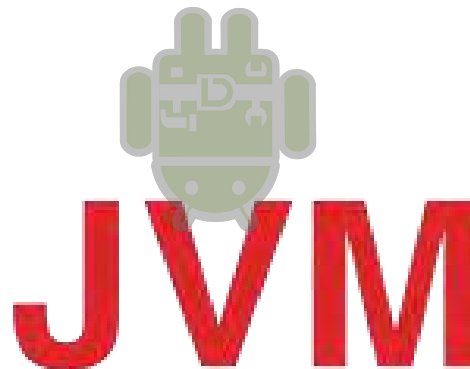


Mocks

Android SDK

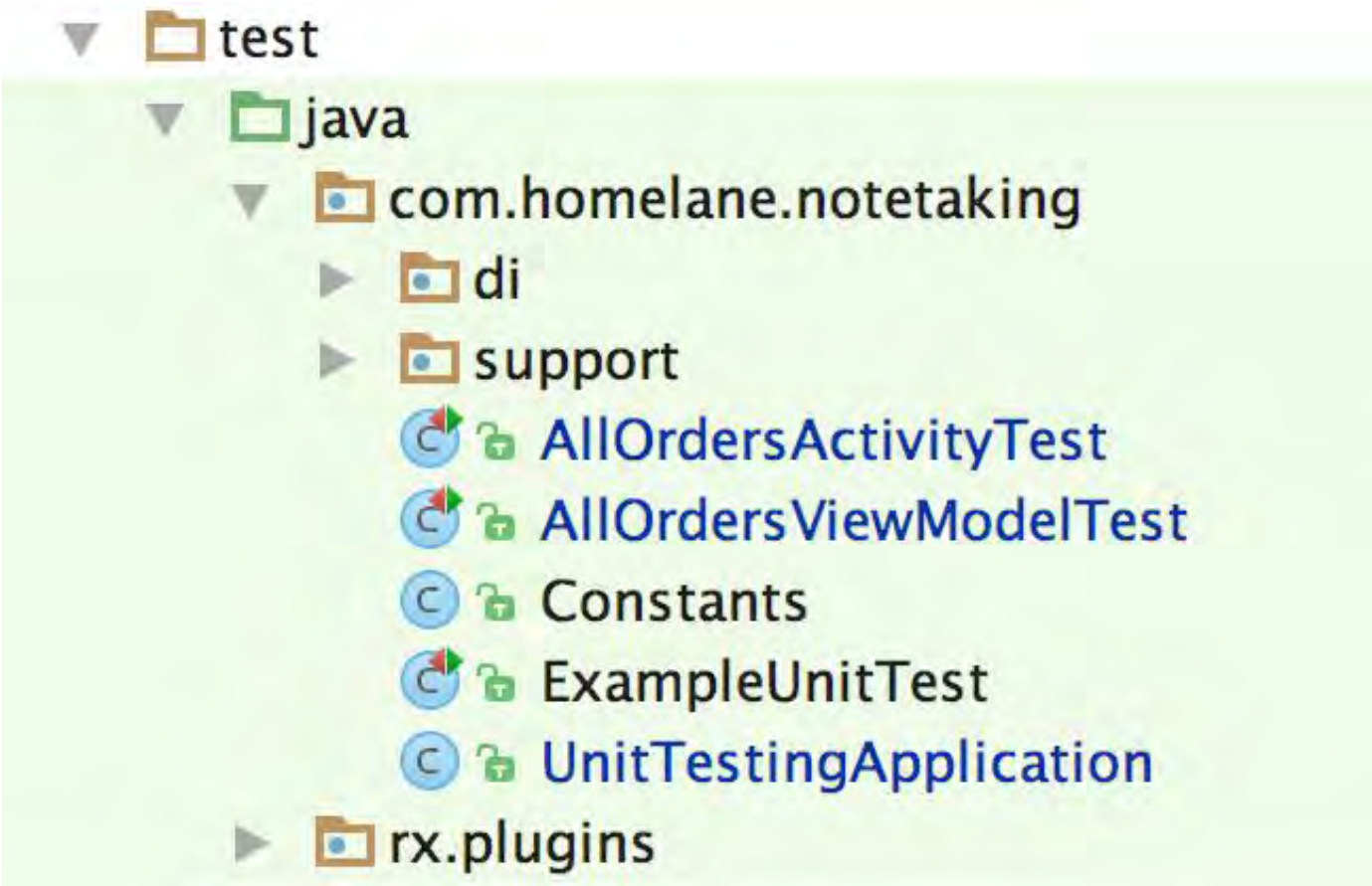
To Run Tests Directly On

JVM





Unit Tests Source Set





Robolectric Test Class



Setting Up
Roblectric Config

```
@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



Initialisation Before Every Test



@RunWith(RobolectricTestRunner.class)

Creating activity

public class AllOrdersActivityTest {

private void reloadOrdersActivity() {

activity = Robolectric.setupActivity(AllOrdersActivity.class);

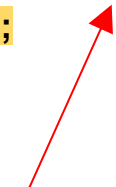
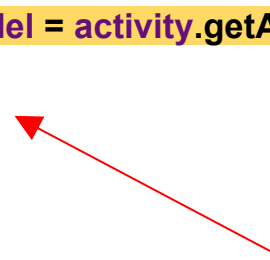
ordersRecyclerView = (RecyclerView) activity.findViewById(R.id.orders_recycler_view);

allOrdersViewModel = activity.getAllOrdersViewModel();

}

}

Referencing Views and View Model





Testing Exception



Creating Order
Observable with
Exception

```
@Test
public void onExceptionErrorWhileFetchingOrders_checkIfSnacBarIsDisplayed() {

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

    reloadOrdersActivity();

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

Checking If Snackbar displays the
correct text or not



Testing labelling Of Order Statuses



@Test

```
public void onOrdersLoaded_checkIfStatusLabelingOfOrderItemsIsCorrect() {
```

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

```
reloadOrdersActivity();
```

```
for(int i = 0; i < OrderLifeCycleConstants.ORDER_STATUSES_ARRAY.length; i++) {
```

```
View itemView = ordersRecyclerView.getChildAt(i);
```

Referencing Views

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

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

```
}  
}
```



Testing labelling Of Order Statuses



@Test

```
public void onOrdersLoaded_checkIfStatusLabelingOfOrderItemsIsCorrect() {
```

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

```
reloadOrdersActivity();
```

```
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



Testing Clicking Of Orders



@Test

```
public void onDeliveryOrderClick_checkIfDeliveryOrderPagelsOpened() {
```

```
FakeOrderDataSource.getInstance().createOrdersObservable(OrderLifecycleConstants.ORDER_STATUSES_ARRAY);
```

```
reloadOrdersActivity();
```

```
ordersRecyclerView.getChildAt(0).performClick();
```

Clicking on an Order Item

```
assertNextActivity(activity, DeliveryActivity.class);
```

Checking If correct Activity has
Opened or Not

```
}
```



It took



18 secs to Shadow
Android Code To
JVM

=

23 secs

+

5 secs to run the **6**
test cases





When to Use Robolectric



For Testing Directly On JVM



Very Useful When App Is Not Well
Architected



Also very Helpful for testing view
properties like colour, style etc.





Testing The View Model



Any Guesses



How much time it took ???



It took

**180 milli
secs**

**To Run The Same
Test Cases**





How Did It Happen???



Lets See...!!!



View Model

```
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() {
```

Accepting a
Repository



Orders Are being
fetched from the
Repository





```
ordersResponseObservable.subscribeOn(Schedulers.io())  
.observeOn(AndroidSchedulers.mainThread())  
.subscribe(new Observer<AllOrdersResponse>() {  
    @Override  
    public void onCompleted() {  
    }
```

```
    @Override  
    public void onError(Throwable e) {
```

```
        dataLoading.set(false);  
        snackbarText.set(exceptionErrorText );  
    }
```

Observables Which
Would directly
Update Views In
The Activity

```
    @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

Layout Files

```
<ProgressBar
  app:layout_constraintTop_toTopOf="@+id/cont_all_orders"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="center"
  android:elevation="2dp"

  android:visibility="@
  { allOrdersViewModel.dataLoading ?
  View.VISIBLE : View.GONE }"
/>
```

Visibility Of This ProgressBar
would depend on
ObservableBoolean variable

ViewModel

```
private final ObservableBoolean dataLoading
= new ObservableBoolean(false);

dataLoading.set(true);

//After getting Response

dataLoading.set(false);
```




The Magic Of MVVM + DataBinding

ViewModel

```
private final ObservableField<String> snackbarText = new  
ObservableField<>();
```

```
@Override  
public void onError(Throwable e) {
```

```
    dataLoading.set(false);
```

```
    snackbarText.set(exceptionErrorText);
```

```
    e.printStackTrace();
```

```
}
```

Activity

```
private Observable.OnPropertyChangedCallback  
snackbarCallback;
```

```
private void setupSnackBar() {
```

```
    snackbarCallback = new Observable.OnPropertyChangedCallback() {
```

```
        @Override
```

```
        public void onPropertyChanged(Observable observable, int  
i) {
```

```
            SnackbarUtils.showSnackBar(activityAllOrdersBinding.mai  
n Cord, allOrdersViewModel.getSnackbarText());
```

```
        }
```

```
    };
```

```
allOrdersViewModel.snackbarText.addOnPropertyChangedCallback(sn  
ackbarCallback);
```

```
}
```

Whenever ObservableField<String>
changes, a Snackbar is shown in the
Activity with the updated value



That Means I can Directly Test The
View Model

And See Whether The Business Logic Works
Fine Or Not





YeSsssss

And Since The **View Model** is Simply a **Java Class**

Without Any Android Specific Code



The Tests Run Very

Fast

On



JVM



Testing ProgressBar

@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());
```

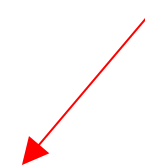
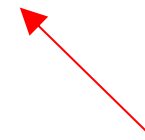
```
}
```

```
private AllOrdersViewModel constructAndGetAllOrdersViewModel(String errorText) {
```

```
    return new AllOrdersViewModel(OrdersRepository.getInstance(FakeOrderDataSource.getInstance()), errorText);
```

```
}
```

Instantiating The View Model





Testing ProgressBar

@Test

```
public void afterSuccessFullOrdersLoading_CheckIfProgressBarsNotDisplayed() {
```

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

```
    AllOrdersViewModel allOrdersViewModel = constructAndGetAllOrdersViewModel  
(EXCEPTION_ERROR_SNACKBAR_TEXT);
```

```
    allOrdersViewModel.loadOrders();
```

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

```
}
```

Loading the orders and checking
that dataloading
ObservableBoolean is false or not



Testing Order Count

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

    AllOrdersViewModel allOrdersViewModel = constructAndGetAllOrdersViewModel
    (EXCEPTION_ERROR_SNACKBAR_TEXT);

    allOrdersViewModel.loadOrders();

    assertEquals(3, allOrdersViewModel.getOrdersList().size());
}
```

Instantiating The View
Model

A red arrow originates from the text 'Instantiating The View Model' and points to the line of code: **AllOrdersViewModel allOrdersViewModel = constructAndGetAllOrdersViewModel (EXCEPTION_ERROR_SNACKBAR_TEXT);**



Testing Order Count

@Test

```
public void onOrdersFetched_CheckIfOrdreCountIsCorrect() {
```

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

```
AllOrdersViewModel allOrdersViewModel = constructAndGetAllOrdersViewModel  
(EXCEPTION_ERROR_SNACKBAR_TEXT);
```

```
allOrdersViewModel.loadOrders();
```

```
assertEquals( 3, allOrdersViewModel.getOrdersList().size() );
```

```
}
```

Loading the orders and checking
that orderList
ObservableList<Order>
Count is 3 or not



Testing On Multiple Devices

[Bonus]





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



Devices		Operating Systems	
samsung	55%	5	85%
OPPO	41%	4	15%



Testing On Multiple Devices



Fire Base Test Labs



SAUCELABS



Robo Tests



Max depth

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

50



Randomly Tests App's UI

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

Can Supply Inputs for EditTexts

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



Save template 4 devices , 2 orientations , 1 locale

START 8 TESTS

Can Choose Maximum Depth of Test Traversal



Run From Android Studio



Deployment Target Options

Target:

Matrix configuration:

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



Passed 7/17/17, 2:01 AM 18 sec English Portrait [VIEW SOURCE FILES](#)

TEST CASES LOGS **VIDEOS** PERFORMANCE

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. Less No. Of Devices

1. Supports Only Android Instrumentation Tests And Robo Tests

1. Network Speed Throttling Not Supported



X Not Supported





AWS Device Farm



**Supports Different
Types Of Tests**

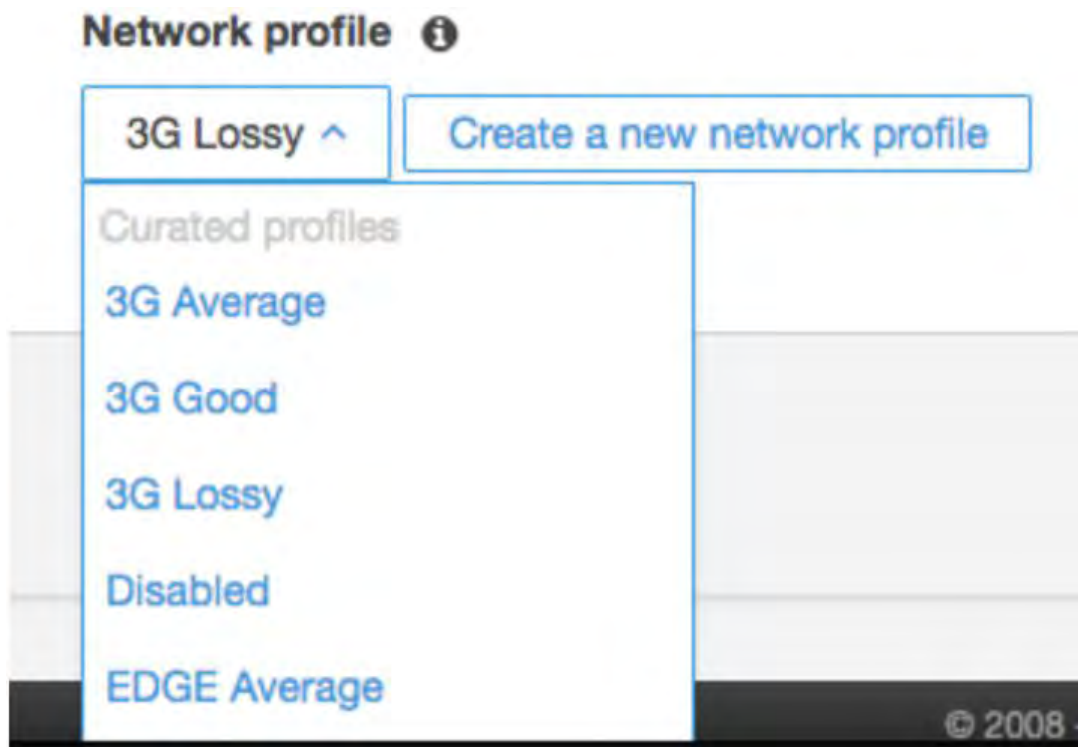
- Built-in: Explorer
- Built-in: Fuzz
- Appium Java JUnit**
- Appium Java TestNG
- Appium Python
- Calabash
- Instrumentation
- UI Automator



AWS Device Farm



Testing At Different
Network Speeds





AWS Device Farm



More Devices To Test On

<input type="radio"/>	HTC One M8 (AT&T)	Android	4.4.4	Phone
<input type="radio"/>	LG G Flex (AT&T)	Android	4.2.2	Phone
<input type="radio"/>	LG G2 (AT&T)	Android	4.4.2	Phone
<input type="radio"/>	LG Optimus L70 (MetroPCS)	Android	4.4.2	Phone
<input type="radio"/>	Motorola DROID Ultra (Verizon)	Android	4.4.4	Phone
<input type="radio"/>	Samsung Galaxy Note 3 (AT&T)	Android	4.4.2	Phone
<input type="radio"/>	Samsung Galaxy Note 3 (Verizon)	Android	4.4.4	Phone
<input type="radio"/>	Samsung Galaxy Note 4 (AT&T)	Android	5.0.1	Phone
<input type="radio"/>	Samsung Galaxy Note 4 (Verizon)	Android	5.0.1	Phone
<input type="radio"/>	Samsung Galaxy S3 (T-Mobile)	Android	4.3	Phone
<input type="radio"/>	Samsung Galaxy S3 (Verizon)	Android	4.4.2	Phone
<input type="radio"/>	Samsung Galaxy S3 LTE (T-Mobile)	Android	4.3	Phone
<input type="radio"/>	Samsung Galaxy S3 Mini (AT&T)	Android	4.4.2	Phone



The Only Con



Not Able To Run Specific TestNG Test Suites



Sauce Labs



Robotium



appium



UI TESTING FOR ANDROID
espresso

Supports Different Testing Frameworks



Sauce Labs



Sauce Labs Acquired TestObject to enable testing on
Real Devices



Sauce Labs



```
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

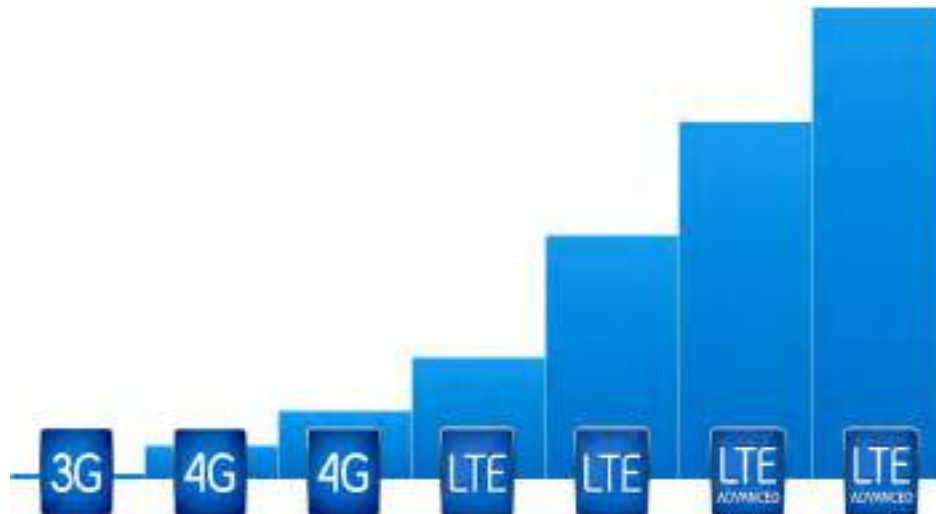


Cons



Network Speed Throttling Is Not Supported

X Not Supported



Oh Lord Of Test Driven Development



Cast Your Light Upon Us

For The Release Is Critical



And Prone To Bugs