Android: Beyond basics

Ing. David Vávra, Step Up Labs
MU Brno, March 2016

About me

- Internship at Google London 2011
- Graduated ČVUT FEL in 2012
- Master thesis: Settle Up
- 2012-2014 Inmite
- 2014-2016 Avast
- 2015 GDE for Android
- 2016 Step Up Labs

http://twitter.com/destil

http://google.com/+DavidVávra

https://medium.com/@david.vavra

Who is this talk for?

- Students with basic Java knowledge
- Android beginners
- Intermediate Android devs
- iOS and WP devs who are interested about Android

Agenda

- Motivation & basics recap
    - QA & Break
- Creating a Play Store-ready app
    - QA & Break
- Professional Android development
    - QA

www.slido.com
#brno

Motivation & Basics Recap

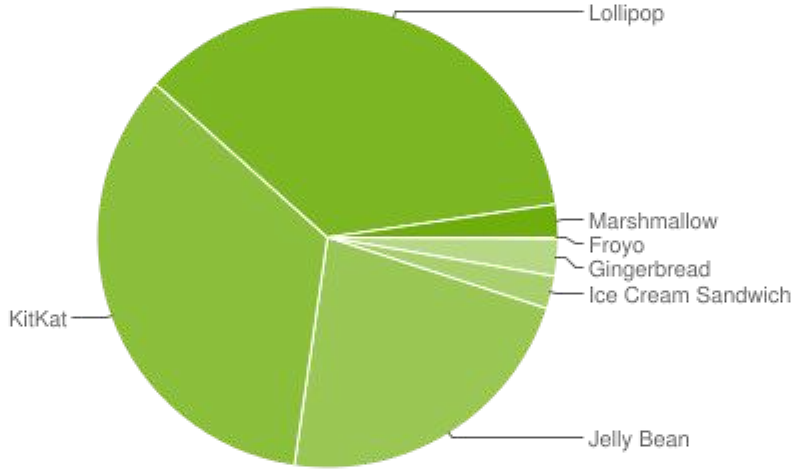
Android is ...

- Linux-based OS for various devices
- Open-source (http://source.android.com)

Some history

- 2003, Android inc., digital cameras
- 2005, acquired by Google
- 2007 iPhone
- Sep 2008, the first Android phone
  - T-Mobile G1
- May 2010, Froyo (Android 2.2)
- Feb 2011, Honeycomb (Android 3.0)
- Oct 2011, Ice Cream Sandwich (4.0)
- July 2012, Jelly Bean (Android 4.1)
- July 2013, Jelly Bean (Android 4.3)
- Oct 2013, KitKat (Android 4.4)
- June 2014, Lollipop (Android 5.0)
- September 2015, Marshmallow (6.0)
- March 2016, N (6.1?)

Android today



Android today

- global marketshare **78.4%**
- 1.5 million devices daily activated
- tablet marketshare 36.5%
- >1.7 million apps in Play Store
- $1.8 billion from app sales in 2014

- Phones
- Tablets
- Android Wear
- Android TV
- Android Auto
- Project Tango
- Brillo
- (Google Glass)

Bright side of Android

- Tons of users!
- Almost instant publishing
- No yearly fees, no need for Mac
- Open-source, built to handle various factors
- Developer freedom
- Support library & Google Play services
- Support from Google
- Nexus & Motorola devices

Dark side of Android

- fragmentation, slow upgrades, manufacturer changes
- Android users less likely to pay
- low-end devices
- lower quality apps in Google Play, malware
- no Play Store in China
- API is getting restricted

Success Stories

- Urbandroid
- Tomáš Hubálek
- TappyTaps
- Inmite + Avast
- Ackee
- STRV
- and many more

Development options

- App-like mobile web
- Other language frameworks (Xamarin, Scala, Kotlin…)
- C-based frameworks (Unity)
- WebView-based frameworks (PhoneGap)
- Native

Native development

- programming in Java
    - Java 6 (<Kitkat)
    - Java 7 (KitKat)
    - Java 8 (N)
- native apps possible via NDK (C++)
- Android Studio (IntelliJ Idea)
    - Windows, Linux, Mac OS X

https://www.youtube.com/watch?v=Z98hXV9GmzY

Recap: Building blocks

- Gradle (Groovy, APKs, flavors, dependencies via Maven repos)
- AndroidManifest.xml (components, API level, permissions, …)
- Resources (bitmaps, vectors, state lists, strings, layouts)
- Activity (screen, contains Fragments and Views)
- Service (long-running background tasks, notification)
- Content provider (share data between apps)
- Broadcast receiver (system-wide or custom events)
- Intents (glue between components, data message)

Recap: Building the UI

- Activity contains Fragments
- Fragments contains Layouts from resources
  - LinearLayout, RelativeLayout, FrameLayout etc.
- Layouts contain Views
  - Button, TextView, EditText, RadioButton, WebView, ...
- List of items uses Adapter pattern to bind data and recyclers views
  - ListView, GridView, Spinner, RecyclerView

Recap: Resources

- Resource qualifiers are powerful
  - drawable-mdpi
  - values-cs
  - layout-sw640dp
  - Drawable-hdpi-v11
- Density-independent units
  - dp
  - sp (for fonts)
  - never use px

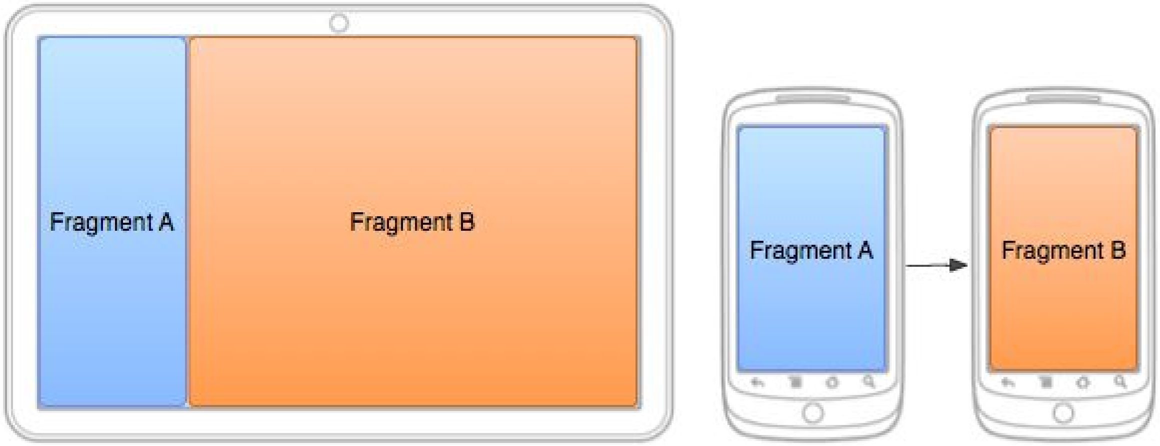
QA & Break

www.slido.com
 #brno

Creating a Play Store-ready app

Fragments

- Created for supporting tablets
- Complicated API & lifecycle
- Allow for one-Activity app
- Sometimes required (ViewPager, TV apps)



Dialogs

- Do you really need to interrupt the user with dialog?
- Hard to style consistently
- They close on rotation
- Solution: https://github.com/avast/android-styled-dialogs

```
SimpleDialogFragment.createBuilder(this, getSupportFragmentManager())
.setTitle(R.string.title)
.setMessage(R.string.message)
.setPositiveButtonText(R.string.positive_button)
.setNegativeButtonText(R.string.negative_button)
.show();
```

Asynchronous calls & rotation

It is tricky, because:

Once a configuration change (such as rotation) happens, Activity instance is killed and recreated. If you have a reference to the old Activity (for example in background thread), you create a memory leak (and bugs).

Hacks people do to workaround it:

- `android:screenOrientation="portrait"`
    - How about language, font, keyboard change? + Android N split mode
- `android:configChanges="orientation"`
    - Same layout in all configurations

Async options

- Java Thread
  - Know nothing about Android, lot of boilerplate
- AsyncTask
  - Simple API, widely (mis)used
  - Inconsistent behaviour on API levels
  - It’s not ties to the Activity lifecycle = creates memory leaks
- IntentService
  - Good option for “do something quick on the background”
  - Doesn’t tie well with the UI, lot of code to do that (properly).
- RxJava
  - Steep learning curve, but robust
  - More about that in last part

Loaders

- Designed to solve this problem
- Part of support library
- Tied with Activity lifecycle
- Good for “loading stuff for this screen”, not for “do stuff after user clicked to something”
- Doesn’t load stuff again after rotation, uses cached stuff

https://medium.com/google-developers/making-loading-data-on-android-lifecycle-aware-897e12760832

```
public static class JsonAsyncTaskLoader extends
   AsyncTaskLoader<List<String>> {
 private List<String> mData;
 public JsonAsyncTaskLoader(Context context) {
   super(context);
 }
 @Override
 protected void onStartLoading() {
   if (mData != null) {
     deliverResult(mData);
   } else {
     forceLoad();
   }
 }
 @Override
 public List<String> loadInBackground() {
   // download and parse JSON
   List<String> data = new ArrayList<>();
   return data;
 }
 @Override
 public void deliverResult(List<String> data) {
   mData = data;
   super.deliverResult(data);
 }
}
```

Saving data

- SharedPreferences
  - simple key-value data like settings
- Sqlite database
  - structured data, a lot of boilerplate
- ContentProvider
  - wrapper around Sqlite (usually), use it only if you wish to share stuff with other apps
- Save files to filesystem
  - Good for files, lot of boilerplace for structured data, don’t rely on SD card
- Save data to the cloud
- ORMs
  - Reduce boilerplate, less flexible, OrmLite, GreenDAO
- Firebase, Realm - in the last part

Notifications

- Use NotificationCompat from support library (not Notification)
- Quite robust API which is also used for Android Wear and Android TV
- Uses Builder pattern:

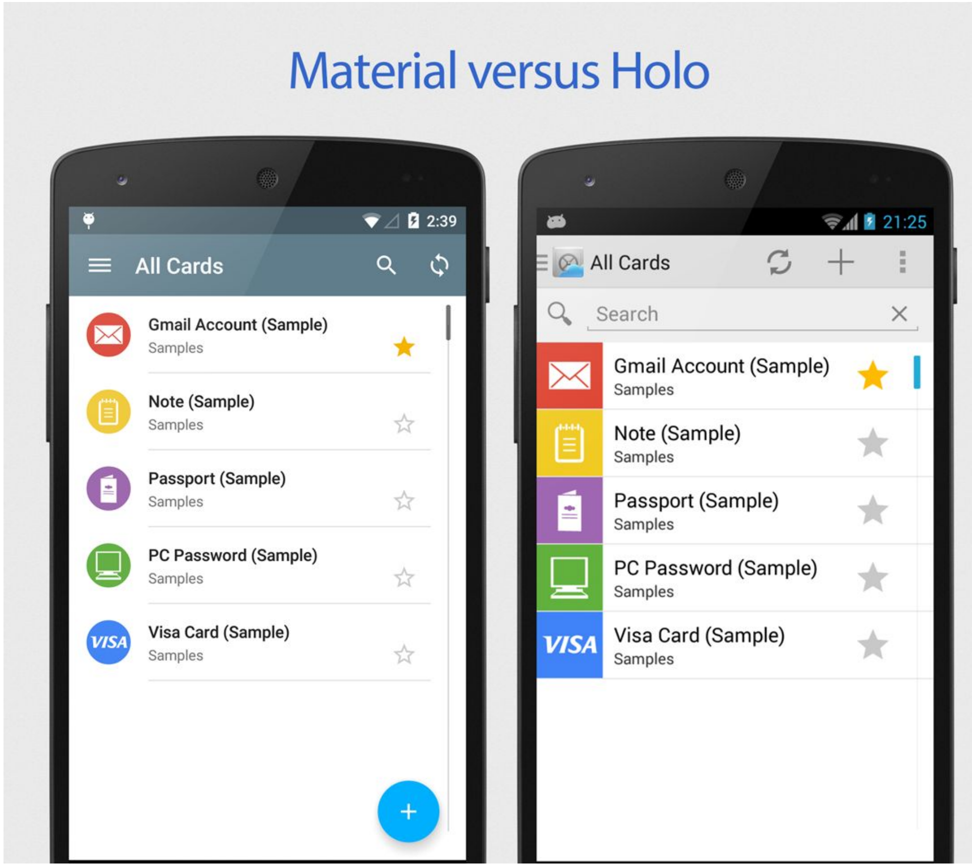
```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this).
setSmallIcon(R.drawable.notification_icon).setContentTitle("My notification").
setContentText("Hello World!");
Intent resultIntent = new Intent(this, ResultActivity.class);
PendingIntent resultPendingIntent =  PendingIntent.getActivity(context,
resultIntent, 0, PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());
```

Useful third-party libs

- Support library (Google)
  - Fragments, Notifications, ViewPager, DrawerLayout, Loaders, Material design, CardView, Google Cast, RecyclerView, Leanback UI (for TVs), Custom Tabs, Percent layouts, …
- Picasso (Square) or Glide (Bumbtech)
  - `Glide.with(this).load("http://goo.gl/gEgYUd").into(imageView);`
- ButterKnife (Jake Wharton)
  - @Inject(R.id.text) TextView mText;
- Retrofit (Square)
  - Working with any REST API
  - Simple definition of the API with @annotations
  - does parsing and networking for you
- Crashlytics (Fabric)

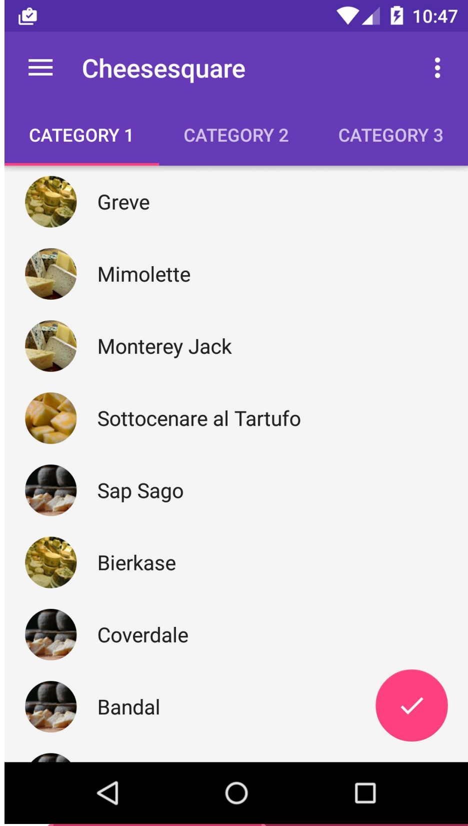
Material design

- Beautiful design which behaves like "materials" = intuitive
- What materials? Mostly paper and ink
- Contains meaningful animations which guide the user
- Brings elevation (done by shadow)
- Support design library does a lot for you
- All material icons are open-source
- Defines a color palette, usually bold colors
- Puts content first, your brand is done by colors, not by logos
- Developers don't need designers (that much)
- More: https://www.google.com/design/spec

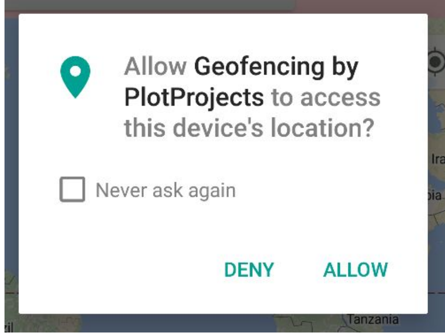


Useful UX patterns

- CardView
  - Better for more data in a list
- Left Drawer + “Hamburger” menu
  - If you have a bigger app with more separate features
- Floating Action Button (FAB)
  - For primary action in your app
- Tabs + ViewPager
  - For sections or categories
- Pull to refresh
  - For updating data
- Delete-undo
  - Goodbye “Are you sure to delete this?”



Permissions


Allow **Geofencing by PlotProjects** to access this device's location?

Never ask again

DENY ALLOW

- You need to list them in AndroidManifest.xml
- From Marshmallow you need to ask some of them in runtime
- User on Marshmallow can remove permissions also to old apps
- That’s why you should update to runtime permissions
- Don’t request something you don’t need (beware about libs)
- UX
  - Ask first without explanation
  - If user denies, explain why you need it
  - Ask for permissions when you need them, not in the beginning
  - Disable only parts of your app if you are missing permission

Publishing to Play Store

- $25 for life
- Release checklist
  - Test the app yourself
  - Prepare screenshots for all devices you support (phones, tablets, watches, TV)
  - Prepare one-liner and description at least in one language
  - Prepare high-res icon (512x512)
  - Prepare promotional graphic (1024x500)
  - Support e-mail (Google Group works well)
  - Publish APK to alpha or beta first
    - People can join either via link or you can invite specific testers
  - Once you are confident, publish to production
  - Watch ratings and stats
  - Remember that there are 1.7M apps in the store

QA & Break

www.slido.com
#brno

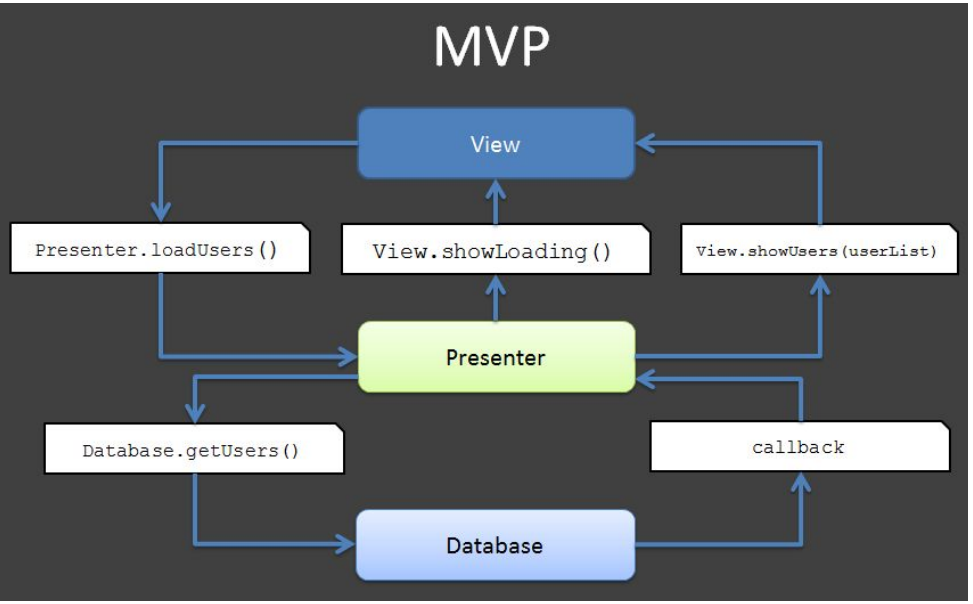
Professional Android development

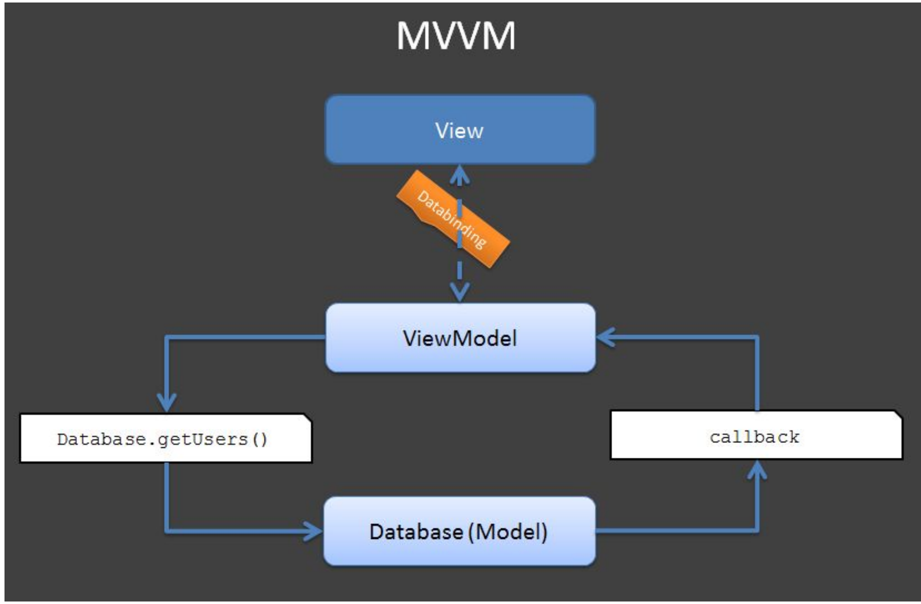
Android architecture

- Why?
  - Big project structure gets messy and hard to debug
  - Activities and Android code mixed with app logic is hard to test. (Activity=ViewController)
- MVP
  - Model - database, network resources etc.
  - View - Activities/Fragments which only render stuff and listen for user input and call Presenter
  - Presenter
    - 1:1 class for each View.
    - Handles all communication between View and Model
    - Prepares data in minimal form for the View
    - It’s easily testable, doesn’t have any Android dependencies

https://labs.ribot.co.uk/android-application-architecture-8b6e34acda65

MVP & MVVM





http://tech.vg.no/2015/07/17/android-databinding-goodbye-presenter-hello-viewmodel/

Data binding

```xml
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.
com/apk/res/android">
   <data>
       <variable name="user" type="com.example.User"/>
   </data>
   <LinearLayout
       android:orientation="vertical"
       android:layout_width="match_parent"
       android:layout_height="match_parent">
       <TextView android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="@{user.firstName}"/>
       <TextView android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="@{user.lastName}"/>
   </LinearLayout>
</layout>
```

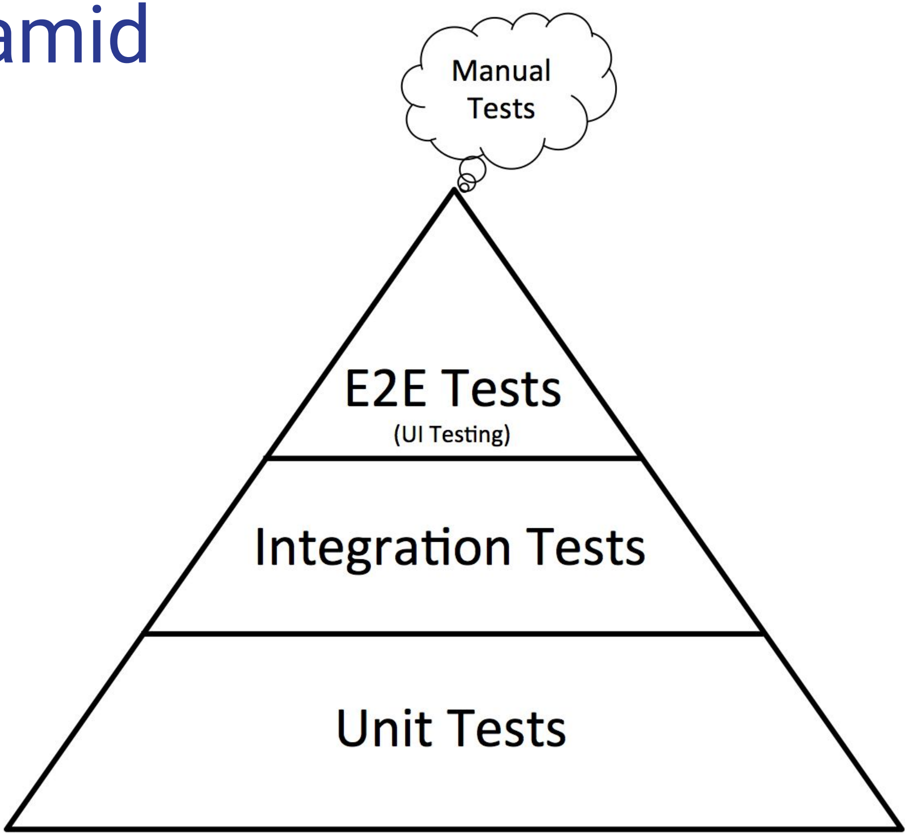
```java
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   MainActivityBinding binding = DataBindingUtil.
setContentView(this, R.layout.main_activity);
   User user = new User("Test", "User");
   binding.setUser(user);
}
```

+ViewModel handling user events and display logic

Testing

- Why?
  - Catch bugs as early as possible, TDD, regression, automation, piece of mind, architecture
- Why not?
  - Harder refactoring, slows down prototypes, find balance
- Unit/Integration testing
  - Local Unit Tests - no Android dependencies - JUnit4
  - Instrumentation Unit Tests - with Android dependencies - runs on device or emulator
  - Roboelectric - Mocks a lot of Android dependencies, can be run locally
- E2E testing
  - Expresso - UI testing within your app
  - UI Automator - UI testing within whole system
  - Monkey - Testing based on random input

Testing pyramid



JUnit4 & Roboelectric

```
@Test
public void
multiplicationOfZeroIntegersShouldReturnZero() {
    // MyClass is tested
    MyClass tester = new MyClass();
    // assert statements
    assertEquals("10 x 0 must be 0", 0,
tester.multiply(10, 0));
    assertEquals("0 x 10 must be 0", 0, tester.
multiply(0, 10));
    assertEquals("0 x 0 must be 0", 0, tester.
multiply(0, 0));
 }
```

```
@Test
public void
clickingButton_shouldChangeResultsViewText() throws
Exception {
    MyActivity activity = Robolectric.setupActivity
(MyActivity.class);
    Button button = (Button) activity.findViewById(R.
id.button);
    TextView results = (TextView) activity.
findViewById(R.id.results);
    button.performClick();
 assertThat(results.getText().toString())
.isEqualTo("Robolectric Rocks!");
  }
```

Espresso & UI Automator

```java
@Test
public void changeText_sameActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput))
           .perform(typeText(mStringToBetyped),
            closeSoftKeyboard());
    onView(withId(R.id.changeTextBt)).perform(click());
    // Check that the text was changed.
    onView(withId(R.id.textToBeChanged))
           .check(matches(withText(mStringToBetyped)));
}
```

```java
public void testTwoPlusThreeEqualsFive() {
    mDevice.findObject(new UiSelector()
.packageName(CALC_PACKAGE).resourceId("two")).click();
    mDevice.findObject(new UiSelector()                 .
packageName(CALC_PACKAGE).resourceId("plus")).click();
    mDevice.findObject(new UiSelector()
.packageName(CALC_PACKAGE).resourceId("three")).click();
    mDevice.findObject(new UiSelector()
.packageName(CALC_PACKAGE).resourceId("equals")).click();

    // Verify the result = 5
    UiObject result = mDevice.findObject(By.res
(CALC_PACKAGE, "result"));
    assertEquals("5", result.getText());
}
```

Dependency injection

- Creates objects for you and handles dependencies (constructor parameters) of other objects
- You can just @Inject something anywhere and don't care how it was created and what it needed for creation
- Most used library: Dagger 2
- Good for keeping @Singleton instances
- Injected objects can be easily mocked in tests
- http://google.github.io/dagger/

```
public abstract class BaseActivity extends
Activity {
  @Inject Settings settings;

  @Override
  protected void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    App.getComponent().inject(this);
  }
}
```

Contest time

Google Play Services

- Automatically updated APIs from Google
- Google fights fragmentation
- APIs
    - Location
    - Maps
    - Activity recognition
    - Google Sign in
    - Google Drive
    - Admob
    - Analytics
    - Google Fit

JobScheduler

- Best way to schedule work for later
- More flexible than AlarmManager
- Backported via Google Play Services (GCM Network Manager)
- Conditions for jobs: network - metered/unmetered, charging state
- Persists across restarts
- Automatic retry with exponential backoff
- OneOff & Periodic

```
OneoffTask task = new OneoffTask.Builder()
        .setService(MyTaskService.class)
        .setTag(TASK_TAG_WIFI)
        .setExecutionWindow(0L, 3600L)
        .setRequiredNetwork(
              Task.NETWORK_STATE_UNMETERED)
        .build();
mGcmNetworkManager.schedule(task);
```

Animations

- View Animations
  - Older, only for Views, animations defined in xml, simple
- Property Animations
  - >Honecomb, general, can animate anything on any property
  - Duration, time interpolation, repeat count, behavior
  - `button.animate().setDuration(1200).alpha(0.5f).x(250);`
- Drawable animation
  - Frame by frame
- Drawing on Canvas
- OpenGL

RxJava

- Reactive programming - based on Observer pattern
- Functional programming
- Helps with complex async calls
- Retrofit has Rx bindings
- Steep learning curve
- Android extensions - RxAndroid
- Building blocks - Observables and Subscribers

```
api.login(new Callback<ResponseBody>() {
    @Override
    public void success(final ResponseBody body,
final Response response) {
        api.getUserStatus(new
Callback<UserStatus>() {
            @Override
            public void success(final UserStatus
status, final Response response) {
                      // update UI according to
user state
                  }
                  ...
//RxJava
eventAPI.login()
.flatMap(status -> api.getUserStatus())
.subscribe(onComplete, onError);
```

Realm

- Cross-platform database
- Replacement for Sqlite
- Fast and modern
- Works with objects
- https://realm.io/

```
public class Person extends RealmObject {
    private String name;
    private RealmList<Dog> dogs;
}

realm.beginTransaction();
Dog mydog = realm.createObject(Dog.class);
Person person = realm.createObject(Person.
class);
person.setName("Tim");
person.getDogs().add(mydog);
realm.commitTransaction();
```

Firebase

- JSON database on the server
- Realtime - keeps connection to the server when you app is active
- Removes the need for backend for most apps
- Handles synchronization
- Handles offline
- Handles authentication to Facebook, Twitter, Google
- https://www.firebase.com/

```
// Create a connection to your Firebase database
Firebase ref = new Firebase("https://<YOUR-FIREBASE-APP>.firebaseio.com");

// Save data
ref.setValue("Alex Wolfe");

// Listen for realtime changes
ref.addValueEventListener(new ValueEventListener() {
  @Override
  public void onDataChange(DataSnapshot snap) {
      System.out.println(snap.getName() + " -> " + snap.getValue());
  }
  @Override public void onCancelled(FirebaseError error) { }
});
```

Kotlin

- New language from JetBrains
- JVM-based, fully compatible with Java
- Full support in Android Studio
- Stable, concise, modern
- Null safe
- Lambdas and other functional stuff
- Extension functions

```
fun Fragment.toast(message: CharSequence,
duration: Int = Toast.LENGTH_SHORT) {
    Toast.makeText(getActivity(), message,
duration).show()
}


view.setOnClickListener { toast("Hello
world!") }
```

Proguard

- Why?
  - Smaller APK size
  - Harder decompilation
  - Staying in 65k method limit

```
buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android.txt'),
        'proguard-rules.pro'
    }
}
```

Analytics

- Why?
  - Best decisions are backed by data
  - You can detect UX problems and prioritize features
- Google Analytics - free and robust, part of Google Play Services
- Automatically tracks Activities and time spent there
- Real-time view
- You can track events with data - clicks, user actions etc.
- Generates lot of graphs - like funnel which shows how your users go through the app and where they leave

Final QA

This slides:
http://bit.ly/android-brno

http://twitter.com/destil

http://google.com/+DavidVávra

https://medium.com/@david.vavra